

Fast Start Hello World

1. Setup

Linux:

Make sure, gcc or clang, make, freeglut3-dev, libglew-dev and Lazarus (Linux version) are installed. The lazbuild command (from Lazarus) should be accessible through the PATH variable. Installing graphviz is recommend, but not necessary.

Go into the unzipped folder and run `./compile.sh` (This will build several test applications and some tests are performed.)

Windows without samples and Test function:

Install and start Lazarus. Project → Open Project. Select `menuedit.lpi` from the `menuEdit` folder. Start → Compile.

You must figure out how to compile and test the samples yourself. The Test function of the editor won't work. If you want to use the graph drawing with graphviz, see instructions below.

Windows with Cygwin:

In order to avoid possible problems: Do not use spaces in you filenames and paths.

Install Cygwin with the packages `gcc-core`, `gcc-g++`, `make`, `libglut-devel`, `libglew-devel` and `xinit`.

Install Lazarus (32 or 64bit Windows version). Open the `menuEdit/menuedit.lpi` project with Lazarus and compile (pressing F9). You can install the windows version of Graphviz. Adjust the "SET BROWSER" and "SET GRAPHVIZ" line inside `makegraph.bat` to your system settings (required if you did use non default installation directories or something other than graphviz version 2.38).

Open the *Cygwin Bash Shell* from the Windows start menu.

Run `./compile.sh` (This will build several test applications and some tests are performed).

Go into the `MenuEditor` folder and open `menutester.bat` with your favourite editor. Adapt the "SET CYGWIN" line to the path you used for the Cygwin installation. (You dont need to do this if you selected the default installation path.) Save the file. In order to start the test program from the Editor, you need to open a *Cygwin Bash Shell* and enter `startxwin`.

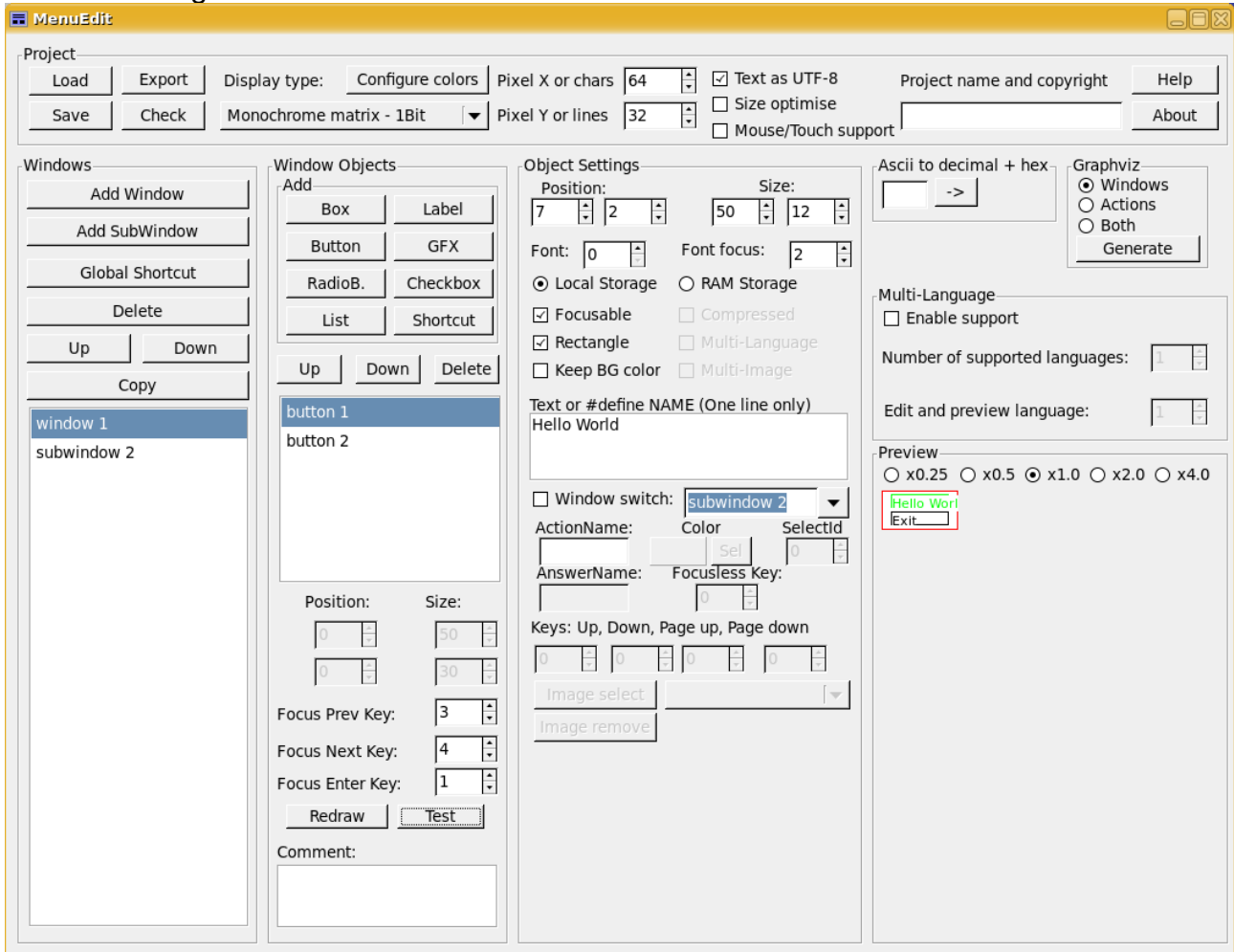
2. An own hello world

This example shows how to build a simple test application which shows the current system time.

1. Start `MenuEdit`
2. Click on *Add Window* and select the *window 1* from the list below.
3. Add two *Buttons*
4. Select *button 1*, set the size values to 50 and 12. While pressing the shift key make a right click on the left *Position* field. This will automatically place the button in the centre. In our case this is 7 pixels from the left border. Set it 3 pixels from the top border.
Note: Sometimes you need to press enter in the number selection fields, in order to update the value.
5. In the big field, enter the text "Hello world".
6. Set the *Font focus* value to 2. This will result in an underlined text if the button has a

focus.

7. Select *button 2* and set the size to 50x12 pixel. Set the position to 7 (from the left) and 17 (from the top) pixel. Set the text to “Exit” and the Font focus to 2.
8. In the *Window Objects* box, set the *Focus Prev Key* to 3, the *Focus Next Key* to 4 and the *Focus Enter Key* to 1.
9. Add a new Sub Window to your menu.
10. Select *button 1* within *window 1* again. Then choose *subwindow 2* from the *Window switch* drop down menu.
(If you like, you can press the Test button.) Your editor now should look like the following:



11. Enter “TIME” as text (without the quotes) into the field *ActionName* of *button 1*. Select *button 2* and enter “EXIT” into the *ActionName* field.
12. Now go to the *subwindow 2* and add two labels and one button.
13. Set the size of the *subwindow 2* to 46x30. Make a shift+right click to the two position numbers in order to place it in the middle. Set the three focus key values to the same as in *window 1*.
14. With *label1*, set the text to “Your time:” and the position to 11x3.
15. With *label2*, set the text to “CLOCK”, select *RAM Storage*, set the position to 18x11 and use font 1.
16. Set the text of *button 3* to “Close”, select 2 for *Font focus*, the size 35x12 and the position 15x18. Enter “RET” (without quotes) as window switch (This is a special keyword).
17. Save your project (at best in an empty folder). Don't forget to manually add the .xml extension to your filename.

18. You can run the *Check* function in order to see if you forget some of the steps above.
19. If you select *window 1* and run the *Test* function, you should be able to show and hide the subwindow by clicking with the mouse on the buttons. The up and down cursor keys can change the selected button. The cursor to the right does an enter on the currently selected button, the same way the mouse does with a click.
20. Now its time to manually add some code. This may be platform depended, but hey you are a developer, I guess you know how to write code for your platform. Run the *Export* function and select the directory where you saved your .xml file. (Any other directory would work too, but please note, that some existing files could be silently overwritten). Copy the files *menu-interpreter.c*, *menu-interpreter.h*, *menu-text.c* and *menu-text.h* into the same directory (Instead of copying, Unix users can use symlinks for the four files). If you do not want to start with empty functions, copy *pc-mouse-demo/mouse-demo.c* too (The code is using OpenGL for drawing).
21. Allow the menu to access the bytecode: You need to have a function *menu_byte_get*, which returns the byte from the bytecode at the given address. The most simple implementation would be:

```
uint8_t menu_byte_get(MENUADDR addr) {
    if (addr >= MENU_DATASIZE) exit(1);
    return menudata[addr];
}
```

menudata is defined in the generated *menudata.c*

22. *void menu_screen_set(SCREENPOS x, SCREENPOS y, SCREENCOLOR color)* should show the screen or better store the changed pixels to a buffer. (Double-buffering). As we have selected a monochrome matrix as display type, *SCREENCOLOR* will be an *uint8_t*. Color can be 0 for black or 1 white. Depending on your editor screen resolution, *SCREENPOS* is either an *uint8_t* or an *uint16_t*.
 23. *void menu_screen_flush(void)* Should do everything necessary to show the data in the buffer as a screen. If no double-buffering is used, this function can be empty.
 24. *void menu_screen_clear(void)* should clear the buffer defined above. A simple implementation would be:
- ```
void menu_screen_clear(void) {
 int i, j;
 for (i = 0; i < MENU_SCREEN_Y; i++) {
 for (j = 0; j < MENU_SCREEN_X; j++) {
 menu_screen_set(j, i 0);
 }
 }
}
```

But often there are faster ways for a clear.

25. In the function *uint8\_t menu\_action(MENUACTION action)* your ActionNames come into play. This function gets called whenever some action is run within the menu. In our example, you need to provide code for showing the time and exiting the program. The ActionNames are part of the defines, which should be used and compared with the action parameter. The prefix is always *MENU\_ACTION\_*.

An implementation for the example could be:

```
uint8_t menu_action(MENUACTION action) {
 static char mytime[6];
 if (action == MENU_ACTION_TIME) {
 time_t t = time(NULL);
 strftime(mytime, 9, "%H:%M", localtime(&t));
 menu_strings[MENU_TEXT_CLOCK] = mytime;
 }
 if (action == MENU_ACTION_EXIT) {
 exit(0);
 }
 return 0;
}
```

}

As you can see, the array `menu_stings` contains a list of pointers to text in the RAM, and the index can be found by the generated define `MENU_TEXT_` and the part entered in the menueditor. If the array contains NULL pointers, simply no text is displayed.

The return value tells the interpreter if a redraw is necessary (0 = No, 1 = Yes). If a screen switch is executed, this is done anyway and the value has no effect. `MENUACTION` can be an `uint8_t` or `uint16_t`. A menu with 16bit addressing always uses a 8bit action, a menu with 24bit addressing always an action with 16bit.

26. In the main function, you should first call `menu_redraw` and then wait for proper user input and convert it into the key codes and then call `menu_keypress(key)`. Key is the number you set as key in the editor (in our example 1=Enter, 3=Previous and 4=Next.).
27. Do not forget to include "menu-interpreter.h" and `<time.h>`. A simple way to access the byte code is to include "menudata.c" too.
28. Now compile your program and test it. If you used the the mouse-example.c, you may need to adapt the includes to your needs and then you should be able to compile with the following command:

```
gcc -o helloworld mouse-demo.c menu-interpreter.c menu-text.c
-lglut -lGL -DMENU_MOUSE_SUPPORT
```

Have fun.

### 3. All those defines

The generated `menu-interpreter-config.h` contains a lot of defines. I hope most are self-explaining, so here are only the important or not self-explaining ones.

**MENU\_ACTION\_X** Defines for the `menu_action` function to find out which function of the menu got activated. X is the `#define NAME` you entered in the editor. Each list has an extra **MENU\_ACTION\_INDEXCHANGE\_Y** to notify about selection changes. Y Is the AnswerName.

**MENU\_CHECKBOX\_X** Defines the index in the `menu_checkboxstate` array to read and write if the checkbox is checked.

**MENU\_RBUTTON\_X** Defines the index in the `menu_radiobuttonstate` array to read and write which radiobutton of a group is selected.

**MENU\_LISTINDEX\_X** Defines the index in the `menu_listindexstate` array to read and write which line of a list is selected.

**MENU\_GFX\_X** Defines the index in the `menu_gfxdata` array to read and write the data for an image.

**MENU\_SDATA\_X\_Y** Defines the index of static data (text or images) in the byte code. Use this if you want to use the data at some other places in your program. X is the number of the (Sub)Window and Y the number of the object with the static data.

### 4. The test function

The test function works by adding an empty window at the beginning which contains a

window switch to the currently selected window. Key code 254 is used for this purpose, so it may not be used as global shortcut. The menu is then exported into a temporary directory and `menutester.sh` (Linux/Unix)/ `menutester.bat` (Windows) is called from the directory where the `menuedit` binary is. The working directory is the path of the executable, the first parameter the temporary path with the exported menu and the second parameter the key code for switching to the right (Sub)Window.

See the `mouse-demo.c` for all features. It's up to you to write an other test function.

Note: Currently, the temporary directory and its content is not deleted.

The `mouse-demo` currently uses the following mapping for the key codes:

Left click or cursor right: 1

Middle click: 3

Right click or cursor left: 2

Scroll wheel up: 4

Scroll wheel down: 5

Cursor up: 3

Cursor down: 4

The left click and cursor right key is automatically adapted to the defined *Focus Enter Key* of the current (Sub)Window if it is not 0. The same is true for the up and down cursor, which uses the values for *Focus Prev Key* and *Focus Next Key*.

## 5. The Check function

The check function tries to find the most common errors I made while defining a menu:

- Warn about objects which have an action or screen change but can not be selected.
- Warn about screens without Prev/Next/Enter keys (if there are objects which can be selected)
- Warn about unreachable screens (does not detect loops or missing ways back)
- Give hint about having the same font for normal and focus