DATE: December 31, 1981

TO: RD&E Personnel

FROM: Peter Stein

SUBJECT: Windows with Text or Menus for Three Glass TTY's

REFERENCE: PE-TI-844, Basic Canonical Terminal Definition Project

KEYWORDS: virtual terminal, window, menu

ABSTRACT

A simple application-level implementation of windows, in which either text or menus can be displayed, is available. Three terminals are supported: Perkin-Elmer Fox, Beehive B150 and HDS C100. Input to the display routines must be printable ASCII.

Table of Contents

1 THE INADEQUACY OF PRIMOS GLASS TTY'S

The user interface provided by PRIMOS for glass tty's is often unsatisfactory. The possibilities of managing input from the keyboard or output on the screen are quite limited. It is difficult or impossible to create an impression of order and to structure the interaction between user and computer.

A screen painted by PRIMOS consists of lines arranged in chronological order. Information becomes invisible when it scrolls off the screen or when it is overwritten. Even if a line remains visible, it may still change position on the screen. Further, the appearance of a screen may differ each time a program is run because of previous output that is still in evidence.

The result of this teletype-style interface is an unfortunate lack of visual and conceptual clarity. Information cannot be positioned on the screen so as to emphasize its importance. The helpful clue of having the form and content of information firmly associated with one another is lacking.

2 SOME REMEDIES: WINDOWS AND MENUS

Windows offer an effective mechanism to manage a screen or parts of a screen. The scrolling tty screen, which is like an endless piece of paper, becomes a well defined, stationary and controlled space. Windows limit access to data and make the use of data discretionary.

In the case of glass tty's, a window consists of a number of lines, each of which is individually accessible. A screen can accommodate multiple windows.

Menus serve to control user interaction with a program. They provide timely cues, guiding the user step by step through procedures that may appear difficult or obscure. The essence of menus is that they allow for some limited, predetermined interaction. By their very nature, they represent clearly defined pieces of user-system dialogue. Menus gain in effectiveness if they are used in windows because multiple, related pieces of information can be displayed simultaneously.

A word of caution should accompany the praise of windows and menus. The use of windows and menus does not guaranty a user interface of superior quality. These mechanisms are merely tools that make the construction of better interfaces possible.

3 IMPLEMENTATION DETAILS

At present, a window by definition consists of a number of lines, i.e., the screen cannot be divided vertically. A calling program can define up to eight text windows and four menu windows. Different facilities are offered for each kind of window.

Windows may overlap. They are not ordered, that is, they all exist in the same plane. The window last written to will appear to be "in front" of another window if the latter was overwritten, but this relation can be reversed by writing to the second window.

Input to the display routines is not interpreted and must be printable ASCII. Programs that manage the screen themselves by issuing control characters will interfere with the management of windows. In other words, the windows provide a simple mechanism for printing output and, in the case of menus, for taking some keyboard input.

4 PROGRAM MODULES

Windows and menus are available via subroutines. The routines are contained in five modules, all coded in PL/I: WINDOW.B (basic subroutine entry points), TSCRN.B (text screen management), MSCRN.B (menu screen management), DSP.B (display mechanism), and VTERM.B (terminal control).

The relationship of the modules above is such that the use of WINDOW.B with its subroutines necessitates the inclusion of all other modules at link time. It is, however, possible to use DSP.B and VTERM.B alone if simple display functions are desired. Likewise, it is possible to use TSCRN.B, DSP.B and VTERM.B if only text windows are needed; or MSCRN.B, DSP.B and VTERM.B if the user's sole concern is menu windows and facilities. To use TSCRN.B or MSCRN.B alone, the user may edit WINDOW.B and compile the entry routines he needs.

All source, binary and list files are contained in <resc43>stein>window.u.

5 SUBROUTINE DESCRIPTIONS: GENERAL ROUTINES

The available subroutines have either general applicability, as is the case with initialization and clearing the terminal screen, or they pertain only to text or only to menu windows.

5.1 Initialize windows (W$INIT)

```
CALL W$INIT (TERMINAL_TYPE, ERASE_MODE, MAX_ROW, MAX_COL)
```

TERMINAL_TYPE	1 Fox or Owl 2 Beehive B150 3 HDS C100
ERASE_MODE	0 hardware-supported erase 1 erase with blanks
MAX_ROW	contains maximum row upon return
MAX_COL	contains maximum column upon return

```
DECLARE  W$INIT   ENTRY    (FIXED BIN(15),
                            FIXED BIN(15),
                            FIXED BIN(15),
                            FIXED BIN(15));
```

W$INIT must be called before any of the routines described below can be called successfully. The type of terminal used must be specified. Specification of the procedure by which lines are erased is optional. Hardware-supported erasing to the end of a line or overwriting lines with blanks is available, with the former being the default. (Overwriting a line with blanks is slower, but avoids the flickering of the screen that is typical of the Perkin-Elmer Fox and Owl.) The routine returns values for the maximum row and column positions available on the screen of the terminal, with values ranging from 1 to n.

5.2 Clear the terminal screen (W$CLSCRN)

```
CALL W$CLRSCRN (CODE);
```

CODE	returned value to indicate result

```
DECLARE  W$CLSCRN ENTRY    (FIXED BIN(15));
```

W$CLSCRN clears the entire terminal screen without regard for windows.

6 SUBROUTINE DESCRIPTIONS: TEXT WINDOWS

6.1 Create a text window (W$MAKETW)

```
CALL W$MAKETW (WINDOW_NBR, BEGIN_ROW, SIZE, BORDER, CODE);

WINDOW_NBR                  1 to 8 for the window to be created

BEGIN_ROW                   1 to n for the row at which to start

SIZE                        1 to n for the number of rows allocated

BORDER                      character to use for drawing a line
                            across screen on last line of window

CODE                        returned value to indicate result

DECLARE  W$MAKETW ENTRY     (FIXED BIN(15),
                             FIXED BIN(15),
                             FIXED BIN(15),
                             FIXED BIN(15),
                             FIXED BIN(15));
```

W$MAKETW is called to create a text window. The number specified for the window identifies it in subsequent subroutine calls. The screen position and size of a window are described in terms of a beginning row and the total number of rows a window encompasses. If a character is specified for drawing a line across the bottom part of the window, then that line will be part of the window, but will not be written into. If, however, a value of zero is given, no line is drawn and the last row remains available for display.

6.2 Cancel a text window (W$CNCLTW)

```
CALL W$CNCLTW (WINDOW_NBR, CODE);

WINDOW_NBR                  1 to 8 for the window to be cancelled

CODE                        returned value to indicate result

DECLARE  W$CNCLTW ENTRY     (FIXED BIN(15),
                             FIXED BIN(15));
```

W$CNCLTW cancels a text window. A window that has been cancelled is no longer available. References to it in subroutine calls will be unsuccessful and have no visible results.

6.3 Clear a text window (W$CLRTW)

```
CALL W$CLRTW (WINDOW_NBR, CODE);
```

WINDOW_NBR	1 to 8 for the window to be cleared
CODE	returned value to indicate result

```
DECLARE  W$CLRTW  ENTRY    (FIXED BIN(15),
                            FIXED BIN(15));
```

W$CLRTW clears a text window. If a line has been drawn across the screen on the last row of the window to set it off from a following window, that line will remain on the display. The procedure used to erase a window has been specified in the call to W$INIT.

6.4 Display text in a text window (W$TXDSTW)

```
CALL W$TXDSTW (WINDOW_NBR, BUFF_ADDR, COUNT, CODE);
```

WINDOW_NBR	1 to 8 for the window to be written to
BUFF_ADDR	address of buffer to be displayed
COUNT	count of characters to be displayed
CODE	returned value to indicate result

```
DECLARE  W$TXDSTW ENTRY    (FIXED BIN(15),
                            POINTER,
                            FIXED BIN(15),
                            FIXED BIN(15));
```

W$TXDSTW writes characters from a buffer to a window. The text screen manager always continues to write where it last left off. When all characters contained in a buffer have been displayed or when the end of a window has been reached, the routine returns. The number of characters left to be displayed is contained in the updated COUNT upon the return of the routine. When a window has been filled with text and W$TXDSTW is called, writing resumes at the beginning of the window after it has been cleared.

The routine does not "remember" what it has written into a window. Consequently, it cannot re-display the contents of a window partially or entirely.

7 SUBROUTINE DESCRIPTIONS: MENU WINDOWS

The row and column specifications for displays in menu windows are relative to the start of a window with only two exceptions, W$MSDS MW and W$MSERMW, where relative or absolute coordinates can be used. The numbering of lines for windows starts at one.

7.1 Create a menu window (W$MAKEMW)

```
CALL W$MAKEMW (WINDOW_NBR, BEGIN_ROW, SIZE, BORDER, CODE);
```

WINDOW_NBR	1 to 4 for the window to be created
BEGIN_ROW	1 to n for the row at which to start
SIZE	1 to n for the number of rows allocated
BORDER	character to use for drawing a line across screen on last line of window
CODE	returned value to indicate result

```
DECLARE  W$MAKEMW ENTRY    (FIXED BIN(15),
                            FIXED BIN(15),
                            FIXED BIN(15),
                            FIXED BIN(15),
                            FIXED BIN(15));
```

W$MAKEMW is called to create a menu window. See the description of W$MAKETW for all pertinent details. The number of menu windows available is limited to four. The operations that can be carried out in a menu window are, in part, different from those possible in text windows.

7.2 Cancel a menu window (W$CNCLMW)

```
CALL W$CNCLMW (WINDOW_NBR, CODE);
```

WINDOW_NBR	1 to 4 for the window to be cancelled
CODE	returned value to indicate result

```
DECLARE  W$CNCLMW ENTRY    (FIXED BIN(15),
                            FIXED BIN(15));
```

W$CNCLMW cancels a menu window. A window that has been cancelled is no longer available. References to it in subroutine calls will be unsuccessful and have no visible results.

7.3 Clear a menu window (W$CLRMW)

```
CALL W$CLRMW (WINDOW_NBR, CODE);
```

WINDOW_NBR	1 to 8 for the window to be cleared
CODE	returned value to indicate result

```
DECLARE  W$CLRMW  ENTRY    (FIXED BIN(15),
                            FIXED BIN(15));
```

W$CLRMW clears a menu window. If a line has been drawn across the screen on the last row of the window to set it off from a following window, that line will remain on the display. The procedure used to erase a window has been specified in the call to W$INIT.

7.4 Get digit input in a menu window (W$DGINMW)

```
CALL W$DGINMW (WINDOW_NBR, T_ADDR, P_ADDR, E_ADDR, RETVAL, CODE);
```

WINDOW_NBR	1 to 4 for the window to get input from
T_ADDR	address of data structure that contains text of all menu items to be displayed in the menu window
P_ADDR	address of data structure that contains text for the prompt
E_ADDR	address of data structure that contains text to be displayed after incorrect input has been received
RETVAL	legal value returned by the routine
CODE	returned value to indicate result

```
DECLARE  W$DGINMW ENTRY    (FIXED BIN(15),
                            POINTER,
                            POINTER,
                            POINTER,
                            FIXED BIN(15),
                            FIXED BIN(15));
```

W$DGINMW displays menu items, a prompt for input and an error message if input was illegal. The routine attempts to automate menu management and to make it more efficient. Digits, which correspond to those of menu items, are the expected input. By convention, the numbering of menu items starts at one. (Zero is not a legal value.)

When a menu is first displayed, the window is cleared as specified in the call to W$INIT. Then the menu items and the prompt are displayed, and the program waits for input from the keyboard. If legal input was received, the routine returns with a binary value in RETVAL. If there was incorrect input, the routine displays the error message and loops to receive input again. The error and any status message from priveous activity are always erased when input is received. The routine avoids any unnecessary display work while looping.

This routine is relatively simple because the data structures associated with it contain all necessary information. Menu items are declared in a PL/I structure as follows. Each of the variables is initalized at compile time.

```
DECLARE
   1 MENU_ITEMS,
      2 NUMBER_OF_ITEMS        FIXED BIN(15),
      2 MAX_LEGAL_VALUE        FIXED BIN(15),
      2 ROW    (1 : 8)         FIXED BIN(15),
      2 COLUMN (1 : 8)         FIXED BIN(15),
      2 TEXT   (1 : 8)         CHARACTER (20) VARYING;
```

This structure is of fixed size. NUMBER_OF_ITEMS indicates how many items are actually used and should be displayed. The row, column and text of each menu item are defined in members of the structure indexed to by the same integer. MAX_LEGAL_VALUE indicates the maximum integer value that is acceptable input.

The prompt for a menu is also stored in a PL/I structure. Likewise initialized at compile time, it is declared as follows:

```
DECLARE
   1 PROMPT,
      2 ROW                    FIXED BIN(15),
      2 COLUMN                 FIXED BIN(15),
      2 INPUT_LEN              FIXED BIN(15),
      2 TEXT                   CHARACTER (nn) VARYING;
```

Here the text of the text of the prompt can be of any length nn. INPUT_LEN indicates how many digits should be accepted as a maximum. A carriage return indicates that the desired number of digits has been typed in. If incorrect input is received, the prompt is refreshed.

NOTE: The prompt should be located so that no menu items follow it. The program erases to the end of the line on which the prompt is located if incorrect input was received. Further, prompts must not be on the last line of the screen lest a carriage return force the screen to scroll up.

The error message is the simplest of the three data structures. It is declared as a character. The length of the message is limited by its position and terminal characteristics.

```
DECLARE
   ERROR_MSG   CHARACTER (nn) VARYING;
```

The position of the error display can be left as a default value (last row, column 1) or can be initialized with a call to W$ERINMW.

7.5 Get text input in a menu window (W$TXINMW)

```
CALL W$TXINMW (WINDOW_NBR, T_ADDR, P_ADDR, E_ADDR, I_ADDR, CODE);
```

WINDOW_NBR	1 to 4 for the window to get input from
T_ADDR	address of data structure that contains text of all menu items to be displayed in the menu window
P_ADDR	address of data structure that contains text for the prompt
E_ADDR	address of data structure that contains text to be displayed after incorrect input has been received
I_ADDR	address of array containing text input
CODE	returned value to indicate result

```
DECLARE  W$TXINMW ENTRY    (FIXED BIN(15),
                            POINTER,
                            POINTER,
                            POINTER,
                            POINTER,
                            FIXED BIN(15));
```

W$TXINMW performs the same services as W$DGINMW, but accepts any ASCII string instead of only ASCII digits. Where W$DGINMW returns an acceptable binary value, W$TXINMW returns a pointer to a PL/I structure, which is declared as follows:

```
DECLARE
   1 INPUT_CHARS,
      2 COUNT                FIXED BIN(15),
      2 TEXT  (MAX_COL)      CHARACTER;
```

COUNT contains the number of characters exclusive of any carriage return, and TEXT contains the keyboarded characters. The length of the character array corresponds to the maximum line length of the terminal in use.

7.6 Display a message in a menu window (W$MSDSMW)

```
CALL S$MSDSMW (WINDOW_NBR, ROW, COL, MSG_ADDR, CODE)
```

WINDOW_NBR	1 to 4 for the window to write to
ROW	row in which to display message
COL	column at which to start displaying
MSG_ADDR	address of data structure containing the text of the message
CODE	returned value to indicate result

```
DECLARE  W$MSDSMW ENTRY    (FIXED BIN(15),
                            FIXED BIN(15),
                            FIXED BIN(15),
                            POINTER,
                            FIXED BIN(15));
```

W$MSDSMW displays a message in a specified window. The length of the message may not exceed the maximum length of a line on the terminal that is in use.

If an existing window is addressed, the line number indicated is relative to the beginning of the window. If, however, zero is specified as a window number, the line number is taken as an absolute value and any window can be overwritten with a message at will.

A message is a PL/I structure declared as follows:

```
DECLARE
   MESSAGE                 CHARACTER (MAX_COL) VARYING;
```

7.7 Erase a message in a menu window (W$MSERMW)

```
CALL W$MSERMW (WINDOW_NBR, ROW, COL, MSG_ADDR, CODE)
```

WINDOW_NBR	1 to 4 for the window to erase in
ROW	row in which to erase message
COL	column at which to start erasing
MSG_ADDR	address of data structure containing the text of the message
CODE	returned value to indicate result

```
DECLARE  W$MSERMW ENTRY    (FIXED BIN(15),
                            FIXED BIN(15),
                            FIXED BIN(15),
                            POINTER,
                            FIXED BIN(15));
```

W$MSERMW erases by displaying blanks in place of the characters contained in a message. The format of the message is the same as for W$MSDSMW. Relative (window) coordinates and absolute screen coordinates may be used as is done in W$MSDSMW.

7.8 Initialize the error display in a menu window (W$ERINMW)

```
CALL W$ERINMW (WINDOW_NBR, ROW, COL, CODE)
```

WINDOW_NBR	1 to 4 for the window
ROW	row in which to display message
COL	column at which to start displaying
CODE	returned value to indicate result

```
DECLARE  W$ERINMW ENTRY    (FIXED BIN(15),
                            FIXED BIN(15),
                            FIXED BIN(15),
                            FIXED BIN(15));
```

W$ERINMW specifies in which row and column of a window error messages are to be displayed. The default values are the last writable row of the window and column one.

7.9 Display an error message in a menu window (W$ERDSMW)

```
CALL W$ERDSMW (WINDOW_NBR, ERR_ADDR, CODE)

WINDOW_NBR                  1 to 4 for the window to write to

ERR_ADDR                    address of data structure containing
                            the error message

CODE                        returned value to indicate result

DECLARE  W$ERDSMW ENTRY     (FIXED BIN(15),
                             POINTER,
                             FIXED BIN(15));
```

W$ERDSMW displays an error message in a specified window at the screen location defined by default or by a call to W$ERINMW. An error message is automatically erased when input is received by W$DGINMW and W$TXINMW. An error message is defined by the same data structure as in W$DGINMW and W$TXINMW.

7.10 Initialize the status display in a menu window (W$STINMW)

```
CALL W$STINMW (WINDOW_NBR, ROW, COL, CODE)

WINDOW_NBR                  1 to 4 for the window

ROW                         row in which to display message

COL                         column at which to start displaying

CODE                        returned value to indicate result

DECLARE  W$STINMW ENTRY     (FIXED BIN(15),
                             FIXED BIN(15),
                             FIXED BIN(15),
                             FIXED BIN(15));
```

W$STINMW provides the same service as W$ERINMW, but for status messages. The default screen location within a window is the same as for error messages, i.e., the last writable row and column one of a window. Status message, like error messages, are erased when input is received in W$DGINMW and W$TXINMW.

7.11 Display a status message in a menu window (W$STDSMW)

```
CALL W$STDSMW (WINDOW_NBR, STAT_ADDR, CODE)

WINDOW_NBR                  1 to 4 for the window to write to

STAT_ADDR                   address of data structure containing
                            the status message

CODE                        returned value to indicate result

DECLARE  W$MSSTMW ENTRY     (FIXED BIN(15),
                             POINTER,
                             FIXED BIN(15));
```

W$STDSMW provides the same service for status messages that W$ERDSMW offers for error messages.

8 CODES RETURNED BY THE SUBROUTINES

The subroutines described return a value in the variable CODE. The values reuturned have the following significance:

0	all is o.k.
1	W$INIT has not yet been called
2	window has not been created
3	window number is illegal
4	pointer to a message is null