Required Hardware

- An LCD1100 Graphic LCD Phidget
- A 3-wire Phidget cable
- A VINT Hub
- A USB cable
- A computer

Connecting the Pieces

1. Connect the LCD1100 to the VINT Hub using the Phidget cable.
2. Connect the VINT Hub to your computer with a USB cable.

Testing Using Windows

Phidget Control Panel

In order to demonstrate the functionality of the LCD1100, the Phidget Control Panel running on a Windows machine will be used.

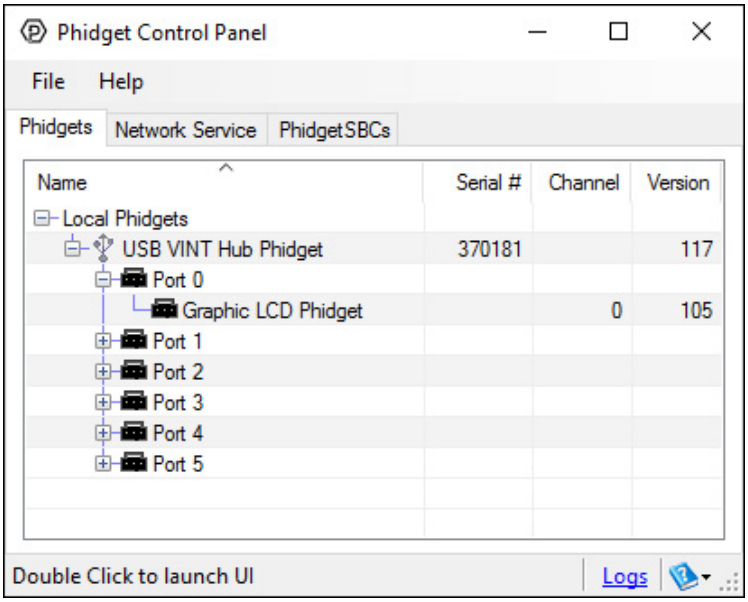
The Phidget Control Panel is available for use on both macOS and Windows machines. If you would like to follow along, first take a look at the getting started guide for your operating system:

- Getting started with Windows
- Getting started with macOS

Linux users can follow the getting started with Linux guide and continue reading here for more information about the LCD1100.

First Look

After plugging the LCD1100 into your computer and opening the Phidget Control Panel, you will see something like this:

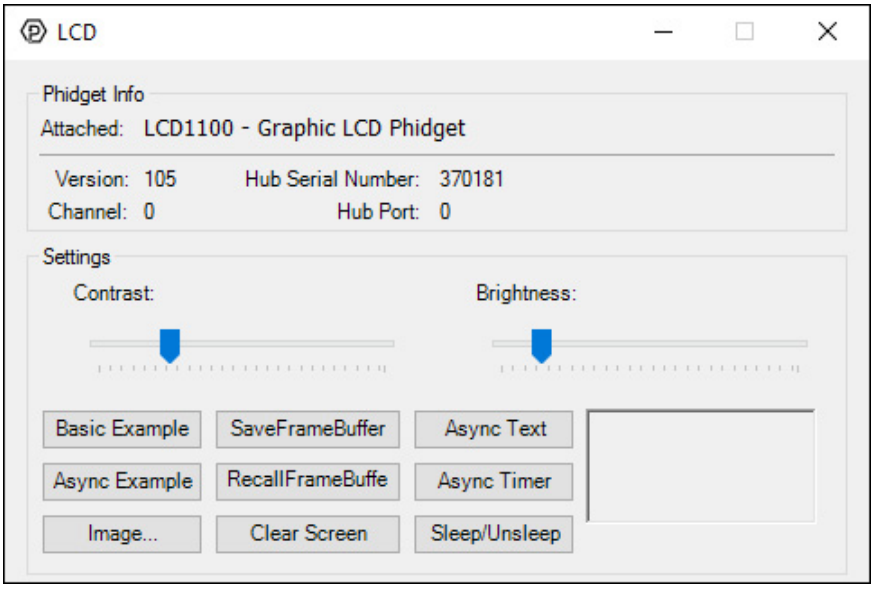


The Phidget Control Panel will list all connected Phidgets and associated objects, as well as the following information:

- Serial number: allows you to differentiate between similar Phidgets.
- Channel: allows you to differentiate between similar objects on a Phidget.
- Version number: corresponds to the firmware version your Phidget is running. If your Phidget is listed in red, your firmware is out of date. Update the firmware by double-clicking the entry.

The Phidget Control Panel can also be used to test your device. Double-clicking on an object will open an example.

Graphic LCD



When you double click on a Graphic LCD object, a window like the one pictured will open.

- At the top of the window, information about your device and the properties of this particular channel will be listed.
- In the middle, brightness and contrast of the LCD screen can be adjusted.
- At the bottom, there are a number of buttons to demonstrate the functionality of the graphics LCD:

- Basic Example: Write some text, lines, boxes and a bitmap to the screen.
- SaveFrameBuffer: Save the current screen to memory.
- RecallFrameBuffer: Load the saved screen from memory and print it to the screen.
- Async Example: Prints a bitmap to the screen asynchronously (can occur while other draw functions are still running).
- Async Text: Prints text to the screen asynchronously.
- Async Timer: As above, but constantly prints numbers in random locations.

- Image: Loads an image from file and converts it to a bitmap to be displayed on the screen.
- Sleep/Unsleep: Toggles sleep mode, where the Phidget is still attached but in a minimal power consumption state.
- Clear Screen: Erases the screen.

Testing Using Mac OS X

1. Go to the Quick Downloads section on the Mac OS X page.
2. Download and run the Phidget OS X Installer
3. Click on System Preferences >> Phidgets (under Other) to activate the Preference Pane
4. Make sure your device is properly attached
5. Double click on your device's objects in the listing to open them. The Preference Pane and examples will function very similarly to the ones described above in the Windows section.

Testing Using Linux

For a general step-by-step guide on getting Phidgets running on Linux, see the Linux page.

Using a Remote OS

We recommend testing your Phidget on a desktop OS before moving on to remote OS. Once you've tested your Phidget, you can go to the PhidgetSBC, or iOS pages to learn how to proceed.

Technical Section

Custom Characters

Bitmaps

Bitmaps define images to be drawn on the screen of the Graphic LCD display. Bitmaps on the Graphic LCD display are made up of pixels arranged in a grid with a size defined when the bitmap is drawn.

You can create a bitmap by defining a byte array of ones and zeroes. Ones are colored in, and zeroes are empty. If you put a line break after each row, it'll be easy to edit the bitmap.

In C#, this may look something like this:

```
Byte[] heart =  [0,0,0,0,0,
                 0,1,0,1,0,
                 1,1,1,1,1,
                 1,1,1,1,1,
                 0,1,1,1,0,
                 0,0,1,0,0,
                 0,0,0,0,0,
                 0,0,0,0,0];

gLCD.WriteBitmap(0, 0, 5, 8, heart);
```

Custom Characters

Custom characters are images associated with given unicode characters. A custom character can be any arrangement of pixels within the space allotted for a single character. Single characters are made up of pixels arranged in a grid with a size defined by `setFontSize()`.

As with regular bitmaps for the Graphic LCD display, you can create a character bitmap by defining a byte array of ones and zeroes. Ones are colored in, and zeroes are empty. If you put a line break after each row, it'll be easy to edit the bitmap.

In C#, this may look something like this:

```
Byte[] heart =  [0,0,0,0,0,
                 0,1,0,1,0,
                 1,1,1,1,1,
                 1,1,1,1,1,
                 0,1,1,1,0,
```

```
                0,0,1,0,0,
                0,0,0,0,0,
                0,0,0,0,0];

gLCD.SetFontSize(LCDFont.User2, 5, 8);
gLCD.SetCharacterBitmap(LCDFont.User2,"\x6",heart);
```

Once stored, characters can be recalled into a text string by using the unicode value for the location (in this example, "\x6"). For example, in C#:

```
gLCD.WriteText(LCDFont.Dimensions_5x8, 0, 0,"I \x6 Phidgets!");
```

Custom characters on the LCD1100 are stored as images on the frame buffer for their font.`FONT_User1` is stored on frame buffer 1 and `FONT_User2` is on frame buffer 2.

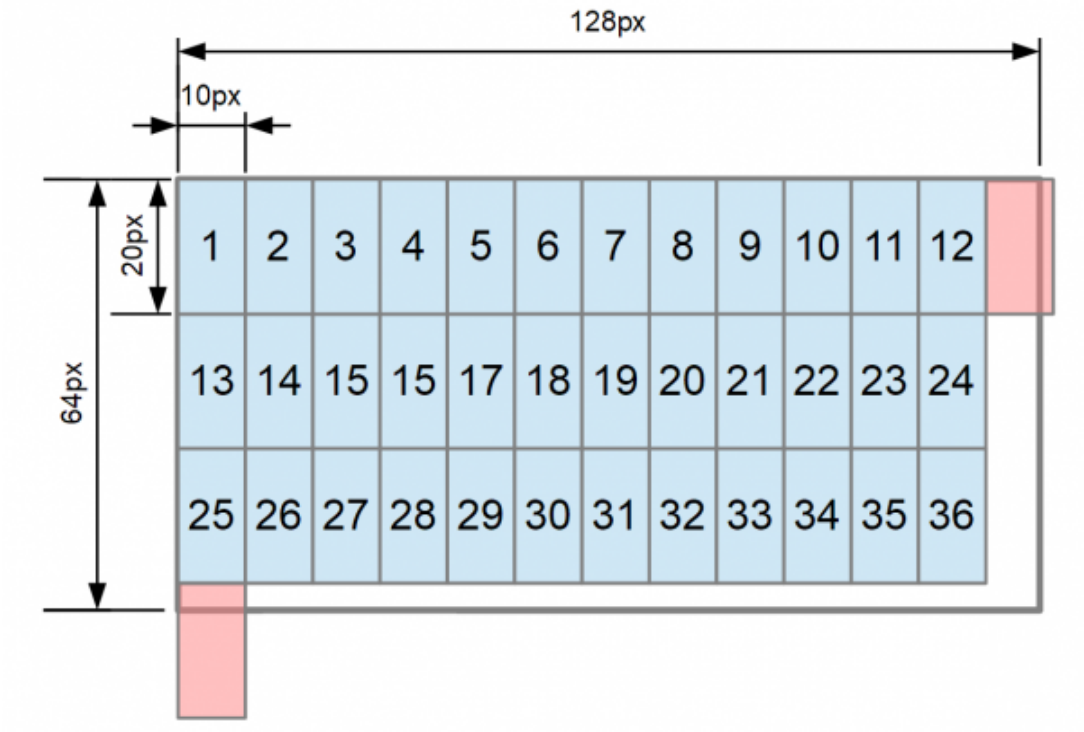
These characters occupy the same space as drawings on their framebuffer, and will be displayed onscreen if their framebuffer is flushed. They can also be overwritten by using drawing funcitons on their framebuffer, so it is recommended to only use a given framebuffer either for drawing or storing fonts, but not both.

We recommend using `FONT_User2` (frame buffer 2) to store custom fonts, as it can be saved for later use.

In order to use the custom fonts, you must first define their size with `setFontSize()`. Once the font size is set, custom characters will be placed on the font's frame buffer at a location corresponding to the character number provided.

Characters for each font are stored in rows ordered left-to-right, top-to-bottom. Rows are filled with as many characters as will completely fit across the width of the screen. There are as many rows as will fit on the screen vertically.

On a screen 128 pixels wide by 64 pixels high, if your font is 10 pixels wide by 20 high, you will have 3 rows of 12 characters. This allows for a maximum of 36 characters of that size.



A demonstration of how many 10x20px characters can fit on the screen.

To quickly determine how many characters can be in your custom font, you can call`getMaxCharacters()` in your code.

Custom character indexing starts with character 0x01 and can be any character between 0x01 and the maximum number of characters that fit on screen.

To determine if an ascii character can be used in a given custom font, you can look at its corresponding ascii value on an ascii table to determine if it is within the limit determined above.

What to do Next

- Software Overview - Find your preferred programming language here to learn how to write your own code with Phidgets!
- General Phidget Programming - Read this general guide to the various aspects of programming with Phidgets. Learn how to log data into a spreadsheet, use Phidgets over the network, and much more.

- Phidget22 API - The API is a universal library of all functions and definitions for programming with Phidgets. Just select your language and device and it'll give you a complete list of all properties, methods, events, and enumerations that are at your disposal.