Chapter 1

Chapter 1: GUI Builder Quickstart Guide

The GUI Builder is a source-code generator that runs on your Mosaic touchscreen controller to assist you in creating an effective and interactive graphical user interface (GUI) for your application. While your controller's GUI Toolkit already provides the tools you need code a custom GUI for your application, the GUI Builder goes a step beyond that by coding your GUI for you. It gives you a what-you-see-is-what-you-get approach to creating menus by placing and grouping graphics and buttons on your controller's display. You can edit and test your layouts directly on your target hardware. As you create your menus, you control the GUI Builder through a serial terminal while simultaneously viewing your menus on your controller's LCD/Touchscreen. When adding new graphics or buttons to menus, you can use your finger on the touchscreen or keys on the PC keyboard to move them around to precisely where you want them. After creating your menu content, the GUI Builder then generates the source code that implements your user interface. You can use this source code directly in your application, or you can modify it as you like. The advantages of the GUI Builder are:

- ⇨ *The GUI Builder allows you to design your GUI visually, eliminating the need to painstakingly work out the screen coordinates of each of your objects.*
- ⇨ *All you need to get started using the GUI Builder is a graphics library. You may use our sample library or create your own custom graphics.*
- ⇨ *The GUI Builder generates all the code you need to define your objects using the GUI Toolkit. You may then further edit this code by hand if you wish.*

To get started with the GUI Builder, you can download a single file that includes the builder plus a demonstration GUI session. For a detailed description of the steps involved in creating your user interface, consult the GUI Builder Users Guide. The remainder of this Quickstart Guide describes how to get started using the GUI Builder with the demo session.

Installing the Software

The GUI Builder will be packaged with future versions of the IDE. You must already have a working installation of the IDE before you begin this installation procedure. Contact Mosaic Industries if you need a replacement copy. The IDE is rooted in a directory called **\mosaic**, which is typically installed in the root directory of your C drive (**c:\mosaic**). The GUI Builder is shipped on CD as a pair of directories that you should simply drag and drop into your **\mosaic** directory. You can download the GUI Builder as a zip file containing these two directories. Again, simply drag and drop them into your **mosaic** directory. They are called **GUI_Builder** and **Mosaic Image Converter**. The image converter directory simply contains a replacement executable for the image converter utility to ensure you are using the most up to date version (V1.10 or later). The **GUI_Builder** directory contains the default image library, example session, this Quickstart Guide, the GUI Builder Users Guide, and all the files needed to install the GUI Builder.

To get started using the GUI Builder, you simply send a single file to the controller to install the GUI Toolkit, the GUI Builder, and the demonstration (demo) session environment. Start the QED Terminal application provided with your IDE, connect your PC's serial port to the controller, power up the controller, and verify communications by tapping the enter key. If you see 'ok' on your terminal screen, you can skip the following troubleshooting paragraph.

If you don't see 'ok' being printed on your terminal screen, verify that the controller is connected to the same serial port to which the terminal program is configured, and that the terminal is set to 19200 baud. If there is another auto-starting program running on the controller such as the GUI toolkit demo, you can clear it by performing the factory cleanup procedure: install the factory clean jumper and cycle power to the controller.

Once communications are established, choose 'Send File' from the 'File' menu of the terminal program, and select the file **gui_builder_w_demo.4th** from the GUI Builder directory. This file takes about 4 minutes to download, after which the GUI Builder starts automatically. ***Next, calibrate the touchscreen by choosing '***`T. (Re)calibrate touch screen`***' from the GUI Toolkit's command list and following the instructions.*** This step only has to be performed once after installing the GUI Toolkit kernel extension (which is installed as part of downloading **gui_builder_w_demo.4th**). Now you're ready to use the GUI Builder.

Test Driving the GUI Builder Demo

The demo session that you have loaded lets you try out the pre-defined sample menus. Select '`4. Exercise an existing menu`' and choose one of the two menus by typing its number and pressing enter. The first menu, '`BASICMENU`' has a couple of static graphic objects and a few buttons. It shows how you can use status graphics to serve as control labels and visually tie buttons together. The second menu, '`MENU2`' has quite a few more buttons on it and illustrates the diversity of the available button designs provided in the graphics library.

Try exercising '`MENU2`.' You will see file tabs, buttons with custom press graphics (as opposed to a generic filled in rectangle), and others. Note the variety of the buttons. If you are willing to invest some time drawing graphics, you can make a very elegant user interface by having custom graphics within each button to help connote its meaning. The QScreen controller has quite a bit of memory available, so you can make as many custom graphics as you need. Other buttons in `MENU2` use generic blank button graphics with text labels. This technique saves lots of memory, and makes it possible for your application software to change the text displayed in buttons. Buttons that are only distinguished by text are visually less distinctive, but they are flexible and easy to create.

Hit any key to return to the GUI Builder's main command list. Select '`6. Edit a button`' and press 'B' to browse though the buttons. Following the instructions, you can navigate forward and backwards through the defined buttons. Press 'enter' to select one to edit. Explore the button setting items. You can hit 'Esc' (the escape key) to return to the previous command list. *Note that beeping settings that you set with the button behavior options are not evident until the code has been generated, so you will not hear beeping while you are exercising a menu.* If you wish to refresh the session to its original state, simply go to the GUI Builder's main command list and select '`B. Load prior session`.'

> **Be sure to save your session if you've invested a great deal of time in it; simply select '`B. Save this session`.' This allows you to recover your work even if the controller memory is overwritten. If you are using a controller with battery backed RAM, you can safely power off the controller at the main command list without losing your work. Most of the command list prompts are safe places to shut down, but turning off the controller while an operation is in progress may cause the session memory to be corrupted, after which the GUI Builder will restart with a blank session.**

You can try adding one of the buttons to one of the example menus. The same buttons may be used multiple times in a menu. Select '`2. Edit an existing menu`' and chose '**`BASICMENU`**.' Choose '`1. Add a graphic object to this menu.`' You will see a list of 62 graphics. You might want to resize your terminal window if you want to see them all. Instead of entering by number, hit 'B' instead to browse the graphics. Just as with buttons, you can navigate forward and backwards through the graphics. Hit Enter on a specific graphic to select it. You will then see the menu on the LCD with a grid superimposed. This shows the 20 touch-sensitive areas on the screen for buttons. The lines of the grid represent the borders of adjacent sensitive areas. You can move the graphic around on the menu with the keyboard, or by touching any location on the screen, which causes the upper left corner of the graphic to be placed at that screen location. You can also drag your finger around on the screen to locate the graphic. Hit enter to place the graphic at the current location or Esc to abort the placement and leave the object in the upper left corner of the screen. Try adding some buttons. Exercise the menu, and experiment with your sample GUI.

Advanced Techniques

File Tabs

The file tab buttons in the demo take advantage of the distinction between the 'release graphic' and the 'draw graphic.' The release graphic is actually the foreground file tab, and the draw graphic is

the background tab. The other trick at work here lies in the button behavior settings. Select, '`6. Edit a button`' from the main command list and choose one of the file tab buttons, `TAB1`, `TAB2`, or `TAB3`. Notice the arrangement of the graphics, and examine the button behavior settings. Option '`7. Does button activation cause all graphics on screen to be redrawn?`' is normally set to 'no', but in this case it is set to 'yes'. For the file tabs, we take advantage of the flag that causes the screen to be redrawn with all the buttons in their 'draw graphic' states when the tab button is pressed. The draw graphics show the tabs as being in the background, and when the button is released, the release graphic will be directly drawn to the screen on top of the background version. This produces the illusion of the activated tab coming to the foreground. This effect can also be accomplished by having the handlers for the tabs use a shared state variable that governs the handler's redraw of the previously pressed tab.

Placing Buttons with Text in Them

You might have noticed that the text is mis-aligned in the button storyboard view (the edit button screen showing the three graphics for a button). This is because the text is rendered by the LCD controller's character generator at the text granularity of 6 pixels wide by 8 pixels tall. When moving a text button around to place it on a menu, the effect of this granularity is most evident. You can take advantage of this visual effect to optimize your button placement for properly aligned text.

Activate on Press versus Activate on Release

Most of the time, it is best for a button to activate (that is, invoke its button handler) when it is pressed. For example, if the button is set to beep and activate on press is specified, the operator will hear the beep as the button is being pressed; this is an intuitive and expected behavior. Repeating buttons (buttons that have the repeat flag set in the button behavior settings) should have their activation set for press. Otherwise, the repeating function doesn't make sense.

Buttons that change screens, however, should be set to activate on release. This prevents the screen from changing underneath the button that was pressed, which can be disorientating to an operator.

Another example of button activation strategy is to have both press and release activation. For example, imagine a button that activates a motor-driven robotic arm. The arm moves during the entire time that the button is depressed. In such a case, motor turn-on is initiated by the button press, and motor turn-off occurs when the button is released.

Conclusion

The best way to get to know the GUI Builder is to experiment with it. By downloading the **`gui_builder_w_demo.4th`** file, examining the pre-defined graphics and buttons, and playing with the tools, you'll see how easy it is to develop an intuitive graphical user interface. Please refer to the full GUI Builder users manual for lots more detailed information, and feel free to contact Mosaic Industries if you have any questions.