# Adafruit GFX Library Build Status

This is the core graphics library for all our displays, providing a common set of graphics primitives (points, lines, circles, etc.). It needs to be paired with a hardware-specific library for each display device we carry (to handle the lower-level functions).

Adafruit invests time and resources providing this open source code, please support Adafruit and open-source hardware by purchasing products from Adafruit!

Written by Limor Fried/Ladyada for Adafruit Industries. BSD license, check license.txt for more information. All text above must be included in any redistribution.

Recent Arduino IDE releases include the Library Manager for easy installation. Otherwise, to download, click the DOWNLOAD ZIP button, uncompress and rename the uncompressed folder Adafruit\_GFX. Confirm that the Adafruit\_GFX folder contains Adafruit\_GFX.cpp and Adafruit\_GFX.h. Place the Adafruit\_GFX library folder your ArduinoSketchFolder/Libraries/ folder. You may need to create the Libraries subfolder if its your first library. Restart the IDE.

**You will also need to install the latest Adafruit BusIO library.** Search for "Adafruit BusIO" in the library manager, or install by hand from <https://github.com/adafruit/Adafruit_BusIO>

# Useful Resources

* Image2Code: This is a handy Java GUI utility to convert a BMP file into the array code necessary to display the image with the drawBitmap function. Check out the code at ehubin's GitHub repository: <https://github.com/ehubin/Adafruit-GFX-Library/tree/master/Img2Code>
* drawXBitmap function: You can use the GIMP photo editor to save a .xbm file and use the array saved in the file to draw a bitmap with the drawXBitmap function. See the pull request here for more details: <https://github.com/adafruit/Adafruit-GFX-Library/pull/31>
* 'Fonts' folder contains bitmap fonts for use with recent (1.1 and later) Adafruit\_GFX. To use a font in your Arduino sketch, #include the corresponding .h file and pass address of GFXfont struct to setFont(). Pass NULL to revert to 'classic' fixed-space bitmap font.
* 'fontconvert' folder contains a command-line tool for converting TTF fonts to Adafruit\_GFX header format.
* You can also use [this GFX Font Customiser tool](https://github.com/tchapi/Adafruit-GFX-Font-Customiser) (*web version* [*here*](https://tchapi.github.io/Adafruit-GFX-Font-Customiser/)) to customize or correct the output from [fontconvert](https://github.com/adafruit/Adafruit-GFX-Library/tree/master/fontconvert), and create fonts with only a subset of characters to optimize size.

### Roadmap

The PRIME DIRECTIVE is to maintain backward compatibility with existing Arduino sketches -- many are hosted elsewhere and don't track changes here, some are in print and can never be changed! This "little" library has grown organically over time and sometimes we paint ourselves into a design corner and just have to live with it or add progressively more ungainly workarounds.

**We are grateful for everyone's contributions, but pull requests for the following will NOT be merged:**

* Additional or incompatible font formats (see Prime Directive above). There are already two formats and the code is quite bloaty there as it is. This also creates liabilities for tools and documentation. What's there isn't perfect but it does the job.
* Additional or incompatible bitmap formats, for similar reasons. It's getting messy.
* Adding background color to custom fonts to erase prior screen contents. The ONLY acceptable methods are to clear the area with a filled rect, or (to avoid flicker) draw text into a GFXcanvas1 and copy to screen with drawBitmap() w/background color. This is on purpose and by design. We've discussed this. Glyphs can overlap.
* Scrolling, whether hardware- or software-based. Such implementations tend to rely on hardware-specific features (not universally available), read access to the screen's framebuffer (ditto) and/or the addition of virtual functions in GFX which them must be added in *every* subclass, of which there are many. The GFX API is largely "set" at this point and this is just a limitation we live with now.
* Please don't reformat code for the sake of reformatting code. The resulting large "visual diff" makes it impossible to untangle actual bug fixes from merely rearranged lines. clang-format will be the final arbiter.
* Please no more pentagram-drawing PRs. Any oddly-specific drawing functions can go in your own code and aren't helpful in a library context.

If you *must* have one of these features, consider creating a fork with the features required for your project...it's easy to keep synced with the upstream code.