# A Brief MUI User Guide.

**By Steve Baker**

## Introduction

The Micro User Interface (MUI) was written by Tom Davis (SGI) and comes as a part of the GLUT (OpenGL Utilities Toolkit) release 3.6. MUI's major drawback is that it is **completely undocumented**. The significance of MUI is that it is written entirely on top of GLUT and OpenGL - and as such should port cleanly across a wide range of UNIX, Mac, VMS and PC operating systems and be completely window-system independant.

To the user, MUI has a look-and-feel similar to X-Motif. All functions use the usual SGI naming conventions with a 'mui' prefix. The API is strongly object-oriented - but uses a C syntax. Here is a snap shot of a simple MUI program:

NB: There appears to be a widely used GUI for the Amiga that is also called MUI - I'm pretty sure it's unrelated to this library.

### Disclaimer

This document is an attempt to make MUI at least partially useful. Since I have built this description solely upon the two available demo programs and from the library source code, I could easily be hopelessly wrong. There will certainly be significant errors and omissions since I'm writing it as I learn to use MUI myself.

Hopefully this documentation will only be a stop-gap measure until something official appears in some future GLUT/MUI release.

## Getting Started

MUI comprises a bunch of header files (currently in the "include/mui" directory within the GLUT release), plus a single library file ('libmui.a' under UNIX, 'mui.lib' under Windoze).

A simple MUI program must start off by doing all the usual GLUT initialisations - for example:

```
#include <stdio.h>
#include <stdlib.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <mui/mui.h>

int main ( int argc, char **argv )
{
        glutInitWindowSize( 640, 480 );
        glutInit( &argc, argv );
        glutInitDisplayMode ( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
        glutCreateWindow ( "My Application"  );
```

The next step is to construct all the MUI objects that comprise your user interface - I'll cover that in the next section (below).

Finally, you should call 'muiInit()' and enter the GLUT main loop:

```
        muiInit () ;
        glutMainLoop () ;
        return 0 ;
}
```

### The `muiInit()` function and GLUT.

This routine is where MUI takes over most (if not all) of the GLUT callbacks:

```
glutKeyboardFunc
glutMouseFunc
glutReshapeFunc
glutMotionFunc
glutPassiveMotionFunc
glutDisplayFunc
glutMenuStateFunc
```

This is unfortunate since it pretty much prevents you from catching any of these events yourself.

## Adding Widgets.

Before you start adding widgets, you have to create a User Interface List.

Example:

```
muiNewUIList ( 1 ) ;
```

Every widget you add goes into that UI List.

### User Interface Lists.

You can manage multiple UI lists using these calls:

```
void muiNewUIList( int listid ) ;
void muiAddToUIList( int listid, muiObject *obj ) ;
void muiSetActiveUIList ( int listid ) ;
int  muiGetActiveUIList () ;
```

Clearly (since there are no routines to remove an object from a UIList, or to delete an entire UIList), it is intended that MUI user interfaces are somewhat static in nature.

### Constructing a Menu Bar.

MUI menu bars are built from a bunch of GLUT popup menus which are then bound to a MUI object.

Example:

```
int menus_for_menubar [ 3 ] ;
/* Build three GLUT popups... */
menus_for_menubar [ 0 ] = glutCreateMenu ( menu_cb  ) ;
glutAddMenuEntry ( "New"      , MENU_FILE_NEW       ) ;
glutAddMenuEntry ( "Open..."  , MENU_FILE_OPEN      ) ;
glutAddMenuEntry ( "Save"     , MENU_FILE_SAVE      ) ;
glutAddMenuEntry ( "SaveAs...", MENU_FILE_SAVEAS    ) ;
glutAddMenuEntry ( "Exit"     , MENU_FILE_EXIT      ) ;
menus_for_menubar [ 1 ] = glutCreateMenu ( menu_cb  ) ;
glutAddMenuEntry ( "Cut"      , MENU_EDIT_CUT       ) ;
glutAddMenuEntry ( "Copy"     , MENU_EDIT_COPY      ) ;
```

```
glutAddMenuEntry ( "Paste"    , MENU_EDIT_PASTE     ) ;
menus_for_menubar [ 2 ] = glutCreateMenu ( menu_cb  ) ;
glutAddMenuEntry ( "About..." , MENU_HELP_ABOUT     ) ;
glutAddMenuEntry ( "Help"     , MENU_HELP_HELP      ) ;
```

In this case, all the MENU_* constants would be #defined somewhere and the 'menu_cb' function would probably just be a big switch statement that performs the appropriate actions as each menu item is invoked. All of this is covered in the GLUT documentation.

Next, you have to attach the GLUT menus to a menu bar:

```
muiObject *menubar = muiNewPulldown () ;
muiAddPulldownEntry ( menubar, "File", menus_for_menubar[0], 0 ) ;
muiAddPulldownEntry ( menubar, "Edit", menus_for_menubar[1], 0 ) ;
muiAddPulldownEntry ( menubar, "Help", menus_for_menubar[2], 1 ) ;
```

The final argument to 'muiAddPulldownEntry' is a boolean which is TRUE for the 'HELP' menu (which is always on the extreme right of the menu bar in traditional GUI's) - and FALSE for all the other entries which are simply packed onto the bar from left to right.

Here is the menu bar API in full:

```
muiObject *muiNewPulldown() ;
void muiAddPulldownEntry ( muiObject *obj, char *title, int glut_menu, int is_help ) ;
```

NB. There seems to be a bug in MUI - if the GLUT window is ever resized - the menu bar can get hidden - or positioned somewhere in the middle of the window instead of at the very top where it belongs.

## Buttons.

There are three different styles of button:

- Button (with text inside).
- Radio Button (with text alongside).
- Tiny Radio Button (also with text alongside).

First, construct the button using one of these three constructor functions:

```
muiObject *muiNewButton  ( int xmin, int xmax, int ymin, int ymax ) ;
muiObject *muiNewRadioButton ( int xmin, int ymin ) ;
muiObject *muiNewTinyRadioButton ( int xmin, int ymin ) ;
```

If a label is needed for the button:

```
void muiLoadButton ( muiObject *button, char *label ) ;
```

'Radio' buttons are often used when pressing one button of a group causes any depressed buttons in that same group to turn off automatically. TO make this behaviour, you need to link all the buttons of a particular group together:

```
void muiLinkButtons ( muiObject *obj1, muiObject *obj2 ) ;
```

Example:

```
muiObject *m1 = muiNewRadioButton ( 100, 100 ) ;
muiObject *m2 = muiNewRadioButton ( 100, 120 ) ;
muiObject *m3 = muiNewRadioButton ( 100, 140 ) ;
muiLinkButtons ( m1, m2 ) ;
```

```
muiLinkButtons ( m2, m3 ) ;
```

Sometimes you need to turn off all the buttons in a group:

```
void muiClearRadio ( muiObject *button ) ;
```

### Text Labels.

Text labels can be placed anywhere in the window using either one of the following two commands:

```
muiObject *muiNewLabel     ( int xmin, int ymin, char *label ) ;
muiObject *muiNewBoldLabel ( int xmin, int ymin, char *label ) ;
```

The two calls are identical except that the second generates a label in bold-faced text. You can change the string in the label after creation using this:

```
void muiChangeLabel ( muiObject *obj, char *label ) ;
```

### TextBox

This is actually a text entry box. Here is the API:

```
muiObject *muiNewTextbox ( int xmin, int xmax, int ymin);
char      *muiGetTBString   ( muiObject *obj ) ;
void       muiClearTBString ( muiObject *obj ) ;
void       muiSetTBString   ( muiObject *obj, char *s ) ;
```

The textbox allows the user to enter text which can be queried using muiGetTBString(), cleared using muiClearTBString() or preset using muiSetTBString().

### TextList

Sorry - I havn't documented these yet.

```
muiObject  *muiNewTextList  (int  xmin,  int  ymin,  int  xmax,  int
listheight ) ;
muiObject  muiSetTLTop ( muiObject *obj, float p ) ;
int   muiGetTLSelectedItem ( muiObject *obj ) ;
void  muiSetTLStrings ( muiObject *obj, char **s ) ;
void  muiSetTLTopInt ( muiObject *obj, int top ) ;
```

### Vertical Slider and Horizontal Slider

Sorry - I havn't documented these yet.

```
muiObject *muiNewVSlider ( int xmin, int ymin, int ymax, int scenter,
int shalf);
muiObject *muiNewHSlider ( int xmin, int ymin, int xmax, int scenter,
int shalf);
float  muiGetVSVal ( muiObject *obj ) ;
float  muiGetHSVal ( muiObject *obj ) ;
void   muiSetVSValue ( muiObject *obj, float val ) ;
void   muiSetHSValue ( muiObject *obj, float val ) ;
void   muiSetVSArrowDelta ( muiObject *obj, int newd ) ;
void   muiSetHSArrowDelta ( muiObject *obj, int newd ) ;
```

## General Functions - Applicable to all muiObjects.

All muiObjects share a collection of useful API.

MUI objects can be visible (ie displayed) or invisible (hidden), active (ie clickable) or inactive ('greyed out') and can be enabled or disabled (for a button, this means depressed or not depressed, for a text box, the cursor is on or off, etc).

```
void muiSetVisible ( muiObject *obj, int state ) ;
void muiSetActive  ( muiObject *obj, int state ) ;
void muiSetEnable  ( muiObject *obj, int state ) ;
int  muiGetVisible ( muiObject *obj ) ;
int  muiGetActive  ( muiObject *obj ) ;
int  muiGetEnable  ( muiObject *obj ) ;
```

All MUI objects has a single integer (called the 'ID') which can contain arbitary user data.

```
void muiSetID ( muiObject *obj, int id ) ;
int  muiGetID ( muiObject *obj ) ;
```

You can query the bounding box of a MUI object. (Note the order of the arguments - which is different from the routines to create objects of a given size!)

```
void muiGetObjectSize(muiObject *obj, int *xmin, int *ymin, int *xmax,
int *ymax);
```

Each object can have a callback function associated with it that is called under various user imput conditions:

```
void muiSetCallback ( muiObject *obj, void (*cb)(muiObject *,
        enum muiReturnValue) ) ;
enum muiReturnValue { MUI_NO_ACTION, MUI_SLIDER_MOVE,
        MUI_SLIDER_RETURN, MUI_SLIDER_SCROLLDOWN, MUI_SLIDER_SCROLLUP,
        MUI_SLIDER_THUMB, MUI_BUTTON_PRESS, MUI_TEXTBOX_RETURN,
        MUI_TEXTLIST_RETURN, MUI_TEXTLIST_RETURN_CONFIRM };
```

The user callback function is passed the address of the muiObject that was clicked and also a parameter indicating how the object was activated.

## Misc. Functions

The following function allows the application to register a callback that is invoked whenever the mouse is clicked and no MUI widget is present at that screen location.

```
void muiSetNonMUIcallback ( void (*cb)(int, int) ) ;
```

The routine is called with the (x,y) coordinate of the mouse at the time.