A Window based Interactive Programming Utility On Unix: A Library Package for C programmers

Richard A. Culshaw

GEC Hirst Research Centre
Data Systems Division
East Lane
Wembley
Middlesex HA9 7PP

INTRODUCTION

A Widget is a user definable **window gadget.** They allow the user to interact with the program in specifying operations to be carried out. Widgetlib is a set of library functions which allows the user to create and interact with **widgets.** A widget can be one of four types :

1) **Command** Widget 2) **Label** Widget 3) **Toggle** Widget 4) **Input** Widget

Command Widgets

A command widget allows the user to select a function to be carried out, e.g in a paging program one such command could be 'Next Page'. In creating a command widget the programmer must pass it the function to be carried out when the widget is selected (activated). The widget is activated using a unique 'activation' character which is displayed in the widget for easy reference.

Label Widgets

A label widget is simply a widget which holds a character string such as a heading, which can be changed at will later. The programmer can supply parameters which can either left justify (default), right justify, or centre the message within the widget (which has a user definable width) as well as highlight (default) or dehighlight it.

Toggle Widgets

A toggle widget contains a message which indicates the current value of an entity which can be more than one value e.g, Yes/No/Don't-Know. The value is toggled when a given character is pressed. The activation character is also displayed in the widget. At any point the programmer can enquire as to the the current toggle value.

Input Widgets

An input widget allows the user to enter information into the program in the form of a string, e.g in a paging program one might want to supply a file name. There are two types of input widgets :

a) The widget is activated immediately on creation therefore an activation character is not required. E.g in a paging program one might have a command widget which activates this type of input widget in order to get a file name.

b) The widget has an associated activation character so therefore it may be activated at any point after creation. The character is displayed in the widget.

For ease of use the input widgets contain character strings which can explain there use within the program.

Text Window

The library also allows the user to define a text window into which the program may report information. On creation the program has to pass a parameter which specifies how much room to leave before the top of the text window. E.g 0 means place the window immediately below the lowest widget, 1 means allow room for one more row of widgets before placing the window. The text window always stretches to the bottom of the screen. The window can be bordered on any 2 opposite sides or on all 4 (effectively boxing

it).

Widget Placement

The library maintains the current X,Y coordinate of the last widget created. When a new widget is created the program must pass a parameter 0 or 1. A 0 means put the new widget on the current row if possible, else go onto the next row. A 1 forces the new widget onto the next row. This means that widgets can only go after preceding ones and not before. But however widgets can be placed over widgets which have previously been destroyed.

Compiling Things

In order to use the library, it is necessary to have certain types and variables defined. Therefore, the program must have the line:

#include <widgets.h>

at the top of the source. The header file includes <curses.h>, <sgetty.h>, <stdio.h> so you don't have to include them also. Also compilations should have the following form:

cc [flags] file ... -lwidgets -lcurses

Type Definitions

The functions which create widgets return a type **WIDGET** which is used for later identification of the widget.

If a function is required to return a widget identifier, then it will return **NULLWIDGET** upon error.

Screen Refresh

The library supports a standard screen refresh invoked by ^L.

USAGE

Getting Started

The first thing to do when using the library is set up the windowing system by calling

initialisewidgets()

Creating Widgets

```
WIDGET mkcmdwidget (message, ch, function, row)
char message[];
char ch;
int (*func)();
int line;
```

Creates a new command widget. message holds the message to be displayed in the command widget. ch is the activation character on which function is executed. row specifies where to place the widget (as described above). The returned value is the widget identifier for later use.

```
WIDGET mklblwidget (message, position, space, row)
char message[];
int position, space, row;
```

Creates a new label widget. Message holds the contents of the label, position specifies how the message is to appear. Using the constants CENTRE, LEFTJUST, RIGHTJUST, NOHIGH, pass a combination to position e.g CENTRE or LEFTJUST|NOHIGH etc.space indicates how large the label widget is to be. A positive value is the absolute length of the widget, whereas value <= 0 specifies how far the widget is to be from the righthand edge of the screen. row specifies where to place the widget (as described above). The r

eturned value is the widget identifier for later use.

```
WIDGET mktglwidget (message, number, tgl, tglch, row)
char message[];
int number;
char tgl[10][20];
char tglch;
int row;
```

Creates a new toggle widget. message contains a label for the widget, and number holds the number of toggle values. tgl is used for holding actual toggle values which are toggled between using the activation character tglch. At present there is a limit of 10 toggle values, each being a maximum length of 20 characters. row specifies where to place the widget (as described above). The returned value is the widget identifier for later use.

```
WIDGET mkinpwidget (message, character, input, lengthstr, lengthwid, row, exec)
char message[], character, input[];
int lengthstr, lengthwid, row, exec;
```

Creates a new input widget. message contains the label for the widget, and character is the activation character for the widget. input is the variable in which the user wishes the data obtained from the input widget to be placed, lengthstr is the length of input. lengthwid specifies how big to make the widget. A positive value indicates the absolute length of the widget, where as a value <= 0 indicates how far from the righthand edge the widget is to be. exec is used to indicate whether the widget should b

e activated on creation by (passing EXEC), or should remain dormant until the activation character is used (passing NOEXEC). (n.b when passing EXEC an activation character is not needed therefore one could pass NULL for character.)

Text Window Management

```
int opentextwindow (position, border)
int position, border;
```

Opens and displays the text window. position specifies where the window is to go. i.e position = 0 will place the window immediately below the lowest widget, position = 1 will allow room for 1 more row of widgets below the current lowest before placing the window etc. border is used to indicate whether the window should be bordered. i.e using the parameters VERTICAL & HORIZONTAL, one can specify whether the window should be bordered vertically (passing VERTICAL), horizontally (passing HORIZONTAL), both (pa

ssing VERTICAL|HORIZONTAL), or neither (passing 0). The function returns the number of lines within the window.

```
report (message)
char *message;
```

Displays the given message in the text window. Scrolling is done automatically when the bottom of the window is reached and is therefore of no concern for the user.

```
cleartextwindow ()
```

Clears the text window and returns its cursor to the top lefthand corner.

```
killtextwindow ()
```

Destroys the text window. The text window can be redefined at a later date.

Killing Widgets

```
int killwidget (ptr)
WIDGET ptr;
```

Kills the given widget. the function returns TRUE if successfull or FALSE if the widget doesn't exist.

Interaction

```
WIDGET widgetinput ()
```

Wait for input from widgets. Program control is passed back when :-
a) A Command widget is activated and control is passed to its associated function.
b) An input widget is activated and after subsequent input, control is passed to the calling function with the widget pointer of the activated input widget.
Activating a toggle widget just toggles the widget, control is only passed back upon one of the above conditions.

```
int tsttglwidget (ptr)
WIDGET ptr;
```

Returns the index value of the current state of the given toggle widget ($0 < \text{index} < \text{number toggle values} - 1$). Returns FALSE if the given widget does not exist.

```
int activate (ptr)
WIDGET ptr;
```

Reactivates a widget which was previously deactivated. Returns TRUE if successful otherwise FALSE.

```
int deactivate (ptr, blank)
WIDGET ptr;
int blank;
```

Deactivates the given active widget. The widget remains on screen but it cannot be activated. blank can be BLANK or NOBLANK which indicates whether the contents of the widget should be blanked out or not while it is inactive. The function returns TRUE if successful or FALSE if the given widget does not exist.

```
int highlight (ptr)
WIDGET ptr;
```

Highlights the given widget. Returns TRUE if successful else FALSE if the widget does not exist.

```
int dehighlight (ptr)
WIDGET ptr;
```

Turns off the highlighting of the given widget. Returns TRUE if successful else FALSE if the widget does not exist.

```
int changelblwidget (ptr, message, position)
WIDGET ptr;
char message[];
int position;
```

Changes the contents of the given label widget to that given in message. position specifies how the message is to appear (see mklblwidget). The function returns TRUE if successful else FALSE.

```
WIDGETTYPE widgettype (ptr)
WIDGET ptr;
```

Returns the type of widget pointed too by ptr. The returned type value is one of CMD, LBL, TGL, INP or FALSE if the widget does not exist.

Finishing Up

```
endwidgets ()
```

Closes down the windowing system and should be used at the end. This function is also automatically called on an interrupt (e.g ^C).