Chapter 13
Window and Display Features

Chapter Table of Contents

Chapter 13
Window and Display Features

Overview

The dynamic nature of IML gives you the ability to create windows on your display for full-screen data entry or menuing. Using the WINDOW statement, you can define a window, its fields, and its attributes. Using the DISPLAY statement, you can display a window and await data entry.

These statements are similar in form and function to the corresponding statements in the SAS DATA step. The specification of fields in the WINDOW or DISPLAY statements is similar to the specifications used in the INPUT and PUT statements. Using these statements you can write applications that behave similarly to other full-screen facilities in the SAS System, such as the BUILD procedure in SAS/AF software and the FSEDIT procedure in SAS/FSP software.

Creating a Display Window for Data Entry

Suppose that your application is a data entry system for a mailing list. You want to create a data set called MAILLIST by prompting the user with a window that displays all the entry fields. You want the data entry window to look as follows.

```
+--MAILLIST----------------------------------------------+
| Command==>                                             |
|                                                        |
|                                                        |
| NAME:                                                  |
| ADDRESS:                                               |
| CITY: STATE: ZIP:                                      |
| PHONE:                                                 |
|                                                        |
+--------------------------------------------------------+
```

The process for creating a display window for this application consists of

- initializing the variables
- creating a SAS data set
- defining a module for collecting data that
  1. defines a window
  2. defines the data fields
  3. defines a loop for collecting data
  4. provides an exit from the loop

- executing the data-collecting routine

The whole system can be implemented with the following code to define modules INITIAL and MAILGET:

```
/*   module to initialize the variables                    */
/*                                                         */
start initial;
   name='        ';
   addr='                    ';
   city='                    ';
   state='  ';
   zip='    ';
   phone='        ';
finish initial;
```

This defines a module named INITIAL that initializes the variables you want to collect. The initialization sets the string length for the character fields. You need to do this prior to creating your data set.

Now define a module for collecting the data:

```
/* module to collect data                                  */
/*                                                         */
start mailget;
/* define the window                                       */
   window maillist cmndline=cmnd msgline=msg
      group=addr
      #2 " NAME: " name
      #3 " ADDRESS:" addr
      #4 " CITY: " city +2 "STATE: " state +2 "ZIP: " zip
      #5 " PHONE: " phone;
/*                                                         */
/* collect addresses until the user enters exit            */
/*                                                         */
   do until(cmnd="EXIT");
      run initial;
      msg="ENTER SUBMIT TO APPEND OBSERVATION, EXIT TO END";
/*                                                         */
/* loop until user types submit or exit                    */
/*                                                         */
      do until(cmnd="SUBMIT"|cmnd="EXIT");
         display maillist.addr;
      end;
      if cmnd="SUBMIT" then append;
   end;
   window close=maillist;
finish mailget;
/* initialize variables                                    */
run initial;
/* create the new data set                                 */
create maillist var{name addr city state zip phone};
/* collect data                                            */
```

```
   run mailget;
   /* close the new data set                                  */
   close maillist;
```

In the module MAILGET, the WINDOW statement creates a window named MAILLIST with a group of fields (the group is named ADDR) presenting data fields for data entry. The program sends messages to the window through the MSGLINE= variable MSG. The program receives commands you enter through the CMNDLINE= variable CMND.

You can enter data into the fields after each prompt field. After you are finished with the entry, press a key defined as SUBMIT, or type SUBMIT in the command field. The data are appended to the data set MAILLIST. When data entry is complete, enter EXIT in the command field. If you enter a command other than SUBMIT, EXIT, or a valid display manager command in the command field, you get this message on the message line:

```
ENTER SUBMIT TO APPEND OBSERVATION, EXIT TO END.
```

Using the WINDOW Statement

You use the WINDOW statement to define a window, its fields, and its attributes. The general form of the WINDOW statement is

WINDOW <**CLOSE=**> *window-name* < *window-options* >
  <**GROUP=***group-name-1 field-specs*

  < . . .**GROUP=***group-name-n field-specs* >>**;**

The following options can be used with the WINDOW statement:

CLOSE=
is used only when you want to close the window.

window-name
is a valid SAS name for the window. This name is displayed in the upper left border of the window.

window-options
control the size, position, and other attributes of the window. You can change the attributes interactively with window commands such as WGROW, WDEF, WSHRINK, and COLOR. These options are described in the next section.

GROUP=*group-name*
starts a repeating sequence of groups of fields defined for the window. The *group-name* is a valid SAS variable name used to identify a group of fields in a DISPLAY statement that occurs later in the program.

field-specs
: is a sequence of field specifications made up of positionals, field operands, formats, and options. These are described in "Field Specifications" later in this chapter.

Window Options

Window-options control the attributes of the window. The following options are valid in the WINDOW statement:

CMNDLINE=*name*
names a character variable in which the command line entered by the user is stored.

COLOR=*operand*
specifies the background color for the window. The *operand* can be either a quoted character literal or the name of a character variable containing the color. The valid values are BLACK, GREEN, MAGENTA, RED, CYAN, GRAY, and BLUE. This default is BLACK.

COLUMNS=*operand*
specifies the starting number of columns of the window. The *operand* can be either a literal number, a variable name, or an expression in parentheses. The default is 78.

ICOLUMN=*operand*
specifies the initial column position of the window on the display screen. The *operand* can be either a literal number or a variable name. The default is column 1.

IROW=*operand*
specifies the initial row position of the window on the display screen. The *operand* can be either a literal number or a variable name. The default is row 1.

MSGLINE=*operand*
specifies the message to be displayed on the standard message line when the window is made active. The *operand* is a quoted character literal or the name of a character variable containing the message.

ROWS=*operand*
determines the starting number of rows of the window. The *operand* is either a literal number, the name of a variable containing the number, or an expression in parentheses yielding the number. The default is 23 rows.

Field Specifications

Both the WINDOW and DISPLAY statements allow field specifications. Field specifications have the general form

<positionals> field-operand <format> <field-options>

Positionals

The *positionals* are directives specifying the position on the screen in which to begin the field. There are four kinds of positionals, any number of which are allowed for each field operand. Positionals are the following:

# *operand*	specifies the row position; that is, it moves the current position to column 1 of the specified line. The *operand* is either a number, a variable name, or an expression in parentheses. The expression must evaluate to a positive number.
/	instructs IML to go to column 1 of the next row.
@ *operand*	specifies the column position. The *operand* is either a number, a variable name, or an expression in parentheses. The @ directive should come after the pound sign (#) positional, if it is specified.
+ *operand*	instructs IML to skip columns. The *operand* is either a number, a variable name, or an expression in parentheses.

Field 0perands

The *field-operand* specifies what goes in the field. It is either a character literal in quotes or the name of a character variable.

Formats

The *format* is the format used for display, for the value, and also as the informat applied to entered values. If no format is specified, the standard numeric or character format is used.

Field Options

The *field-options* specify the attributes of the field as follows:

PROTECT=YES
P=YES

specifies that the field is protected; that is, you cannot enter values in the field. If the field operand is a literal, it is already protected.

COLOR=*operand*

specifies the color of the field. The *operand* can be either a literal character value in quotes, a variable name, or an expression in parentheses. The colors available are WHITE, BLACK, GREEN, MAGENTA, RED, YELLOW, CYAN, GRAY, and BLUE. The default is BLUE. Note that the color specification is different from that of the corresponding DATA step value because it is an operand rather than a name without quotes.

Using the DISPLAY Statement

After you have opened a window with the WINDOW statement, you can use the DISPLAY statement to display the fields in the window.

The DISPLAY statement specifies a list of groups to be displayed. Each group is separated from the next by a comma. The general form of the DISPLAY statement is as follows.

DISPLAY <*group-spec-1 group-options,*< . . . , *group-spec-n group-options*>>**;**

Group Specifications

The group specification names a group, either a compound name of the form *windowname.groupname* or a *windowname* followed by a group defined by fields and enclosed in parentheses. For example, you can specify *windowname.groupname* or *windowname*(*field-specs*), where *field-specs* are as defined earlier for the WINDOW statement.

In the example, you used the statement

```
display maillist.addr;
```

to display the window MAILLIST and the group ADDR.

Group Options

The *group-options* can be any of the following:

BELL
rings the bell, sounds the alarm, or beeps the speaker at your workstation when the window is displayed.

NOINPUT
requests that the group be displayed with all the fields protected so that no data entry can be done.

REPEAT
specifies that the group be repeated for each element of the matrices specified as *field-operands*. See "Repeating Fields" later in this chapter.

Details about Windows

The following sections discuss some of the ideas behind windows.

The Number and Position of Windows

You can have any number of windows. They can overlap each other or be disjoint. Each window behaves independently from the others. You can specify the starting size, position, and color of the window when you create it. Each window responds to display manager commands so that it can be moved, sized, popped, or changed in color dynamically by the user.

You can list all active windows in a session by using the SHOW WINDOWS command. This makes it easy to keep track of multiple windows.

Windows and the Display Surface

A window is really a viewport into a display. The display can be larger or smaller than the window. If the display is larger than the window, you can use scrolling commands to move the surface under the window (or equivalently, move the window over the display surface). The scrolling commands are as follows:

RIGHT $< n >$	scrolls right.
LEFT $< n >$	scrolls left.
FORWARD $< n >$	scrolls forward (down).
BACKWARD $< n >$	scrolls backward (up).
TOP	scrolls to the top of the display surface.
BOTTOM	scrolls to the bottom of the display surface.

The argument n is an optional numeric argument that indicates the number of positions to scroll. The default is 5.

Only one window is active at a time. You can move, zoom, enlarge, shrink, or recolor inactive windows, but you cannot scroll or enter data.

Each display starts with the same standard lines: first a command line for entering commands, then a message line for displaying messages (such as error messages).

The remainder of the display is up to you to design. You can put fields in any positive row and column position of the display surface, even if it is off the displayed viewport.

Deciding Where to Define Fields

You have a choice of whether to define your fields in the WINDOW statement, the DISPLAY statement, or both. Defining field groups in the WINDOW statement saves work if you access the window from many different DISPLAY statements. Specifying field groups in the DISPLAY statement provides more flexibility.

Groups of Fields

All fields must be part of field groups. The group is just a mechanism to treat multiple fields together as a unit in the DISPLAY statement. There is only one rule about the field positions of different groups: active fields must not overlap. Overlapping is acceptable among fields as long as they are not simultaneously active. Active fields are the ones that are specified together in the current DISPLAY statement.

You name groups specified in the WINDOW statement. You specify groups in the DISPLAY statement just by putting them in parentheses; they are not named.

Field Attributes

There are two types of fields you can define:

- Protected fields are for constants on the screen.

- Unprotected fields accept data entry.

If the field consists of a character string in quotes, it is protected. If the field is a variable name, it is not protected unless you specify PROTECT=YES as a field option. If you want all fields protected, specify the NOINPUT group option in the DISPLAY statement.

Display Execution

When you execute a DISPLAY statement, the SAS System displays the window with all current values of the variables. You can then enter data into the unprotected fields. All the basic editing keys (cursor controls, delete, end, insert, and so forth) work, as well as display manager commands to scroll or otherwise manage the window. Control does not return to the IML code until you enter a command on the command line that is not recognized as a display manager command. Typically, a SUBMIT command is used since most users define a function key for this command. Before control is returned to you, IML moves all modified field values from the screen back into IML variables using standard or specified informat routines. If you have specified the CMNDLINE= option in the WINDOW statement, the current command line is passed back to the specified variable.

The window remains visible with the last values entered until the next DISPLAY statement or until the window is closed by a WINDOW statement with the CLOSE= option.

Only one window is active at a time. Every window may be subject to display manager commands, but only the window specified in the current DISPLAY statement transfers data to IML.

Each window is composed dynamically every time it is displayed. If you position fields by variables, you can make them move to different parts of the screen simply by programming the values of the variables.

The DISPLAY statement even allows general expressions in parentheses as positional or field operands. The WINDOW statement only allows literal constants or variable names as operands. If a field operand is an expression in parentheses, then it is always a protected field. You cannot use the statement

```
display w(log(X));
```

and expect it to return the log function of the data entered. You would need the following code to do that:

```
lx=log(x);
display w(lx);
```

Field Formatting and Inputting

The length of a field on the screen is specified in the format after the field operand, if you give one. If a format is not given, IML uses standard character or numeric formats and informats. Numeric informats allow scientific notation and missing values (represented with periods). The default length for character variables is the size of the variable element. The default size for numeric fields is as given with the FW= option (see the discussion of the RESET statement in Chapter 17, “Language Reference.”) If you specify a named format (such as DATE7.), IML attempts to use it for both the output format and the input informat. If IML cannot find an input informat of that name, it uses the standard informats.

Display-only Windows

If a window consists only of protected fields, it is merely displayed; that is, it does not wait for user input. These display-only windows can be displayed rapidly.

Opening Windows

The WINDOW statement is executable. When a WINDOW statement is executed, it checks to see if the specific window has already been opened. If it has not been opened, then the WINDOW statement opens it; otherwise, the WINDOW statement does nothing.

Closing Windows

To close a window, use the CLOSE= option in the WINDOW statement. In the example given earlier, you closed MAILLIST with the statement

```
    window close=maillist;
```

Repeating Fields

If you specify an operand for a field that is a multi-element matrix, the routines deal with the first value of the matrix. However, there is a special group option, REPEAT, that enables you to display and retrieve values from all the elements of a matrix. If the REPEAT option is specified, IML determines the maximum number of elements of any field-operand matrix, and then it repeats the group that number of times. If any field operand has fewer elements, the last element is repeated the required number of times (the last one becomes the data entered). Be sure to write your specifications so that the fields do not overlap. If the fields overlap, an error message results. Although the fields must be matrices, the positional operands are never treated as matrices.

The repeat feature can come in very handy in situations where you want to menu a list of items. For example, suppose you want to build a restaurant billing system and you have stored the menu items and prices in the matrices ITEM and PRICE. You want to obtain the quantity ordered in a matrix called AMOUNT. Enter the following code:

```
    item={ "Hamburger", "Hot Dog", "Salad Bar", "Milk" };
    price={1.10 .90 1.95 .45};
    amount= repeat(0,nrow(item),1);
    window menu
    group=top
```

```
#1 @2 "Item" @44 "Price" @54 "Amount"
group=list
/ @2 item $10. @44 price 6.2 @54 amount 4.
;
display menu.top, menu.list repeat;
```

This defines the window

```
+-----Menu---------------------------------------+
+  Command --->                                  +
+                                                +
+  Item                  Price   Amount          +
+                                                +
+  Hamburger             1.10      0             +
+  Hot Dog               0.90      0             +
+  Salad Bar             1.95      0             +
+  Milk                  0.45      0             +
+                                                +
+----------------------------------------------- +
```

Example

This example illustrates the following features:

- multiple windows
- the repeat feature
- command- and message-line usage
- a large display surface needing scrolling
- windows linked to data set transactions

This example uses two windows, FIND and ED. The FIND window instructs you to enter a name. Then a data set is searched for all the names starting with the entered value. If no observations are found, you receive the following message:

```
Not found, enter request
```

If any observations are found, they are displayed in the ED window. You can then edit all the fields. If several observations are found, you need to use the scrolling commands to view the entire display surface. If you enter the SUBMIT command, the data are updated in place in the data set. Otherwise, you receive the following message:

```
Not replaced, enter request
```

If you enter a blank field for the request, you are advised that EXIT is the keyword needed to exit the system.

```
start findedit;
   window ed rows=10 columns=40 icolumn=40 cmndline=c;
   window find rows=5 columns=35 icolumn=1 msgline=msg;
   edit user.class;
   display ed ( "Enter a name in the FIND window, and this"
   / "window will display the observations "
   / "starting with that name. Then you can"
   / "edit them and enter the submit command"
   / "to replace them in the data set. Enter cancel"
   / "to not replace the values in the data set."
   /
   / "Enter exit as a name to exit the program." );
   do while(1);
      msg=’ ’;
      again:
      name=" ";
      display find ("Search for name: " name);
      if name=" " then
         do;
            msg=’Enter exit to end’;
            goto again;
         end;
      if name="exit" then goto x;
      if name="PAUSE" then
         do;
            pause;
            msg=’Enter again’;
            goto again;
         end;
      find all where(name=:name) into p;
      if nrow(p)=0 then
         do;
            msg=’Not found, enter request’;
            goto again;
         end;
      read point p;
      display ed (//" name: " name
         " sex: " sex
         " age: " age
         /" height: " height
         " weight: " weight ) repeat;
      if c=’submit’ then
         do;
            msg="replaced, enter request";
            replace point p;
         end;
      else
         do;
           msg=’Not replaced, enter request’;
          end;
   end;
   x:
   display find ("Closing Data Set and Exiting");
   close user.class;
   window close=ed;
   window close=find;
finish findedit;
run findedit;
```

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ IML User's Guide, Version 8*, Cary, NC: SAS Institute Inc., 1999. 846 pp.

SAS/IML User's Guide, Version 8

ISBN 1–58025–553–1

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

1st printing, October 1999