

---

# **CircuitPython Documentation**

***Release 6.3.0***

**CircuitPython Contributors**

**Jun 10, 2021**



## API AND USAGE

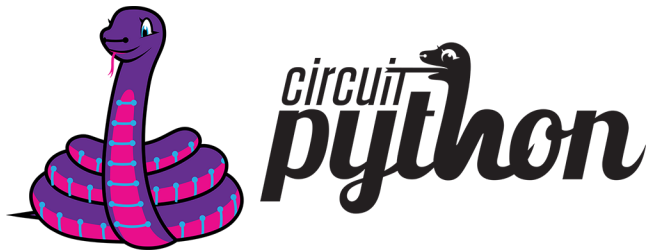
<b>1</b>	<b>CircuitPython</b>	<b>3</b>
1.1	Get CircuitPython	3
1.2	Documentation	4
1.3	Code Search	4
1.4	Contributing	4
1.5	Branding	4
1.6	Differences from MicroPython	5
1.6.1	Behavior	5
1.6.2	API	5
1.6.3	Modules	6
1.7	Project Structure	6
1.7.1	Core	6
1.7.2	Ports	6
1.8	Full Table of Contents	7
1.8.1	Core Modules	7
1.8.2	Supported Ports	173
1.8.3	Troubleshooting	182
1.8.4	Additional CircuitPython Libraries and Drivers on GitHub	183
1.8.5	Design Guide	183
1.8.6	Architecture	193
1.8.7	Porting	193
1.8.8	Adding <code>*i/o</code> support to other ports	195
1.8.9	MicroPython libraries	197
1.8.10	CircuitPython	228
1.8.11	Contributing	232
1.8.12	Building CircuitPython	233
1.8.13	Building	234
1.8.14	Testing	234
1.8.15	Debugging	235
1.8.16	Code Quality Checks	235
1.8.17	Adafruit Community Code of Conduct	235
1.8.18	MicroPython & CircuitPython license information	238
1.8.19	WebUSB Serial Support	238
<b>2</b>	<b>Indices and tables</b>	<b>241</b>
	<b>Python Module Index</b>	<b>243</b>
	<b>Index</b>	<b>245</b>



Welcome to the API reference documentation for Adafruit CircuitPython. This contains low-level API reference docs which may link out to separate “*getting started*” guides. [Adafruit](#) has many excellent tutorials available through the [Adafruit Learning System](#).



## CIRCUITPYTHON



[circuitpython.org](https://circuitpython.org) | [Get CircuitPython](#) | [Documentation](#) | [Contributing](#) | [Branding](#) | [Differences from Micropython](#) | [Project Structure](#)

**CircuitPython** is a *beginner friendly*, open source version of Python for tiny, inexpensive computers called microcontrollers. Microcontrollers are the brains of many electronics including a wide variety of development boards used to build hobby projects and prototypes. CircuitPython in electronics is one of the best ways to learn to code because it connects code to reality. Simply install CircuitPython on a supported board via drag and drop and then edit a `code.py` file on the CIRCUITPY drive. The code will automatically reload. No software installs are needed besides a text editor (we recommend [Mu](#) for beginners.)

CircuitPython features unified Python core APIs and a growing list of 150+ device libraries and drivers that work with it. These libraries also work on single board computers with regular Python via the [Adafruit Blinka Library](#).

CircuitPython is based on [MicroPython](#). See [below](#) for differences. CircuitPython development is sponsored by [Adafruit](#) and is available on their educational development boards. Please support both MicroPython and Adafruit.

### 1.1 Get CircuitPython

Official binaries for all supported boards are available through [circuitpython.org/downloads](https://circuitpython.org/downloads). The site includes stable, unstable and continuous builds. Full release notes and assets are available through [GitHub releases](#) as well.

## 1.2 Documentation

Guides and videos are available through the [Adafruit Learning System](#) under the [CircuitPython](#) category. An API reference is also available on [Read the Docs](#). A collection of awesome resources can be found at [Awesome CircuitPython](#).

Specifically useful documentation when starting out:

- [Welcome to CircuitPython](#)
- [CircuitPython Essentials](#)
- [Example Code](#)

## 1.3 Code Search

GitHub doesn't currently support code search on forks. Therefore, CircuitPython doesn't have code search through GitHub because it is a fork of MicroPython. Luckily, [SourceGraph](#) has free code search for public repos like CircuitPython. So, visit [sourcegraph.com/github.com/adafruit/circuitpython](https://sourcegraph.com/github.com/adafruit/circuitpython) to search the CircuitPython codebase online.

## 1.4 Contributing

See [CONTRIBUTING.md](#) for full guidelines but please be aware that by contributing to this project you are agreeing to the [Code of Conduct](#). Contributors who follow the [Code of Conduct](#) are welcome to submit pull requests and they will be promptly reviewed by project admins. Please join the [Discord](#) too.

## 1.5 Branding

While we are happy to see CircuitPython forked and modified, we'd appreciate it if forked releases not use the name "CircuitPython" or the Blinka logo. "CircuitPython" means something special to us and those who learn about it. As a result, we'd like to make sure products referring to it meet a common set of requirements.

If you'd like to use the term "CircuitPython" and Blinka for your product here is what we ask:

- Your product is supported by the primary "[adafruit/circuitpython](#)" repo. This way we can update any custom code as we update the CircuitPython internals.
- Your product is listed on [circuitpython.org](https://circuitpython.org) (source [here](#)). This is to ensure that a user of your product can always download the latest version of CircuitPython from the standard place.
- Your product has a user accessible USB plug which appears as a CIRCUITPY drive when plugged in.

If you choose not to meet these requirements, then we ask you call your version of CircuitPython something else (for example, SuperDuperPython) and not use the Blinka logo. You can say it is "CircuitPython-compatible" if most CircuitPython drivers will work with it.



## 1.6 Differences from MicroPython

CircuitPython:

- Supports native USB on all boards, allowing file editing without special tools.
- Floats (aka decimals) are enabled for all builds.
- Error messages are translated into 10+ languages.
- Does not support concurrency within Python (including interrupts and threading). Some concurrency is achieved with native modules for tasks that require it such as audio file playback.

### 1.6.1 Behavior

- The order that files are run and the state that is shared between them. CircuitPython's goal is to clarify the role of each file and make each file independent from each other.
- `boot.py` (or `settings.py`) runs only once on start up before USB is initialized. This lays the ground work for configuring USB at startup rather than it being fixed. Since serial is not available, output is written to `boot_out.txt`.
- `code.py` (or `main.py`) is run after every reload until it finishes or is interrupted. After it is done running, the vm and hardware is reinitialized. **This means you cannot read state from `code.py` in the REPL anymore, as the REPL is a fresh vm.** CircuitPython's goal for this change includes reducing confusion about pins and memory being used.
- After the main code is finished the REPL can be entered by pressing any key.
- Autoreload state will be maintained across reload.
- Adds a safe mode that does not run user code after a hard crash or brown out. The hope is that this will make it easier to fix code that causes nasty crashes by making it available through mass storage after the crash. A reset (the button) is needed after it's fixed to get back into normal mode.
- RGB status LED indicating CircuitPython state, and errors through a sequence of colored flashes.
- Re-runs `code.py` or other main file after file system writes over USB mass storage. (Disable with `supervisor.disable_autoreload()`)
- Autoreload is disabled while the REPL is active.
- Main is one of these: `code.txt`, `code.py`, `main.py`, `main.txt`
- Boot is one of these: `settings.txt`, `settings.py`, `boot.py`, `boot.txt`

### 1.6.2 API

- Unified hardware APIs. Documented on [ReadTheDocs](#).
- API docs are rST within the C files in `shared-bindings`.
- No machine API.

### 1.6.3 Modules

- No module aliasing. (`uos` and `utime` are not available as `os` and `time` respectively.) Instead `os`, `time`, and `random` are CPython compatible.
  - New `storage` module which manages file system mounts. (Functionality from `uos` in MicroPython.)
  - Modules with a CPython counterpart, such as `time`, `os` and `random`, are strict subsets of their CPython version. Therefore, code from CircuitPython is runnable on CPython but not necessarily the reverse.
  - tick count is available as `time.monotonic()`
- 

## 1.7 Project Structure

Here is an overview of the top-level source code directories.

### 1.7.1 Core

The core code of MicroPython is shared amongst ports including CircuitPython:

- `docs` High level user documentation in Sphinx reStructuredText format.
- `drivers` External device drivers written in Python.
- `examples` A few example Python scripts.
- `extmod` Shared C code used in multiple ports' modules.
- `lib` Shared core C code including externally developed libraries such as FATFS.
- `logo` The CircuitPython logo.
- `mpy-cross` A cross compiler that converts Python files to byte code prior to being run in MicroPython. Useful for reducing library size.
- `py` Core Python implementation, including compiler, runtime, and core library.
- `shared-bindings` Shared definition of Python modules, their docs and backing C APIs. Ports must implement the C API to support the corresponding module.
- `shared-module` Shared implementation of Python modules that may be based on `common-hal`.
- `tests` Test framework and test scripts.
- `tools` Various tools, including the `pyboard.py` module.

### 1.7.2 Ports

Ports include the code unique to a microcontroller line and also variations based on the board.

Supported	Support status
atmel-samd	SAMD21 stable   SAMD51 stable
cxd56	stable
esp32s2	beta
litex	alpha
mimxrt10xx	alpha
nrf	stable
raspberrypi	beta
stm	F4 stable   others beta
unix	alpha

- **stable** Highly unlikely to have bugs or missing functionality.
- **beta** Being actively improved but may be missing functionality and have bugs.
- **alpha** Will have bugs and missing functionality.

The remaining port directories not listed above are in the repo to maintain compatibility with the [MicroPython](#) parent project.

[Back to Top](#)

## 1.8 Full Table of Contents

### 1.8.1 Core Modules

These core modules are intended on being consistent across ports and boards. A module may not exist on a port/board if no underlying hardware support is present or if flash space is limited. For example, a microcontroller without analog features will not have *analogio*. See the [Module Support Matrix - Which Modules Are Available on Which Boards](#) page for a list of modules supported on each board.

#### Module Support Matrix - Which Modules Are Available on Which Boards

The following table lists the available built-in modules for each CircuitPython capable board.

Board	Modules Available
8086 Com-mander	<i>analogio, board, busio, digitalio, gamepad, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid</i>
@sarfata shIRtty	<i>analogio, board, busio, digitalio, i2cperipheral, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, usb_hid, usb_midi</i>
Adafruit BLM Badge	<i>analogio, audiobusio, audiocore, audioio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, random, storage, struct, supervisor, time, touchio, usb_hid</i>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Adafruit Circuit Playground Bluefruit	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
Adafruit Circuit-Play-ground Express	<code>_pixelbuf</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audioio</code> , <code>bitbangio</code> , <code>board</code> , <code>busio</code> , <code>countio</code> , <code>digitalio</code> , <code>errno</code> , <code>gamepad</code> , <code>math</code> , <code>microcontroller</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rotaryio</code> , <code>rtc</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>time</code> , <code>touchio</code> , <code>usb_hid</code> , <code>usb_midi</code>
Adafruit Circuit-Play-ground Express with Crickit libraries	<code>_pixelbuf</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audioio</code> , <code>board</code> , <code>busio</code> , <code>countio</code> , <code>digitalio</code> , <code>errno</code> , <code>gamepad</code> , <code>math</code> , <code>microcontroller</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>time</code> , <code>touchio</code> , <code>usb_hid</code> , <code>usb_midi</code>
Adafruit Circuit-Play-ground Express with displayio	<code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audioio</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>gamepad</code> , <code>math</code> , <code>microcontroller</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>usb_hid</code> , <code>usb_midi</code>
Adafruit CLUE nRF52840 Express	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
Adafruit Feather Bluefruit Sense	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
Adafruit Feather M0 Adalogger	<code>analogio</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>math</code> , <code>microcontroller</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>rotaryio</code> , <code>rtc</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>time</code> , <code>touchio</code> , <code>usb_hid</code> , <code>usb_midi</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Adafruit Feather M0 Basic	<i>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</i>
Adafruit Feather M0 Express	<i>_pixelbuf, analogio, audiobusio, audiocore, audioio, board, busio, digitalio, displayio, errno, gamepad, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi</i>
Adafruit Feather M0 Express with Crickit libraries	<i>_pixelbuf, analogio, audiobusio, audiocore, audioio, board, busio, countio, digitalio, errno, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</i>
Adafruit Feather M0 RFM69	<i>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, random, storage, struct, supervisor, time</i>
Adafruit Feather M0 RFM9x	<i>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, random, storage, struct, supervisor, time</i>
Adafruit Feather M4 CAN	<i>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</i>
Adafruit Feather M4 Express	<i>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</i>
Adafruit Feather MIMXRT1011	<i>_bleio, _pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, os, pulseio, pwmio, random, re, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</i>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Adafruit Feather nRF52840 Express	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
Adafruit Feather RadioFruit Zigbee	<code>_pixelbuf</code> , <code>analogio</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>gamepad</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rotaryio</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code>
Adafruit Feather RP2040	<code>_bleio</code> , <code>_pixelbuf</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>bitops</code> , <code>board</code> , <code>busio</code> , <code>countio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
Adafruit Feather STM32F405 Express	<code>_bleio</code> , <code>_pixelbuf</code> , <code>analogio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>canio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>sdcardio</code> , <code>sdioio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>vectorio</code>
Adafruit FunHouse	<code>_bleio</code> , <code>_pixelbuf</code> , <code>alarm</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>canio</code> , <code>countio</code> , <code>digitalio</code> , <code>displayio</code> , <code>dualbank</code> , <code>errno</code> , <code>framebufferio</code> , <code>frequencyio</code> , <code>gamepad</code> , <code>ipaddress</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>ps2io</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>socketpool</code> , <code>ssl</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>vectorio</code> , <code>watchdog</code> , <code>wifi</code>
Adafruit Gemma M0	<code>analogio</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>math</code> , <code>microcontroller</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>rotaryio</code> , <code>rtc</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>time</code> , <code>touchio</code> , <code>usb_hid</code> , <code>usb_midi</code>
Adafruit Grand Central M4 Express	<code>_bleio</code> , <code>_pixelbuf</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audioio</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>countio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>frequencyio</code> , <code>gamepad</code> , <code>i2cperipheral</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>ps2io</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sdioio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Adafruit Hallow- ing M4 Express	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Adafruit ItsyBitsy M0 Ex- press	<code>_pixelbuf, analogio, audiocore, audioio, board, busio, digitalio, displayio, errno, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi</code>
Adafruit ItsyBitsy M4 Ex- press	<code>_bleio, _pixelbuf, analogio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Adafruit ItsyBitsy nRF52840 Express	<code>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
Adafruit ItsyBitsy RP2040	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, bitops, board, busio, countio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
Adafruit MagTag	<code>_bleio, _pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, vectorio, watchdog, wifi</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Adafruit Matrix Portal M4	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Adafruit Metro ESP32S2	<code>_bleio, _pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, vectorio, watchdog, wifi</code>
Adafruit Metro M0 Express	<code>_pixelbuf, analogio, audiobusio, audiocore, audioio, board, busio, digitalio, displayio, errno, gamepad, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi</code>
Adafruit Metro M4 Airlift Lite	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Adafruit Metro M4 Express	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Adafruit Metro nRF52840 Express	<code>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>

continues on next page



Table 1 – continued from previous page

Board	Modules Available
Adafruit Monster M4SK	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Adafruit NeoKey Trinkey M0	<code>_pixelbuf, board, digitalio, math, microcontroller, neopixel_write, nvm, os, random, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
Adafruit NeoPixel Trinkey M0	<code>_pixelbuf, board, digitalio, math, microcontroller, neopixel_write, nvm, os, random, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
Adafruit pIRKey M0	<code>board, busio, digitalio, microcontroller, nvm, os, pulseio, pwmio, random, storage, struct, supervisor, time, usb_hid, usb_midi</code>
Adafruit ProxLight Trinkey M0	<code>_pixelbuf, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, random, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
Adafruit Pybadge	<code>_bleio, _pixelbuf, _stage, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, gamepadshift, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Adafruit Pybadge Airlift	<code>_bleio, _pixelbuf, _stage, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, gamepadshift, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Adafruit PyGamer	<code>_bleio, _pixelbuf, _stage, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, gamepadshift, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Adafruit PyPortal	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Adafruit PyPortal Titano	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Adafruit PyRuler	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
Adafruit QT Py M0	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
Adafruit QT Py M0 Haxpress	<code>_pixelbuf, analogio, audiocore, audioio, board, busio, digitalio, displayio, errno, gamepad, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi, vectorio</code>
Adafruit QT2040 Trinkey	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, bitops, board, busio, countio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
Adafruit QTPy RP2040	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, bitops, board, busio, countio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
Adafruit Rotary Trinkey M0	<code>_pixelbuf, board, digitalio, math, microcontroller, neopixel_write, nvm, os, random, rotaryio, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Adafruit Slide Trickey M0	<code>_pixelbuf, analogio, board, digitalio, math, microcontroller, neopixel_write, nvm, os, random, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
Adafruit Trellis M4 Express	<code>_bleio, _pixelbuf, analogio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Adafruit Trinket M0	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
AloriumTech Evo M51	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
ARAM-CON Badge 2019	<code>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
ARAM-CON2 Badge	<code>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
Arduino MKR Zero	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
Arduino MKR1300	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Arduino Nano 33 BLE	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
Arduino Nano 33 IoT	<code>analogio</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>math</code> , <code>microcontroller</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>rotaryio</code> , <code>rtc</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>time</code> , <code>touchio</code> , <code>usb_hid</code> , <code>usb_midi</code>
Arduino Nano RP2040 Connect	<code>_bleio</code> , <code>_pixelbuf</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>bitops</code> , <code>board</code> , <code>busio</code> , <code>countio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
Arduino Zero	<code>analogio</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>math</code> , <code>microcontroller</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>rotaryio</code> , <code>rtc</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>time</code> , <code>touchio</code> , <code>usb_hid</code> , <code>usb_midi</code>
Artisense Reference Design RD00	<code>_bleio</code> , <code>_pixelbuf</code> , <code>alarm</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>canio</code> , <code>countio</code> , <code>digitalio</code> , <code>displayio</code> , <code>dualbank</code> , <code>errno</code> , <code>framebufferio</code> , <code>frequencyio</code> , <code>gamepad</code> , <code>ipaddress</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>ps2io</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>socketpool</code> , <code>ssl</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>vectorio</code> , <code>watchdog</code> , <code>wifi</code>
Atelier-DuMaker nRF52840 Breakout	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
BastBLE	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
BastWiFi	<i>_bleio, _pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, vectorio, watchdog, wifi</i>
BDMI-CRO VINA-D21	<i>_pixelbuf, analogio, audiobusio, audiocore, audioio, board, busio, digitalio, displayio, errno, gamepad, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi</i>
BDMI-CRO VINA-D51	<i>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</i>
BLE-SS dev board Multi Sensor	<i>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</i>
Capable Robot Pro- grammable USB Hub	<i>_bleio, _pixelbuf, analogio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</i>
Cedar Grove String- Car M0 Express	<i>_pixelbuf, analogio, audiocore, audioio, board, busio, countio, digitalio, displayio, errno, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi</i>
Circuit- Brains Basic	<i>_pixelbuf, analogio, audiobusio, audiocore, audioio, board, busio, digitalio, displayio, errno, gamepad, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi</i>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Circuit-Brains Deluxe	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
CP Sapling M0	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
CP Sapling M0 w/ SPI Flash	<code>_pixelbuf, analogio, board, busio, digitalio, displayio, errno, gamepad, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi, vectorio</code>
CP32-M4	<code>_bleio, _pixelbuf, analogio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
DynaLoRa_USB	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
DynOSSAT-EDU-EPS	<code>_pixelbuf, analogio, audiobusio, audiocore, audioio, board, busio, digitalio, errno, gamepad, i2cperipheral, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rtc, storage, struct, supervisor, time, usb_hid, usb_midi</code>
DynOSSAT-EDU-OBC	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Electronic Cats Bast Pro Mini M0	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
Electronic Cats Cat-WAN USBStick	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Electronic Cats Hunter Cat NFC	<i>analogio, board, busio, digitalio, math, microcontroller, nvm, os, pulseio, pwmio, random, storage, struct, supervisor, time, usb_hid, usb_midi</i>
Electronic Cats NFC Copy Cat	<i>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</i>
Electronut Labs Blip	<i>_bleio, _pixelbuf, _stage, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</i>
Electronut Labs Papyr	<i>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</i>
Escornabot Makech	<i>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</i>
Espruino Pico	<i>_bleio, _pixelbuf, analogio, binascii, bitbangio, board, busio, digitalio, displayio, errno, gamepad, json, math, microcontroller, msgpack, neopixel_write, os, pulseio, pwmio, random, re, sdcardio, storage, struct, supervisor, terminalio, time, touchio, vectorio</i>
Espruino Wifi	<i>_bleio, _pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, os, pulseio, pwmio, random, re, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, vectorio</i>
Feather ESP32S2 without PSRAM	<i>_bleio, _pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, vectorio, watchdog, wifi</i>

continues on next page



Table 1 – continued from previous page

Board	Modules Available
Feather MIMXRT1011	<code>_bleio</code> , <code>_pixelbuf</code> , <code>analogio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rtc</code> , <code>sdcario</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code>
Feather MIMXRT1062	<code>_bleio</code> , <code>_pixelbuf</code> , <code>analogio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rtc</code> , <code>sdcario</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code>
FeatherS2	<code>_bleio</code> , <code>_pixelbuf</code> , <code>alarm</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>canio</code> , <code>countio</code> , <code>digitalio</code> , <code>displayio</code> , <code>dualbank</code> , <code>errno</code> , <code>framebufferio</code> , <code>frequencyio</code> , <code>gamepad</code> , <code>ipaddress</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>ps2io</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcario</code> , <code>sharpdisplay</code> , <code>socketpool</code> , <code>ssl</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>vectorio</code> , <code>watchdog</code> , <code>wifi</code>
FeatherS2 PreRelease	<code>_bleio</code> , <code>_pixelbuf</code> , <code>alarm</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>canio</code> , <code>countio</code> , <code>digitalio</code> , <code>displayio</code> , <code>dualbank</code> , <code>errno</code> , <code>framebufferio</code> , <code>frequencyio</code> , <code>gamepad</code> , <code>ipaddress</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>ps2io</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcario</code> , <code>sharpdisplay</code> , <code>socketpool</code> , <code>ssl</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>vectorio</code> , <code>watchdog</code> , <code>wifi</code>
Fluff M0	<code>analogio</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>math</code> , <code>microcontroller</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>rotaryio</code> , <code>rtc</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>time</code> , <code>touchio</code> , <code>usb_hid</code> , <code>usb_midi</code>
Fomu	<code>_pixelbuf</code> , <code>binascii</code> , <code>digitalio</code> , <code>errno</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>os</code> , <code>random</code> , <code>re</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code>
Franzin- inho WIFI w/Wroom	<code>_bleio</code> , <code>_pixelbuf</code> , <code>alarm</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>canio</code> , <code>countio</code> , <code>digitalio</code> , <code>displayio</code> , <code>dualbank</code> , <code>errno</code> , <code>framebufferio</code> , <code>frequencyio</code> , <code>gamepad</code> , <code>ipaddress</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>ps2io</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcario</code> , <code>sharpdisplay</code> , <code>socketpool</code> , <code>ssl</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>vectorio</code> , <code>watchdog</code> , <code>wifi</code>

continues on next page



Table 1 – continued from previous page

Board	Modules Available
Franzin- inho WIFI w/Wrover	<i>_bleio, _pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, vectorio, watchdog, wifi</i>
Hacked Feather M0 Ex- press with 8Mbyte SPI flash	<i>_pixelbuf, analogio, audiobusio, audiocore, audioio, board, busio, digitalio, displayio, errno, gamepad, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi</i>
HalloW- ing M0 Express	<i>_pixelbuf, analogio, audiocore, audioio, bitmaptools, board, busio, digitalio, displayio, errno, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi</i>
HiiBot BlueFi	<i>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</i>
Iki- gaiSense Vita nRF52840	<i>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</i>
iMX RT 1020 EVK	<i>_bleio, _pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, os, pulseio, pwmio, random, re, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</i>
iMX RT 1060 EVK	<i>_bleio, _pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, os, pulseio, pwmio, random, re, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</i>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
IMXRT1010-EVK	<code>_bleio, _pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, os, pulseio, pwmio, random, re, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
J&J Studios datum-Distance	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
J&J Studios datum-IMU	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
J&J Studios datum-Light	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
J&J Studios datum-Weather	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
Kaluga 1	<code>_bleio, _pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_midi, vectorio, watchdog, wifi</code>
keithp.com snekboard	<code>_pixelbuf, analogio, audiobusio, audiocore, audioio, board, busio, countio, digitalio, displayio, errno, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi</code>
LILYGO TTGO T8 ESP32-S2 w/Display	<code>_bleio, _pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, vectorio, watchdog, wifi</code>
LoC BeR M4 base board	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Maker-diary M60 Keyboard	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
Mak-erdiary nRF52840 M.2 De-veloper Kit	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
Maker-Diary nRF52840 MDK	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
Maker-Diary nRF52840 MDK USB Dongle	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
Mak-erdiary Pitaya Go	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
MDBT50Q-DB-40	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Meow Meow	<i>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</i>
MEOW-BIT	<i>_bleio, _pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, os, pulseio, pwmio, random, re, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, vectorio</i>
Metro MIMXRT1011	<i>_bleio, _pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, os, pulseio, pwmio, random, re, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</i>
microS2	<i>_bleio, _pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, vectorio, watchdog, wifi</i>
Mini SAM M4	<i>_bleio, _pixelbuf, analogio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</i>
nanoESP32-S2	<i>_bleio, _pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, vectorio, watchdog, wifi</i>
ndGarage[n°] Bit6: FeatherSnow- v2	<i>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</i>
ndGarage[n°] Bit6: FeatherSnow	<i>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</i>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
nice!nano	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
NUCLEO STM32F746	<code>_bleio</code> , <code>_pixelbuf</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>vectorio</code>
NUCLEO STM32F767	<code>_bleio</code> , <code>_pixelbuf</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>vectorio</code>
NUCLEO STM32H743	<code>_bleio</code> , <code>_pixelbuf</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>os</code> , <code>random</code> , <code>re</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>vectorio</code>
Open Hardware Summit 2020 Badge	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>
OPENMV-H7 R1	<code>_bleio</code> , <code>_pixelbuf</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>os</code> , <code>random</code> , <code>re</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>vectorio</code>
Particle Argon	<code>_bleio</code> , <code>_pixelbuf</code> , <code>aesio</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>audiomixer</code> , <code>audiomp3</code> , <code>audiopwmio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rgbmatrix</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcardio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code> , <code>watchdog</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Particle Boron	<code>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
Particle Xenon	<code>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
PCA10056 nRF52840-DK	<code>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
PCA10059 nRF52840 Dongle	<code>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
PCA10100 nRF52833 Dongle	<code>_bleio, analogio, audiobusio, audiocore, audiomixer, audiopwmio, binascii, board, busio, digitalio, errno, gamepad, json, math, microcontroller, os, pulseio, pwmio, random, re, rtc, storage, struct, supervisor, time, usb_hid, usb_midi, watchdog</code>
PewPew 10.2	<code>_pew, analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, storage, struct, supervisor, time, touchio, usb_hid</code>
PewPew M4	<code>_stage, analogio, audiocore, audioio, audiomixer, board, busio, digitalio, displayio, gamepad, math, microcontroller, nvm, os, pulseio, pwmio, random, storage, struct, supervisor, terminalio, time</code>
PicoPlanet	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>

continues on next page



Table 1 – continued from previous page

Board	Modules Available
Pimoroni Keybow 2040	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, bitops, board, busio, countio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
Pimoroni Pico LiPo (16MB)	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, bitops, board, busio, countio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
Pimoroni Pico LiPo (4MB)	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, bitops, board, busio, countio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
Pimoroni PicoSystem	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, bitops, board, busio, countio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
Pimoroni Tiny 2040	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, bitops, board, busio, countio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
PYB LR Nano V2	<code>_bleio, _pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, os, pulseio, pwmio, random, re, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, vectorio</code>
Py-boardV1_1	<code>_bleio, _pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, canio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, os, pulseio, pwmio, random, re, rgbmatrix, sdcardio, sdioio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, vectorio</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Py-Cubedv04	<code>_bleio, _pixelbuf, analogio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, board, busio, countio, digitalio, errno, frequencyio, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, storage, struct, supervisor, time, touchio, ulab, usb_hid, usb_midi</code>
PyCubedv04-MRAM	<code>_bleio, _pixelbuf, analogio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, board, busio, countio, digitalio, errno, frequencyio, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, storage, struct, supervisor, time, touchio, ulab, usb_hid, usb_midi</code>
Raspberry Pi Pico	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, bitops, board, busio, countio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
Robo HAT MM1 M4	<code>_bleio, _pixelbuf, analogio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, board, busio, countio, digitalio, errno, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, storage, struct, supervisor, time, touchio, ulab, usb_hid, usb_midi</code>
SAM E54 Xplained Pro	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sdioio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
SAM32v26	<code>_bleio, _pixelbuf, analogio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, ustack, vectorio</code>
Saola 1 w/Wroom	<code>_bleio, _pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, vectorio, watchdog, wifi</code>

continues on next page



Table 1 – continued from previous page

Board	Modules Available
Saola 1 w/Wrover	<code>_bleio, _pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, vectorio, watchdog, wifi</code>
See- duino Wio Terminal	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Seeeduino XIAO	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
Serpente	<code>_pixelbuf, analogio, audiocore, audioio, bitbangio, board, busio, countio, digitalio, displayio, errno, i2cperipheral, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi, vectorio</code>
Silicogni- tion LLC M4-Shim	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
Simmel	<code>_bleio, aesio, analogio, audiobusio, audiocore, audiomixer, audiopwmio, binascii, bitbangio, board, busio, digitalio, json, math, microcontroller, os, pulseio, pwmio, random, re, rtc, storage, struct, supervisor, time, usb_hid, watchdog</code>
SparkFun LU- MIDrive	<code>_pixelbuf, analogio, bitbangio, board, busio, countio, digitalio, displayio, errno, gamepad, i2cperipheral, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
SparkFun MicroMod ATP - RP2040	<i>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, bitops, board, busio, countio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</i>
SparkFun MicroMod nRF52840	<i>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</i>
SparkFun Pro Micro RP2040	<i>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, bitops, board, busio, countio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</i>
Spark- Fun Pro nRF52840 Mini	<i>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</i>
SparkFun Qwiic Micro	<i>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</i>
SparkFun RedBoard Turbo	<i>_pixelbuf, analogio, audiobusio, audiocore, audioio, board, busio, digitalio, displayio, errno, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi</i>
SparkFun SAMD21 Dev Breakout	<i>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</i>
SparkFun SAMD21 Mini Breakout	<i>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</i>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
SparkFun Thing Plus - RP2040	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, bitops, board, busio, countio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
SparkFun Thing Plus - SAMD51	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
SPRE-SENSE	<code>_bleio, _pixelbuf, analogio, binascii, bitbangio, board, busio, camera, digitalio, errno, gnss, json, math, microcontroller, os, pulseio, pwmio, random, re, rtc, sdcardio, sdioio, storage, struct, supervisor, time, ulab</code>
Sprite_v2b	<code>_pixelbuf, analogio, audiocore, audioio, audiomixer, binascii, bitbangio, board, busio, countio, digitalio, errno, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
ST STM32F746G Discovery	<code>_bleio, _pixelbuf, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, os, pulseio, pwmio, random, re, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, vectorio</code>
StackR-duino M0 PRO	<code>_pixelbuf, analogio, audiobusio, audiocore, audioio, board, busio, digitalio, displayio, errno, gamepad, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi</code>
stm32f411ce-blackpill	<code>_bleio, _pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, os, pulseio, pwmio, random, re, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, vectorio</code>
stm32f411ce-blackpill-with-flash	<code>_bleio, _pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, os, pulseio, pwmio, random, re, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, vectorio</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
STM32F411E	<code>_DISCO</code> , <code>_pixelbuf</code> , <code>analogio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>sdcario</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>vectorio</code>
STM32F412G	<code>_DISCO</code> , <code>_pixelbuf</code> , <code>analogio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>sdcario</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>vectorio</code>
STM32F412C	<code>_DISCO</code> , <code>_pixelbuf</code> , <code>analogio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>canio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>sdcario</code> , <code>sdioio</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>vectorio</code>
Targett Mod- ule Clip w/Wroom	<code>_bleio</code> , <code>_pixelbuf</code> , <code>alarm</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>canio</code> , <code>countio</code> , <code>digitalio</code> , <code>displayio</code> , <code>dualbank</code> , <code>errno</code> , <code>framebufferio</code> , <code>frequencyio</code> , <code>gamepad</code> , <code>ipaddress</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>ps2io</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcario</code> , <code>sharpdisplay</code> , <code>socketpool</code> , <code>ssl</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>vectorio</code> , <code>watchdog</code> , <code>wifi</code>
Targett Mod- ule Clip w/Wrover	<code>_bleio</code> , <code>_pixelbuf</code> , <code>alarm</code> , <code>analogio</code> , <code>audiobusio</code> , <code>audiocore</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>canio</code> , <code>countio</code> , <code>digitalio</code> , <code>displayio</code> , <code>dualbank</code> , <code>errno</code> , <code>framebufferio</code> , <code>frequencyio</code> , <code>gamepad</code> , <code>ipaddress</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>nvm</code> , <code>os</code> , <code>ps2io</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rotaryio</code> , <code>rtc</code> , <code>sdcario</code> , <code>sharpdisplay</code> , <code>socketpool</code> , <code>ssl</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>vectorio</code> , <code>watchdog</code> , <code>wifi</code>
Teensy 4.0	<code>_bleio</code> , <code>_pixelbuf</code> , <code>analogio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rtc</code> , <code>sdcario</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code>
Teensy 4.1	<code>_bleio</code> , <code>_pixelbuf</code> , <code>analogio</code> , <code>binascii</code> , <code>bitbangio</code> , <code>bitmaptools</code> , <code>board</code> , <code>busio</code> , <code>digitalio</code> , <code>displayio</code> , <code>errno</code> , <code>framebufferio</code> , <code>gamepad</code> , <code>json</code> , <code>math</code> , <code>microcontroller</code> , <code>msgpack</code> , <code>neopixel_write</code> , <code>os</code> , <code>pulseio</code> , <code>pwmio</code> , <code>random</code> , <code>re</code> , <code>rtc</code> , <code>sdcario</code> , <code>sharpdisplay</code> , <code>storage</code> , <code>struct</code> , <code>supervisor</code> , <code>terminalio</code> , <code>time</code> , <code>touchio</code> , <code>ulab</code> , <code>usb_hid</code> , <code>usb_midi</code> , <code>vectorio</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Teknikio Bluebird	<code>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
TG-Boards' Datalore IP M4	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
TG-Watch	<code>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
The Open Book Feather	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, gamepadshift, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
THUNDER-PACK_v11	<code>_pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, vectorio</code>
THUNDER-PACK_v12	<code>_pixelbuf, analogio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, vectorio</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
Tinker- ingTech Scout- Makes Azul	<code>_bleio, _pixelbuf, aesio, analogio, audiobusio, audiocore, audiomixer, audiomp3, audiopwmio, binascii, bitbangio, bitmaptools, board, busio, digitalio, displayio, errno, framebufferio, gamepad, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio, watchdog</code>
TinyS2	<code>_bleio, _pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, errno, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, vectorio, watchdog, wifi</code>
Trinket M0 Haxpress	<code>_pixelbuf, analogio, audiocore, audioio, board, busio, digitalio, displayio, errno, gamepad, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, storage, struct, supervisor, terminalio, time, touchio, usb_hid, usb_midi, vectorio</code>
UART- Logger II	<code>_bleio, _pixelbuf, analogio, audiobusio, audiocore, audioio, audiomixer, audiomp3, binascii, bitbangio, bitmaptools, board, busio, countio, digitalio, displayio, errno, framebufferio, frequencyio, gamepad, i2cperipheral, json, math, microcontroller, msgpack, neopixel_write, nvm, os, ps2io, pulseio, pwmio, random, re, rgbmatrix, rotaryio, rtc, sdcardio, sharpdisplay, storage, struct, supervisor, terminalio, time, touchio, ulab, usb_hid, usb_midi, vectorio</code>
uChip	<code>analogio, board, busio, digitalio, math, microcontroller, neopixel_write, nvm, os, pulseio, pwmio, random, rotaryio, rtc, storage, struct, supervisor, time, touchio, usb_hid, usb_midi</code>
uGame10	<code>_stage, analogio, audiocore, audioio, board, busio, countio, digitalio, displayio, errno, gamepad, math, microcontroller, msgpack, nvm, os, random, re, rotaryio, storage, struct, supervisor, terminalio, time, vectorio</code>
Win- terbloom Big Honk- ing Button	<code>_pixelbuf, analogio, audiocore, audioio, board, busio, countio, digitalio, errno, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, storage, struct, supervisor, time</code>
Win- terbloom Sol	<code>_pixelbuf, analogio, binascii, bitbangio, board, busio, countio, digitalio, errno, frequencyio, json, math, microcontroller, msgpack, neopixel_write, nvm, os, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, storage, struct, supervisor, time, ulab, usb_midi</code>
XinaBox CC03	<code>board, busio, digitalio, math, microcontroller, nvm, os, random, rtc, storage, struct, supervisor, time, usb_hid, usb_midi</code>

continues on next page

Table 1 – continued from previous page

Board	Modules Available
XinaBox CS11	<i>board, busio, digitalio, math, microcontroller, nvm, os, random, storage, struct, supervisor, time, usb_hid</i>

## Modules

### `_bleio` – Bluetooth Low Energy (BLE) communication

The `_bleio` module provides necessary low-level functionality for communicating using Bluetooth Low Energy (BLE). The ‘\_’ prefix indicates this module is meant for internal use by libraries but not by the end user. Its API may change incompatibly between minor versions of CircuitPython. Please use the `adafruit_ble` CircuitPython library instead, which builds on `_bleio`, and provides higher-level convenience functionality, including predefined beacons, clients, servers.

#### `_bleio.adapter :Adapter`

BLE Adapter used to manage device discovery and connections. This object is the sole instance of `_bleio.Adapter`.

#### **exception** `_bleio.BluetoothError`

Bases: `Exception`

Catchall exception for Bluetooth related errors.

Initialize self. See `help(type(self))` for accurate signature.

#### **exception** `_bleio.RoleError`

Bases: `BluetoothError`

Raised when a resource is used as the mismatched role. For example, if a local CCCD is attempted to be set but they can only be set when remote.

Initialize self. See `help(type(self))` for accurate signature.

#### **exception** `_bleio.SecurityError`

Bases: `BluetoothError`

Raised when a security related error occurs.

Initialize self. See `help(type(self))` for accurate signature.

#### `_bleio.set_adapter(adapter: Optional[Adapter]) → None`

Set the adapter to use for BLE, such as when using an HCI adapter. Raises `NotImplementedError` when the adapter is a singleton and cannot be set.

#### **class** `_bleio.Adapter` (\*, `uart: busio.UART`, `rts: digitalio.DigitalInOut`, `cts: digitalio.DigitalInOut`)

The BLE Adapter object manages the discovery and connection to other nearby Bluetooth Low Energy devices. This part of the Bluetooth Low Energy Specification is known as Generic Access Profile (GAP).

Discovery of other devices happens during a scanning process that listens for small packets of information, known as advertisements, that are broadcast unencrypted. The advertising packets have two different uses. The first is to broadcast a small piece of data to anyone who cares and nothing more. These are known as beacons. The second class of advertisement is to promote additional functionality available after the devices establish a connection. For example, a BLE heart rate monitor would advertise that it provides the standard BLE Heart Rate Service.

The Adapter can do both parts of this process: it can scan for other device advertisements and it can advertise its own data. Furthermore, Adapters can accept incoming connections and also initiate connections.



On boards that do not have native BLE, you can use an HCI co-processor. Pass the `uart` and pins used to communicate with the co-processor, such as an Adafruit AirLift. The co-processor must have been reset and put into BLE mode beforehand by the appropriate pin manipulation. The `uart`, `rts`, and `cts` objects are used to communicate with the HCI co-processor in HCI mode. The `Adapter` object is enabled during this call.

After instantiating an `Adapter`, call `_bleio.set_adapter()` to set `_bleio.adapter`

On boards with native BLE, you cannot create an instance of `_bleio.Adapter`; this constructor will raise `NotImplementedError`. Use `_bleio.adapter` to access the sole instance already available.

**enabled :bool**

State of the BLE adapter.

**address :Address**

MAC address of the BLE adapter.

**name :str**

name of the BLE adapter used once connected. The name is “CIRCUITPY” + the last four hex digits of `adapter.address`, to make it easy to distinguish multiple CircuitPython boards.

**advertising :bool**

True when the adapter is currently advertising. (read-only)

**connected :bool**

True when the adapter is connected to another device regardless of who initiated the connection. (read-only)

**connections :Tuple[Connection]**

Tuple of active connections including those initiated through `_bleio.Adapter.connect()`. (read-only)

**start\_advertising** (*self*, *data*: `_typing.ReadableBuffer`, \*, *scan\_response*: *Optional[\_typing.ReadableBuffer]* = *None*, *connectable*: *bool* = *True*, *anonymous*: *bool* = *False*, *timeout*: *int* = *0*, *interval*: *float* = *0.1*) → *None*

Starts advertising until `stop_advertising` is called or if `connectable`, another device connects to us.

#### Parameters

- **data** (`ReadableBuffer`) – advertising data packet bytes
- **scan\_response** (`ReadableBuffer`) – scan response data packet bytes. *None* if no scan response is needed.
- **connectable** (*bool*) – If *True* then other devices are allowed to connect to this peripheral.
- **anonymous** (*bool*) – If *True* then this device’s MAC address is randomized before advertising.
- **timeout** (*int*) – If set, we will only advertise for this many seconds. Zero means no timeout.
- **interval** (*float*) – advertising interval, in seconds

**stop\_advertising** (*self*) → *None*

Stop sending advertising packets.

**start\_scan** (*self*, *prefixes*: `_typing.ReadableBuffer` = *b''*, \*, *buffer\_size*: *int* = *512*, *extended*: *bool* = *False*, *timeout*: *Optional[float]* = *None*, *interval*: *float* = *0.1*, *window*: *float* = *0.1*, *minimum\_rssi*: *int* = *-80*, *active*: *bool* = *True*) → *Iterable[ScanEntry]*

Starts a BLE scan and returns an iterator of results. Advertisements and scan responses are filtered and returned separately.



**Parameters**

- **prefixes** ([ReadableBuffer](#)) – Sequence of byte string prefixes to filter advertising packets with. A packet without an advertising structure that matches one of the prefixes is ignored. Format is one byte for length (n) and n bytes of prefix and can be repeated.
- **buffer\_size** ([int](#)) – the maximum number of advertising bytes to buffer.
- **extended** ([bool](#)) – When True, support extended advertising packets. Increasing `buffer_size` is recommended when this is set.
- **timeout** ([float](#)) – the scan timeout in seconds. If None or zero, will scan until `stop_scan` is called.
- **interval** ([float](#)) – the interval (in seconds) between the start of two consecutive scan windows. Must be in the range 0.0025 - 40.959375 seconds.
- **window** ([float](#)) – the duration (in seconds) to scan a single BLE channel. window must be  $\leq$  interval.
- **minimum\_rssi** ([int](#)) – the minimum rssi of entries to return.
- **active** ([bool](#)) – retrieve scan responses for scannable advertisements.

**Returns** an iterable of `_bleio.ScanEntry` objects

**Return type** iterable

**stop\_scan** (*self*) → [None](#)

Stop the current scan.

**connect** (*self*, *address*: [Address](#), \*, *timeout*: [float](#)) → [Connection](#)

Attempts a connection to the device with the given address.

**Parameters**

- **address** ([Address](#)) – The address of the peripheral to connect to
- **timeout** ([float/int](#)) – Try to connect for timeout seconds.

**erase\_bonding** (*self*) → [None](#)

Erase all bonding information stored in flash memory.

**class** `_bleio.Address` (*address*: [\\_typing.ReadableBuffer](#), *address\_type*: [int](#))

Encapsulates the address of a BLE device.

Create a new Address object encapsulating the address value. The value itself can be one of:

**Parameters**

- **address** ([ReadableBuffer](#)) – The address value to encapsulate. A buffer object (bytearray, bytes) of 6 bytes.
- **address\_type** ([int](#)) – one of the integer values: `PUBLIC`, `RANDOM_STATIC`, `RANDOM_PRIVATE_RESOLVABLE`, or `RANDOM_PRIVATE_NON_RESOLVABLE`.

**address\_bytes** :[bytes](#)

The bytes that make up the device address (read-only).

Note that the `bytes` object returned is in little-endian order: The least significant byte is `address_bytes[0]`. So the address will appear to be reversed if you print the raw `bytes` object. If you print or use `str()` on the `Attribute` object itself, the address will be printed in the expected order. For example:

```
>>> import _bleio
>>> _bleio.adapter.address
<Address c8:1d:f5:ed:a8:35>
>>> _bleio.adapter.address.address_bytes
b'5\xa8\xed\xf5\x1d\xc8'
```

**type :int**

The address type (read-only).

One of the integer values: *PUBLIC*, *RANDOM\_STATIC*, *RANDOM\_PRIVATE\_RESOLVABLE*, or *RANDOM\_PRIVATE\_NON\_RESOLVABLE*.

**PUBLIC :int**

A publicly known address, with a company ID (high 24 bits) and company-assigned part (low 24 bits).

**RANDOM\_STATIC :int**

A randomly generated address that does not change often. It may never change or may change after a power cycle.

**RANDOM\_PRIVATE\_RESOLVABLE :int**

An address that is usable when the peer knows the other device's secret Identity Resolving Key (IRK).

**RANDOM\_PRIVATE\_NON\_RESOLVABLE :int**

A randomly generated address that changes on every connection.

**\_\_eq\_\_ (self, other: object) → bool**

Two Address objects are equal if their addresses and address types are equal.

**\_\_hash\_\_ (self) → int**

Returns a hash for the Address data.

**class \_bleio.Attribute**

Definitions associated with all BLE attributes: characteristics, descriptors, etc.

*Attribute* is, notionally, a superclass of *Characteristic* and *Descriptor*, but is not defined as a Python superclass of those classes.

You cannot create an instance of *Attribute*.

**NO\_ACCESS :int**

security\_mode: access not allowed

**OPEN :int**

security\_mode: no security (link is not encrypted)

**ENCRYPT\_NO\_MITM :int**

security\_mode: unauthenticated encryption, without man-in-the-middle protection

**ENCRYPT\_WITH\_MITM :int**

security\_mode: authenticated encryption, with man-in-the-middle protection

**LESC\_ENCRYPT\_WITH\_MITM :int**

security\_mode: LESC encryption, with man-in-the-middle protection

**SIGNED\_NO\_MITM :int**

security\_mode: unauthenticated data signing, without man-in-the-middle protection

**SIGNED\_WITH\_MITM :int**

security\_mode: authenticated data signing, without man-in-the-middle protection

**class \_bleio.Characteristic**

Stores information about a BLE service characteristic and allows reading and writing of the characteristic's value.

There is no regular constructor for a `Characteristic`. A new local `Characteristic` can be created and attached to a `Service` by calling `add_to_service()`. Remote `Characteristic` objects are created by `Connection.discover_remote_services()` as part of remote `Services`.

**properties** :int

An int bitmask representing which properties are set, specified as bitwise or'ing of of these possible values. `BROADCAST`, `INDICATE`, `NOTIFY`, `READ`, `WRITE`, `WRITE_NO_RESPONSE`.

**uuid** :Optional[UUID]

The UUID of this characteristic. (read-only)

Will be `None` if the 128-bit UUID for this characteristic is not known.

**value** :bytearray

The value of this characteristic.

**descriptors** :Descriptor

A tuple of `Descriptor` objects related to this characteristic. (read-only)

**service** :Service

The `Service` this `Characteristic` is a part of.

**BROADCAST** :int

property: allowed in advertising packets

**INDICATE** :int

property: server will indicate to the client when the value is set and wait for a response

**NOTIFY** :int

property: server will notify the client when the value is set

**READ** :int

property: clients may read this characteristic

**WRITE** :int

property: clients may write this characteristic; a response will be sent back

**WRITE\_NO\_RESPONSE** :int

property: clients may write this characteristic; no response will be sent back

**add\_to\_service** (self, service: `Service`, uuid: `UUID`, \*, properties: int = 0, read\_perm: int = `Attribute.OPEN`, write\_perm: int = `Attribute.OPEN`, max\_length: int = 20, fixed\_length: bool = False, initial\_value: Optional[\_typing.ReadableBuffer] = None) → `Characteristic`

Create a new `Characteristic` object, and add it to this `Service`.

#### Parameters

- **service** (`Service`) – The service that will provide this characteristic
- **uuid** (`UUID`) – The uuid of the characteristic
- **properties** (int) – The properties of the characteristic, specified as a bitmask of these values bitwise-or'd together: `BROADCAST`, `INDICATE`, `NOTIFY`, `READ`, `WRITE`, `WRITE_NO_RESPONSE`.
- **read\_perm** (int) – Specifies whether the characteristic can be read by a client, and if so, which security mode is required. Must be one of the integer values `Attribute.NO_ACCESS`, `Attribute.OPEN`, `Attribute.ENCRYPT_NO_MITM`, `Attribute.ENCRYPT_WITH_MITM`, `Attribute.LESC_ENCRYPT_WITH_MITM`, `Attribute.SIGNED_NO_MITM`, or `Attribute.SIGNED_WITH_MITM`.

- **write\_perm**(*int*) – Specifies whether the characteristic can be written by a client, and if so, which security mode is required. Values allowed are the same as `read_perm`.
- **max\_length**(*int*) – Maximum length in bytes of the characteristic value. The maximum allowed is 512, or possibly 510 if `fixed_length` is `False`. The default, 20, is the maximum number of data bytes that fit in a single BLE 4.x ATT packet.
- **fixed\_length**(*bool*) – True if the characteristic value is of fixed length.
- **initial\_value**(*ReadableBuffer*) – The initial value for this characteristic. If not given, will be filled with zeros.

**Returns** the new Characteristic.

**set\_cccd**(*self*, \*, *notify*: *bool* = *False*, *indicate*: *bool* = *False*) → *None*

Set the remote characteristic's CCCD to enable or disable notification and indication.

#### Parameters

- **notify**(*bool*) – True if Characteristic should receive notifications of remote writes
- **indicate**(*float*) – True if Characteristic should receive indications of remote writes

**class** `_bleio.CharacteristicBuffer`(*characteristic*: *Characteristic*, \*, *timeout*: *int* = 1, *buffer\_size*: *int* = 64)

Accumulates a Characteristic's incoming values in a FIFO buffer.

Monitor the given Characteristic. Each time a new value is written to the Characteristic add the newly-written bytes to a FIFO buffer.

#### Parameters

- **characteristic**(*Characteristic*) – The Characteristic to monitor. It may be a local Characteristic provided by a Peripheral Service, or a remote Characteristic in a remote Service that a Central has connected to.
- **timeout**(*int*) – the timeout in seconds to wait for the first character and between subsequent characters.
- **buffer\_size**(*int*) – Size of ring buffer that stores incoming data coming from client. Must be  $\geq 1$ .

**in\_waiting** :*int*

The number of bytes in the input buffer, available to be read

**read**(*self*, *nbytes*: *Optional*[*int*] = *None*) → *Optional*[*bytes*]

Read characters. If *nbytes* is specified then read at most that many bytes. Otherwise, read everything that arrives until the connection times out. Providing the number of bytes expected is highly recommended because it will be faster.

**Returns** Data read

**Return type** *bytes* or *None*

**readinto**(*self*, *buf*: *\_typing*.*WriteableBuffer*) → *Optional*[*int*]

Read bytes into the *buf*. Read at most `len(buf)` bytes.

**Returns** number of bytes read and stored into *buf*

**Return type** *int* or *None* (on a non-blocking error)

**readline**(*self*) → *bytes*

Read a line, ending in a newline character.

**Returns** the line read

Return type *int* or *None*

**reset\_input\_buffer** (*self*) → *None*

Discard any unread characters in the input buffer.

**deinit** (*self*) → *None*

Disable permanently.

**class** `_bleio.Connection`

A BLE connection to another device. Used to discover and interact with services on the other device.

Usage:

```
import _bleio

my_entry = None
for entry in _bleio.adapter.scan(2.5):
    if entry.name is not None and entry.name == 'InterestingPeripheral':
        my_entry = entry
        break

if not my_entry:
    raise Exception("'InterestingPeripheral' not found")

connection = _bleio.adapter.connect(my_entry.address, timeout=10)
```

Connections cannot be made directly. Instead, to initiate a connection use `Adapter.connect`. Connections may also be made when another device initiates a connection. To use a Connection created by a peer, read the `Adapter.connections` property.

**connected** :bool

True if connected to the remote peer.

**paired** :bool

True if paired to the remote peer.

**connection\_interval** :float

Time between transmissions in milliseconds. Will be multiple of 1.25ms. Lower numbers increase speed and decrease latency but increase power consumption.

When setting `connection_interval`, the peer may reject the new interval and `connection_interval` will then remain the same.

Apple has additional guidelines that dictate should be a multiple of 15ms except if HID is available. When HID is available Apple devices may accept 11.25ms intervals.

**attribute** :int

The maximum number of data bytes that can be sent in a single transmission, not including overhead bytes.

This is the maximum number of bytes that can be sent in a notification, which must be sent in a single packet. But for a regular characteristic read or write, may be sent in multiple packets, so this limit does not apply.

**disconnect** (*self*) → *None*

Disconnects from the remote peripheral. Does nothing if already disconnected.

**pair** (*self*, \*, *bond*: bool = True) → *None*

Pair to the peer to improve security.

**discover\_remote\_services** (*self*, *service\_uuids\_whitelist*: Optional[Iterable[UUID]] = None) → Tuple[Service, ...]

Do BLE discovery for all services or for the given service UUIDS, to find their handles and characteristics,

and return the discovered services. `Connection.connected` must be True.

**Parameters** `service_uuids_whitelist` (*iterable*) – an iterable of `UUID` objects for the services provided by the peripheral that you want to use.

The peripheral may provide more services, but services not listed are ignored and will not be returned.

If `service_uuids_whitelist` is None, then all services will undergo discovery, which can be slow.

If the service UUID is 128-bit, or its characteristic UUID's are 128-bit, you must have already created a `UUID` object for that UUID in order for the service or characteristic to be discovered. Creating the UUID causes the UUID to be registered for use. (This restriction may be lifted in the future.)

**Returns** A tuple of `_bleio.Service` objects provided by the remote peripheral.

**class** `_bleio.Descriptor`

Stores information about a BLE descriptor.

Descriptors are attached to BLE characteristics and provide contextual information about the characteristic.

There is no regular constructor for a Descriptor. A new local Descriptor can be created and attached to a Characteristic by calling `add_to_characteristic()`. Remote Descriptor objects are created by `Connection.discover_remote_services()` as part of remote Characteristics in the remote Services that are discovered.

**uuid** : `UUID`

The descriptor uuid. (read-only)

**characteristic** : `Characteristic`

The Characteristic this Descriptor is a part of.

**value** : `bytearray`

The value of this descriptor.

**classmethod** `add_to_characteristic` (*cls*, *characteristic*: `Characteristic`, *uuid*: `UUID`, \*, *read\_perm*: `int` = `Attribute.OPEN`, *write\_perm*: `int` = `Attribute.OPEN`, *max\_length*: `int` = 20, *fixed\_length*: `bool` = `False`, *initial\_value*: `_typing.ReadableBuffer` = `b''`) → `Descriptor`

Create a new Descriptor object, and add it to this Service.

**Parameters**

- **characteristic** (`Characteristic`) – The characteristic that will hold this descriptor
- **uuid** (`UUID`) – The uuid of the descriptor
- **read\_perm** (`int`) – Specifies whether the descriptor can be read by a client, and if so, which security mode is required. Must be one of the integer values `Attribute.NO_ACCESS`, `Attribute.OPEN`, `Attribute.ENCRYPT_NO_MITM`, `Attribute.ENCRYPT_WITH_MITM`, `Attribute.LESC_ENCRYPT_WITH_MITM`, `Attribute.SIGNED_NO_MITM`, or `Attribute.SIGNED_WITH_MITM`.
- **write\_perm** (`int`) – Specifies whether the descriptor can be written by a client, and if so, which security mode is required. Values allowed are the same as `read_perm`.

- **max\_length** (*int*) – Maximum length in bytes of the descriptor value. The maximum allowed is 512, or possibly 510 if `fixed_length` is `False`. The default, 20, is the maximum number of data bytes that fit in a single BLE 4.x ATT packet.
- **fixed\_length** (*bool*) – True if the descriptor value is of fixed length.
- **initial\_value** (*ReadableBuffer*) – The initial value for this descriptor.

**Returns** the new Descriptor.

**class** `_bleio.PacketBuffer` (*characteristic: Characteristic*, \*, *buffer\_size: int*)

Accumulates a Characteristic's incoming packets in a FIFO buffer and facilitates packet aware outgoing writes. A packet's size is either the characteristic length or the maximum transmission unit (MTU) minus overhead, whichever is smaller. The MTU can change so check `incoming_packet_length` and `outgoing_packet_length` before creating a buffer to store data.

When we're the server, we ignore all connections besides the first to subscribe to notifications.

Monitor the given Characteristic. Each time a new value is written to the Characteristic add the newly-written bytes to a FIFO buffer.

Monitor the given Characteristic. Each time a new value is written to the Characteristic add the newly-written packet of bytes to a FIFO buffer.

#### Parameters

- **characteristic** (*Characteristic*) – The Characteristic to monitor. It may be a local Characteristic provided by a Peripheral Service, or a remote Characteristic in a remote Service that a Central has connected to.
- **buffer\_size** (*int*) – Size of ring buffer (in packets of the Characteristic's maximum length) that stores incoming packets coming from the peer.

**packet\_size** :*int*

`packet_size` is the same as `incoming_packet_length`. The name `packet_size` is deprecated and will be removed in CircuitPython 6.0.0.

**incoming\_packet\_length** :*int*

Maximum length in bytes of a packet we are reading.

**outgoing\_packet\_length** :*int*

Maximum length in bytes of a packet we are writing.

**readinto** (*self*, *buf*: *\_typing.WriteableBuffer*) → *int*

Reads a single BLE packet into the `buf`. Raises an exception if the next packet is longer than the given buffer. Use `packet_size` to read the maximum length of a single packet.

**Returns** number of bytes read and stored into `buf`

**Return type** *int*

**write** (*self*, *data*: *\_typing.ReadableBuffer*, \*, *header: Optional[bytes] = None*) → *int*

Writes all bytes from `data` into the same outgoing packet. The bytes from `header` are included before `data` when the pending packet is currently empty.

This does not block until the data is sent. It only blocks until the data is pending.

**Returns** number of bytes written. May include header bytes when packet is empty.

**Return type** *int*

**deinit** (*self*) → *None*

Disable permanently.

**class** `_bleio.ScanEntry`

Encapsulates information about a device that was received during scanning. It can be advertisement or scan response data. This object may only be created by a `_bleio.ScanResults`: it has no user-visible constructor.

Cannot be instantiated directly. Use `_bleio.Adapter.start_scan`.

**address** :`Address`

The address of the device (read-only), of type `_bleio.Address`.

**advertisement\_bytes** :`bytes`

All the advertisement data present in the packet, returned as a `bytes` object. (read-only)

**rsssi** :`int`

The signal strength of the device at the time of the scan, in integer dBm. (read-only)

**connectable** :`bool`

True if the device can be connected to. (read-only)

**scan\_response** :`bool`

True if the entry was a scan response. (read-only)

**matches** (*self*, *prefixes*: `ScanEntry`, \*, *all*: `bool = True`) → `bool`

Returns True if the `ScanEntry` matches all prefixes when `all` is True. This is stricter than the scan filtering which accepts any advertisements that match any of the prefixes where `all` is False.

**class** `_bleio.ScanResults`

Iterates over advertising data received while scanning. This object is always created by a `_bleio.Adapter`: it has no user-visible constructor.

Cannot be instantiated directly. Use `_bleio.Adapter.start_scan`.

**\_\_iter\_\_** (*self*) → `Iterator[ScanEntry]`

Returns itself since it is the iterator.

**\_\_next\_\_** (*self*) → `ScanEntry`

Returns the next `_bleio.ScanEntry`. Blocks if none have been received and scanning is still active. Raises `StopIteration` if scanning is finished and no other results are available.

**class** `_bleio.Service` (*uuid*: `UUID`, \*, *secondary*: `bool = False`)

Stores information about a BLE service and its characteristics.

Create a new Service identified by the specified UUID. It can be accessed by all connections. This is known as a Service server. Client Service objects are created via `Connection.discover_remote_services`.

To mark the Service as secondary, pass `True` as `secondary`.

**Parameters**

- **uuid** (`UUID`) – The uuid of the service
- **secondary** (`bool`) – If the service is a secondary one

**Returns** the new Service

**characteristics** :`Tuple[Characteristic, ...]`

A tuple of `Characteristic` designating the characteristics that are offered by this service. (read-only)

**remote** :`bool`

True if this is a service provided by a remote device. (read-only)

**secondary** :`bool`

True if this is a secondary service. (read-only)



**uuid** :Optional[UUID]

The UUID of this service. (read-only)

Will be None if the 128-bit UUID for this service is not known.

**class** `_bleio.UUID` (*value: Union[int, \_typing.ReadableBuffer, str]*)

A 16-bit or 128-bit UUID. Can be used for services, characteristics, descriptors and more.

Create a new UUID or UUID object encapsulating the uuid value. The value can be one of:

- an *int* value in range 0 to 0xFFFF (Bluetooth SIG 16-bit UUID)
- a buffer object (bytearray, bytes) of 16 bytes in little-endian order (128-bit UUID)
- a string of hex digits of the form 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'

Creating a 128-bit UUID registers the UUID with the onboard BLE software, and provides a temporary 16-bit UUID that can be used in place of the full 128-bit UUID.

**Parameters** *value* (*int*, *ReadableBuffer* or *str*) – The uuid value to encapsulate

**uuid16** :int

The 16-bit part of the UUID. (read-only)

**Type** *int*

**uuid128** :bytes

The 128-bit value of the UUID Raises AttributeError if this is a 16-bit UUID. (read-only)

**Type** *bytes*

**size** :int

128 if this UUID represents a 128-bit vendor-specific UUID. 16 if this UUID represents a 16-bit Bluetooth SIG assigned UUID. (read-only) 32-bit UUIDs are not currently supported.

**Type** *int*

**pack\_into** (*self*, *buffer: \_typing.WritableBuffer*, *offset: int = 0*) → None

Packs the UUID into the given buffer at the given offset.

**\_\_eq\_\_** (*self*, *other: object*) → bool

Two UUID objects are equal if their values match and they are both 128-bit or both 16-bit.

## **`_eve` – Low-level BridgeTek EVE bindings**

The `_eve` module provides a class `_EVE` which contains methods for constructing EVE command buffers and appending basic graphics commands.

**class** `_eve._EVE`

**register** (*self*, *o: object*) → None

**flush** (*self*) → None

Send any queued drawing commands directly to the hardware.

**Parameters** *width* (*int*) – The width of the grid in tiles, or 1 for sprites.

**cc** (*self*, *b: \_typing.ReadableBuffer*) → None

Append bytes to the command FIFO.

**Parameters** *b* (*ReadableBuffer*) – The bytes to add

**AlphaFunc** (*self*, *func*: int, *ref*: int) → None

Set the alpha test function

**Parameters**

- **func** (int) – specifies the test function, one of NEVER, LESS, LEQUAL, GREATER, GEQUAL, EQUAL, NOTEQUAL, or ALWAYS. Range 0-7. The initial value is ALWAYS(7)
- **ref** (int) – specifies the reference value for the alpha test. Range 0-255. The initial value is 0

These values are part of the graphics context and are saved and restored by *SaveContext()* and *RestoreContext()*.

**Begin** (*self*, *prim*: int) → None

Begin drawing a graphics primitive

**Parameters** **prim** (int) – graphics primitive.

Valid primitives are BITMAPS, POINTS, LINES, LINE\_STRIP, EDGE\_STRIP\_R, EDGE\_STRIP\_L, EDGE\_STRIP\_A, EDGE\_STRIP\_B and RECTS.

**BitmapExtFormat** (*self*, *format*: int) → None

Set the bitmap format

**Parameters** **format** (int) – bitmap pixel format.

**BitmapHandle** (*self*, *handle*: int) → None

Set the bitmap handle

**Parameters** **handle** (int) – bitmap handle. Range 0-31. The initial value is 0

This value is part of the graphics context and is saved and restored by *SaveContext()* and *RestoreContext()*.

**BitmapLayoutH** (*self*, *linestride*: int, *height*: int) → None

Set the source bitmap memory format and layout for the current handle. high bits for large bitmaps

**Parameters**

- **linestride** (int) – high part of bitmap line stride, in bytes. Range 0-7
- **height** (int) – high part of bitmap height, in lines. Range 0-3

**BitmapLayout** (*self*, *format*: int, *linestride*: int, *height*: int) → None

Set the source bitmap memory format and layout for the current handle

**Parameters**

- **format** (int) – bitmap pixel format, or GLFORMAT to use BITMAP\_EXT\_FORMAT instead. Range 0-31
- **linestride** (int) – bitmap line stride, in bytes. Range 0-1023
- **height** (int) – bitmap height, in lines. Range 0-511

**BitmapSizeH** (*self*, *width*: int, *height*: int) → None

Set the screen drawing of bitmaps for the current handle. high bits for large bitmaps

**Parameters**

- **width** (int) – high part of drawn bitmap width, in pixels. Range 0-3
- **height** (int) – high part of drawn bitmap height, in pixels. Range 0-3

**BitmapSize** (*self*, *filter*: int, *wrapx*: int, *wrapy*: int, *width*: int, *height*: int) → None

Set the screen drawing of bitmaps for the current handle

**Parameters**

- **filter** (*int*) – bitmap filtering mode, one of NEAREST or BILINEAR. Range 0-1
- **wrapx** (*int*) – bitmap *x* wrap mode, one of REPEAT or BORDER. Range 0-1
- **wrapy** (*int*) – bitmap *y* wrap mode, one of REPEAT or BORDER. Range 0-1
- **width** (*int*) – drawn bitmap width, in pixels. Range 0-511
- **height** (*int*) – drawn bitmap height, in pixels. Range 0-511

**BitmapSource** (*self*, *addr*: *int*) → *None*

Set the source address for bitmap graphics

**Parameters** **addr** (*int*) – Bitmap start address, pixel-aligned. May be in SRAM or flash. Range 0-16777215

**BitmapSwizzle** (*self*, *r*: *int*, *g*: *int*, *b*: *int*, *a*: *int*) → *None*

Set the source for the r,g,b and a channels of a bitmap

**Parameters**

- **r** (*int*) – red component source channel. Range 0-7
- **g** (*int*) – green component source channel. Range 0-7
- **b** (*int*) – blue component source channel. Range 0-7
- **a** (*int*) – alpha component source channel. Range 0-7

**BitmapTransformA** (*self*, *p*: *int*, *v*: *int*) → *None*

Set the *a* component of the bitmap transform matrix

**Parameters**

- **p** (*int*) – precision control: 0 is 8.8, 1 is 1.15. Range 0-1. The initial value is 0
- **v** (*int*) – The *a* component of the bitmap transform matrix, in signed 8.8 or 1.15 bit fixed-point form. Range 0-131071. The initial value is 256

The initial value is **p** = 0, **v** = 256. This represents the value 1.0.

These values are part of the graphics context and are saved and restored by `SaveContext()` and `RestoreContext()`.

**BitmapTransformB** (*self*, *p*: *int*, *v*: *int*) → *None*

Set the *b* component of the bitmap transform matrix

**Parameters**

- **p** (*int*) – precision control: 0 is 8.8, 1 is 1.15. Range 0-1. The initial value is 0
- **v** (*int*) – The *b* component of the bitmap transform matrix, in signed 8.8 or 1.15 bit fixed-point form. Range 0-131071. The initial value is 0

The initial value is **p** = 0, **v** = 0. This represents the value 0.0.

These values are part of the graphics context and are saved and restored by `SaveContext()` and `RestoreContext()`.

**BitmapTransformC** (*self*, *v*: *int*) → *None*

Set the *c* component of the bitmap transform matrix

**Parameters** **v** (*int*) – The *c* component of the bitmap transform matrix, in signed 15.8 bit fixed-point form. Range 0-16777215. The initial value is 0

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**BitmapTransformD**(*self*, *p*: int, *v*: int) → None

Set the *d* component of the bitmap transform matrix

**Parameters**

- **p** (int) – precision control: 0 is 8.8, 1 is 1.15. Range 0-1. The initial value is 0
- **v** (int) – The *d* component of the bitmap transform matrix, in signed 8.8 or 1.15 bit fixed-point form. Range 0-131071. The initial value is 0

The initial value is **p** = 0, **v** = 0. This represents the value 0.0.

These values are part of the graphics context and are saved and restored by `SaveContext()` and `RestoreContext()`.

**BitmapTransformE**(*self*, *p*: int, *v*: int) → None

Set the *e* component of the bitmap transform matrix

**Parameters**

- **p** (int) – precision control: 0 is 8.8, 1 is 1.15. Range 0-1. The initial value is 0
- **v** (int) – The *e* component of the bitmap transform matrix, in signed 8.8 or 1.15 bit fixed-point form. Range 0-131071. The initial value is 256

The initial value is **p** = 0, **v** = 256. This represents the value 1.0.

These values are part of the graphics context and are saved and restored by `SaveContext()` and `RestoreContext()`.

**BitmapTransformF**(*self*, *v*: int) → None

Set the *f* component of the bitmap transform matrix

**Parameters** **v** (int) – The *f* component of the bitmap transform matrix, in signed 15.8 bit fixed-point form. Range 0-16777215. The initial value is 0

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**BlendFunc**(*self*, *src*: int, *dst*: int) → None

Set pixel arithmetic

**Parameters**

- **src** (int) – specifies how the source blending factor is computed. One of ZERO, ONE, SRC\_ALPHA, DST\_ALPHA, ONE\_MINUS\_SRC\_ALPHA or ONE\_MINUS\_DST\_ALPHA. Range 0-7. The initial value is SRC\_ALPHA(2)
- **dst** (int) – specifies how the destination blending factor is computed, one of the same constants as **src**. Range 0-7. The initial value is ONE\_MINUS\_SRC\_ALPHA(4)

These values are part of the graphics context and are saved and restored by `SaveContext()` and `RestoreContext()`.

**Call**(*self*, *dest*: int) → None

Execute a sequence of commands at another location in the display list

**Parameters** **dest** (int) – display list address. Range 0-65535

**Cell**(*self*, *cell*: int) → None

Set the bitmap cell number for the vertex2f command

**Parameters** **cell** (int) – bitmap cell number. Range 0-127. The initial value is 0

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**ClearColorA**(*self*, *alpha*: int) → None

Set clear value for the alpha channel

**Parameters** **alpha** (int) – alpha value used when the color buffer is cleared. Range 0-255. The initial value is 0

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**ClearColorRGB**(*self*, *red*: int, *green*: int, *blue*: int) → None

Set clear values for red, green and blue channels

#### Parameters

- **red** (int) – red value used when the color buffer is cleared. Range 0-255. The initial value is 0
- **green** (int) – green value used when the color buffer is cleared. Range 0-255. The initial value is 0
- **blue** (int) – blue value used when the color buffer is cleared. Range 0-255. The initial value is 0

These values are part of the graphics context and are saved and restored by `SaveContext()` and `RestoreContext()`.

**Clear**(*self*, *c*: int, *s*: int, *t*: int) → None

Clear buffers to preset values

#### Parameters

- **c** (int) – clear color buffer. Range 0-1
- **s** (int) – clear stencil buffer. Range 0-1
- **t** (int) – clear tag buffer. Range 0-1

**ClearStencil**(*self*, *s*: int) → None

Set clear value for the stencil buffer

**Parameters** **s** (int) – value used when the stencil buffer is cleared. Range 0-255. The initial value is 0

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**ClearTag**(*self*, *s*: int) → None

Set clear value for the tag buffer

**Parameters** **s** (int) – value used when the tag buffer is cleared. Range 0-255. The initial value is 0

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**ColorA**(*self*, *alpha*: int) → None

Set the current color alpha

**Parameters** **alpha** (int) – alpha for the current color. Range 0-255. The initial value is 255

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**ColorMask** (*self*, *r*: int, *g*: int, *b*: int, *a*: int) → None

Enable and disable writing of frame buffer color components

**Parameters**

- **r** (int) – allow updates to the frame buffer red component. Range 0-1. The initial value is 1
- **g** (int) – allow updates to the frame buffer green component. Range 0-1. The initial value is 1
- **b** (int) – allow updates to the frame buffer blue component. Range 0-1. The initial value is 1
- **a** (int) – allow updates to the frame buffer alpha component. Range 0-1. The initial value is 1

These values are part of the graphics context and are saved and restored by `SaveContext()` and `RestoreContext()`.

**ColorRGB** (*self*, *red*: int, *green*: int, *blue*: int) → None

Set the drawing color

**Parameters**

- **red** (int) – red value for the current color. Range 0-255. The initial value is 255
- **green** (int) – green for the current color. Range 0-255. The initial value is 255
- **blue** (int) – blue for the current color. Range 0-255. The initial value is 255

These values are part of the graphics context and are saved and restored by `SaveContext()` and `RestoreContext()`.

**Display** (*self*) → None

End the display list

**End** (*self*) → None

End drawing a graphics primitive

`Vertex2ii()` and `Vertex2f()` calls are ignored until the next `Begin()`.

**Jump** (*self*, *dest*: int) → None

Execute commands at another location in the display list

**Parameters** **dest** (int) – display list address. Range 0-65535

**Macro** (*self*, *m*: int) → None

Execute a single command from a macro register

**Parameters** **m** (int) – macro register to read. Range 0-1

**Nop** (*self*) → None

No operation

**PaletteSource** (*self*, *addr*: int) → None

Set the base address of the palette

**Parameters** **addr** (int) – Address in graphics SRAM, 2-byte aligned. Range 0-4194303. The initial value is 0

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**RestoreContext** (*self*) → None

Restore the current graphics context from the context stack

**Return** (*self*) → *None*

Return from a previous call command

**SaveContext** (*self*) → *None*

Push the current graphics context on the context stack

**ScissorSize** (*self*, *width*: *int*, *height*: *int*) → *None*

Set the size of the scissor clip rectangle

#### Parameters

- **width** (*int*) – The width of the scissor clip rectangle, in pixels. Range 0-4095. The initial value is *hsz*
- **height** (*int*) – The height of the scissor clip rectangle, in pixels. Range 0-4095. The initial value is 2048

These values are part of the graphics context and are saved and restored by *SaveContext()* and *RestoreContext()*.

**ScissorXY** (*self*, *x*: *int*, *y*: *int*) → *None*

Set the top left corner of the scissor clip rectangle

#### Parameters

- **x** (*int*) – The *x* coordinate of the scissor clip rectangle, in pixels. Range 0-2047. The initial value is 0
- **y** (*int*) – The *y* coordinate of the scissor clip rectangle, in pixels. Range 0-2047. The initial value is 0

These values are part of the graphics context and are saved and restored by *SaveContext()* and *RestoreContext()*.

**StencilFunc** (*self*, *func*: *int*, *ref*: *int*, *mask*: *int*) → *None*

Set function and reference value for stencil testing

#### Parameters

- **func** (*int*) – specifies the test function, one of NEVER, LESS, LEQUAL, GREATER, GEQUAL, EQUAL, NOTEQUAL, or ALWAYS. Range 0-7. The initial value is ALWAYS(7)
- **ref** (*int*) – specifies the reference value for the stencil test. Range 0-255. The initial value is 0
- **mask** (*int*) – specifies a mask that is ANDed with the reference value and the stored stencil value. Range 0-255. The initial value is 255

These values are part of the graphics context and are saved and restored by *SaveContext()* and *RestoreContext()*.

**StencilMask** (*self*, *mask*: *int*) → *None*

Control the writing of individual bits in the stencil planes

**Parameters** **mask** (*int*) – the mask used to enable writing stencil bits. Range 0-255. The initial value is 255

This value is part of the graphics context and is saved and restored by *SaveContext()* and *RestoreContext()*.

**StencilOp** (*self*, *sfail*: *int*, *spass*: *int*) → *None*

Set stencil test actions

#### Parameters

- **sfail** (*int*) – specifies the action to take when the stencil test fails, one of `KEEP`, `ZERO`, `REPLACE`, `INCR`, `INCR_WRAP`, `DECR`, `DECR_WRAP`, and `INVERT`. Range 0-7. The initial value is `KEEP(1)`
- **spass** (*int*) – specifies the action to take when the stencil test passes, one of the same constants as **sfail**. Range 0-7. The initial value is `KEEP(1)`

These values are part of the graphics context and are saved and restored by `SaveContext()` and `RestoreContext()`.

**TagMask** (*self*, *mask*: *int*) → *None*  
Control the writing of the tag buffer

**Parameters** **mask** (*int*) – allow updates to the tag buffer. Range 0-1. The initial value is 1

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**Tag** (*self*, *s*: *int*) → *None*  
Set the current tag value

**Parameters** **s** (*int*) – tag value. Range 0-255. The initial value is 255

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**Vertex2ii** (*self*, *x*: *int*, *y*: *int*, *handle*: *int*, *cell*: *int*) → *None*

**Parameters**

- **x** (*int*) – x-coordinate in pixels. Range 0-511
- **y** (*int*) – y-coordinate in pixels. Range 0-511
- **handle** (*int*) – bitmap handle. Range 0-31
- **cell** (*int*) – cell number. Range 0-127

This method is an alternative to `Vertex2f()`.

**Vertex2f** (*self*, *b*: *float*) → *None*  
Draw a point.

**Parameters**

- **x** (*float*) – pixel x-coordinate
- **y** (*float*) – pixel y-coordinate

**LineWidth** (*self*, *width*: *float*) → *None*  
Set the width of rasterized lines

**Parameters** **width** (*float*) – line width in pixels. Range 0-511. The initial value is 1

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**PointSize** (*self*, *size*: *float*) → *None*  
Set the diameter of rasterized points

**Parameters** **size** (*float*) – point diameter in pixels. Range 0-1023. The initial value is 1

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**VertexTranslateX** (*self*, *x*: *float*) → *None*  
Set the vertex transformation's x translation component



**Parameters** *x* (*float*) – signed x-coordinate in pixels. Range  $\pm 4095$ . The initial value is 0

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**VertexTranslateY** (*self*, *y*: *float*) → *None*

Set the vertex transformation's y translation component

**Parameters** *y* (*float*) – signed y-coordinate in pixels. Range  $\pm 4095$ . The initial value is 0

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**VertexFormat** (*self*, *frac*: *int*) → *None*

Set the precision of vertex2f coordinates

**Parameters** *frac* (*int*) – Number of fractional bits in X,Y coordinates, 0-4. Range 0-7. The initial value is 4

This value is part of the graphics context and is saved and restored by `SaveContext()` and `RestoreContext()`.

**cmd0** (*self*, *n*: *int*) → *None*

Append the command word *n* to the FIFO

**Parameters** *n* (*int*) – The command code

This method is used by the `eve` module to efficiently add commands to the FIFO.

**cmd** (*self*, *n*: *int*, *fmt*: *str*, *args*: *Tuple[str, ...]*) → *None*

Append a command packet to the FIFO.

**Parameters**

- *n* (*int*) – The command code
- *fmt* (*str*) – The command format `struct` layout
- *args* (*tuple* (*str*, ...)) – The command's arguments

Supported format codes: h, H, i, I.

This method is used by the `eve` module to efficiently add commands to the FIFO.

## `_pew` – LED matrix driver

**class** `_pew.PewPew` (*buffer*: `_typing.ReadableBuffer`, *rows*: `List[digitalio.DigitalInOut]`, *cols*: `List[digitalio.DigitalInOut]`, *buttons*: `digitalio.DigitalInOut`)

This is an internal module to be used by the `pew.py` library from <https://github.com/pewpew-game/pew-pewpew-standalone-10.x> to handle the LED matrix display and buttons on the `pewpew10` board.

Usage:

This singleton class is instantiated by the ```pew``` library, and used internally by it. All user-visible interactions are done through that library.

Initializes matrix scanning routines.

The `buffer` is a 64 byte long `bytearray` that stores what should be displayed on the matrix. `rows` and `cols` are both lists of eight `DigitalInputOutput` objects that are connected to the matrix rows and columns. `buttons` is a `DigitalInputOutput` object that is connected to the common side of all buttons (the other sides of the buttons are connected to rows of the matrix).

## `_pixelbuf` – A fast RGB(W) pixel buffer library for like NeoPixel and DotStar

The `_pixelbuf` module provides the `PixelBuf` class to accelerate RGB(W) strip/matrix manipulation, such as DotStar and NeoPixel.

Byteorders are configured with strings, such as “RGB” or “RGBD”.

`_pixelbuf.colorwheel(n: float) → int`

C implementation of the common `wheel()` function found in many examples. Returns the colorwheel RGB value as an integer value for `n` (usable in `PixelBuf`, `neopixel`, and `dotstar`).

`_pixelbuf.wheel(n: float) → int`

Use of `wheel()` is deprecated. Please use `colorwheel()`.

**class** `_pixelbuf.PixelBuf` (`size: int, *, byteorder: str = 'BGR', brightness: float = 0, auto_write: bool = False, header: typing.ReadableBuffer = b'', trailer: typing.ReadableBuffer = b''`)

A fast RGB[W] pixel buffer for LED and similar devices.

Create a `PixelBuf` object of the specified size, byteorder, and bits per pixel.

When brightness is less than 1.0, a second buffer will be used to store the color values before they are adjusted for brightness.

When `P` (PWM duration) is present as the 4th character of the byteorder string, the 4th value in the tuple/list for a pixel is the individual pixel brightness (0.0-1.0) and will enable a Dotstar compatible 1st byte for each pixel.

### Parameters

- **size** (`int`) – Number of pixels
- **byteorder** (`str`) – Byte order string (such as “RGB”, “RGBW” or “PBGR”)
- **brightness** (`float`) – Brightness (0 to 1.0, default 1.0)
- **auto\_write** (`bool`) – Whether to automatically write pixels (Default False)
- **header** (`ReadableBuffer`) – Sequence of bytes to always send before pixel values.
- **trailer** (`ReadableBuffer`) – Sequence of bytes to always send after pixel values.

**bpp** : `int`

The number of bytes per pixel in the buffer (read-only)

**brightness** : `float`

Float value between 0 and 1. Output brightness.

When brightness is less than 1.0, a second buffer will be used to store the color values before they are adjusted for brightness.

**auto\_write** : `bool`

Whether to automatically write the pixels after each update.

**byteorder** : `str`

byteorder string for the buffer (read-only)

**show** (`self`) → `None`

Transmits the color data to the pixels so that they are shown. This is done automatically when `auto_write` is True.

**fill** (`self, color: Union[int, Tuple[int, int, int], Tuple[int, int, int, float]]`) → `None`

Fills the given `pixelbuf` with the given color.

**\_\_getitem\_\_** (`self, index: slice`) → `Union[Tuple[Tuple[int, int, int], ...], Tuple[Tuple[int, int, int, float], ...]]`

`__getitem__` (*self*, *index*: `int`) → `Union[Tuple[int, int, int], Tuple[int, int, int, float]]`  
 Returns the pixel value at the given index as a tuple of (Red, Green, Blue[, White]) values between 0 and 255. When in PWM (DotStar) mode, the 4th tuple value is a float of the pixel intensity from 0-1.0.

`__setitem__` (*self*, *index*: `slice`, *value*: `Tuple[Union[int, Tuple[float, ...], List[float]], ...]`) → `None`  
`__setitem__` (*self*, *index*: `slice`, *value*: `List[Union[int, Tuple[float, ...], List[float]]]`) → `None`  
`__setitem__` (*self*, *index*: `int`, *value*: `Union[int, Tuple[float, ...], List[float]]`) → `None`  
 Sets the pixel value at the given index. Value can either be a tuple or integer. Tuples are The individual (Red, Green, Blue[, White]) values between 0 and 255. If given an integer, the red, green and blue values are packed into the lower three bytes (0xRRGGBB). For RGBW byteorders, if given only RGB values either as an int or as a tuple, the white value is used instead when the red, green, and blue values are the same.

## `_stage` – C-level helpers for animation of sprites on a stage

The `_stage` module contains native code to speed-up the `stage` Library <<https://github.com/python-ugame/circuitpython-stage>>`.

`_stage.render` (*x0*: `int`, *y0*: `int`, *x1*: `int`, *y1*: `int`, *layers*: `List[Layer]`, *buffer*: `_typing.WriteableBuffer`, *display*: `displayio.Display`, *scale*: `int`, *background*: `int`) → `None`  
 Render and send to the display a fragment of the screen.

### Parameters

- ***x0*** (`int`) – Left edge of the fragment.
- ***y0*** (`int`) – Top edge of the fragment.
- ***x1*** (`int`) – Right edge of the fragment.
- ***y1*** (`int`) – Bottom edge of the fragment.
- ***layers*** (`list [Layer]`) – A list of the `Layer` objects.
- ***buffer*** (`WriteableBuffer`) – A buffer to use for rendering.
- ***display*** (`Display`) – The display to use.
- ***scale*** (`int`) – How many times should the image be scaled up.
- ***background*** (`int`) – What color to display when nothing is there.

There are also no sanity checks, outside of the basic overflow checking. The caller is responsible for making the passed parameters valid.

This function is intended for internal use in the `stage` library and all the necessary checks are performed there.

**class** `_stage.Layer` (*width*: `int`, *height*: `int`, *graphic*: `_typing.ReadableBuffer`, *palette*: `_typing.ReadableBuffer`, *grid*: `_typing.ReadableBuffer`)

Keep information about a single layer of graphics

Keep internal information about a layer of graphics (either a `Grid` or a `Sprite`) in a format suitable for fast rendering with the `render()` function.

### Parameters

- ***width*** (`int`) – The width of the grid in tiles, or 1 for sprites.
- ***height*** (`int`) – The height of the grid in tiles, or 1 for sprites.
- ***graphic*** (`ReadableBuffer`) – The graphic data of the tiles.
- ***palette*** (`ReadableBuffer`) – The color palette to be used.

- **grid** (*ReadableBuffer*) – The contents of the grid map.

This class is intended for internal use in the `stage` library and it shouldn't be used on its own.

**move** (*self*, *x*: *int*, *y*: *int*) → *None*

Set the offset of the layer to the specified values.

**frame** (*self*, *frame*: *int*, *rotation*: *int*) → *None*

Set the animation frame of the sprite, and optionally rotation its graphic.

**class** `_stage.Text` (*width*: *int*, *height*: *int*, *font*: *\_typing.ReadableBuffer*, *palette*: *\_typing.ReadableBuffer*, *chars*: *\_typing.ReadableBuffer*)

Keep information about a single grid of text

Keep internal information about a grid of text in a format suitable for fast rendering with the `render()` function.

#### Parameters

- **width** (*int*) – The width of the grid in tiles, or 1 for sprites.
- **height** (*int*) – The height of the grid in tiles, or 1 for sprites.
- **font** (*ReadableBuffer*) – The font data of the characters.
- **palette** (*ReadableBuffer*) – The color palette to be used.
- **chars** (*ReadableBuffer*) – The contents of the character grid.

This class is intended for internal use in the `stage` library and it shouldn't be used on its own.

**move** (*self*, *x*: *int*, *y*: *int*) → *None*

Set the offset of the text to the specified values.

## `_typing` – Types for the C-level protocols

### `_typing.ReadableBuffer`

Classes that implement the readable buffer protocol

- *bytes*
- *bytearray*
- *memoryview*
- *array.array*
- *ulab.array*
- *rgbmatrix.RGBMatrix*

### `_typing.WritableBuffer`

Classes that implement the writable buffer protocol

- *bytearray*
- *memoryview*
- *array.array*
- *ulab.array*
- *rgbmatrix.RGBMatrix*

### `_typing.AudioSample`

Classes that implement the audiosample protocol

- `audiocore.WaveFile`
- `audiocore.RawSample`
- `audiomixer.Mixer`
- `audiomp3.MP3Decoder`

You can play these back with `audioio.AudioOut`, `audiobusio.I2SOut` or `audiopwmio.PWMAudioOut`.

#### `_typing.FrameBuffer`

Classes that implement the framebuffer protocol

- `rgbmatrix.RGBMatrix`

#### `_typing.Alarm`

Classes that implement alarms for sleeping and asynchronous notification.

- `alarm.pin.PinAlarm`
- `alarm.time.TimeAlarm`

You can use these alarms to wake up from light or deep sleep.

### adafruit\_bus\_device – Hardware accelerated external bus access

The `I2CDevice` and `SPIDevice` helper classes make managing transaction state on a bus easy. For example, they manage locking the bus to prevent other concurrent access. For SPI devices, it manages the chip select and protocol changes such as mode. For I2C, it manages the device address.

**class** `adafruit_bus_device.I2CDevice` (*i2c*: `busio.I2C`, *device\_address*: `int`, *probe*: `bool` = `True`)  
I2C Device Manager

Represents a single I2C device and manages locking the bus and the device address. :param ~busio.I2C *i2c*: The I2C bus the device is on :param int *device\_address*: The 7 bit device address :param bool *probe*: Probe for the device upon object creation, default is true

Example:

```
import busio
from board import *
from adafruit_bus_device.i2c_device import I2CDevice
with busio.I2C(SCL, SDA) as i2c:
    device = I2CDevice(i2c, 0x70)
    bytes_read = bytearray(4)
    with device:
        device.readinto(bytes_read)
    # A second transaction
    with device:
        device.write(bytes_read)
```

**\_\_enter\_\_** (*self*) → `I2CDevice`  
Context manager entry to lock bus.

**\_\_exit\_\_** (*self*) → `None`  
Automatically unlocks the bus on exit.

**readinto** (*self*, *buf*: `_typing.WriteableBuffer`, \*, *start*: `int` = 0, *end*: `Optional[int]` = `None`) → `None`  
Read into *buf* from the device. The number of bytes read will be the length of *buf*. If *start* or *end* is provided, then the buffer will be sliced as if *buf*[*start*:*end*]. This will not cause an allocation like

`buf[start:end]` will so it saves memory. :param bytearray buf: buffer to write into :param int start: Index to start writing at :param int end: Index to write up to but not include; if None, use `len(buf)`

**write**(*self*, buf: `_typing.ReadableBuffer`, \*, start: `int = 0`, end: `Optional[int] = None`) → `None`

Write the bytes from buffer to the device, then transmit a stop bit. If start or end is provided, then the buffer will be sliced as if `buffer[start:end]`. This will not cause an allocation like `buffer[start:end]` will so it saves memory. :param bytearray buf: buffer containing the bytes to write :param int start: Index to start writing from :param int end: Index to read up to but not include; if None, use `len(buf)`

**write\_then\_readinto**(*self*, out\_buffer: `_typing.WritableBuffer`, in\_buffer: `_typing.ReadableBuffer`, \*, out\_start: `int = 0`, out\_end: `Optional[int] = None`, in\_start: `int = 0`, in\_end: `Optional[int] = None`) → `None`

Write the bytes from out\_buffer to the device, then immediately reads into in\_buffer from the device. The number of bytes read will be the length of in\_buffer. If out\_start or out\_end is provided, then the output buffer will be sliced as if `out_buffer[out_start:out_end]`. This will not cause an allocation like `buffer[out_start:out_end]` will so it saves memory. If in\_start or in\_end is provided, then the input buffer will be sliced as if `in_buffer[in_start:in_end]`. This will not cause an allocation like `in_buffer[in_start:in_end]` will so it saves memory. :param bytearray out\_buffer: buffer containing the bytes to write :param bytearray in\_buffer: buffer containing the bytes to read into :param int out\_start: Index to start writing from :param int out\_end: Index to read up to but not include; if None, use `len(out_buffer)` :param int in\_start: Index to start writing at :param int in\_end: Index to write up to but not include; if None, use `len(in_buffer)`

**class** adafruit\_bus\_device.SPIDevice (*spi*: `busio.SPI`, *chip\_select*: `microcontroller.Pin`, \*, *baudrate*: `int = 100000`, *polarity*: `int = 0`, *phase*: `int = 0`, *extra\_clocks*: `int = 0`)

SPI Device Manager

Represents a single SPI device and manages locking the bus and the device address. :param ~busio.SPI spi: The SPI bus the device is on :param ~digitalio.DigitalInOut chip\_select: The chip select pin object that implements the DigitalInOut API. :param int extra\_clocks: The minimum number of clock cycles to cycle the bus after CS is high. (Used for SD cards.)

Example:

```
import busio
import digitalio
from board import *
from adafruit_bus_device.spi_device import SPIDevice
with busio.SPI(SCK, MOSI, MISO) as spi_bus:
    cs = digitalio.DigitalInOut(D10)
    device = SPIDevice(spi_bus, cs)
    bytes_read = bytearray(4)
    # The object assigned to spi in the with statements below
    # is the original spi_bus object. We are using the busio.SPI
    # operations busio.SPI.readinto() and busio.SPI.write().
    with device as spi:
        spi.readinto(bytes_read)
    # A second transaction
    with device as spi:
        spi.write(bytes_read)
```

**\_\_enter\_\_**(*self*) → `busio.SPI`

Starts a SPI transaction by configuring the SPI and asserting chip select.

**\_\_exit\_\_**(*self*) → `None`

Ends a SPI transaction by deasserting chip select. See *Lifetime and ContextManagers* for more info.

## aesio – AES encryption routines

The `AES` module contains classes used to implement encryption and decryption. It aims to be low overhead in terms of memory.

**class** `aesio.AES` (*key*: `_typing.ReadableBuffer`, *mode*: `int` = 0, *iv*: *Optional*`[_typing.ReadableBuffer]` = `None`, *segment\_size*: `int` = 8)

Encrypt and decrypt AES streams

Create a new AES state with the given key.

### Parameters

- **key** (`ReadableBuffer`) – A 16-, 24-, or 32-byte key
- **mode** (`int`) – AES mode to use. One of: `AES.MODE_ECB`, `AES.MODE_CBC`, or `AES.MODE_CTR`
- **iv** (`ReadableBuffer`) – Initialization vector to use for CBC or CTR mode

Additional arguments are supported for legacy reasons.

Encrypting a string:

```
import aesio
from binascii import hexlify

key = b'Sixteen byte key'
inp = b'Circuit Python!!!' # Note: 16-bytes long
outp = bytearray(len(inp))
cipher = aesio.AES(key, aesio.mode.MODE_ECB)
cipher.encrypt_into(inp, outp)
hexlify(outp)
```

**encrypt\_into** (*self*, *src*: `_typing.ReadableBuffer`, *dest*: `_typing.WritableBuffer`) → `None`

Encrypt the buffer from `src` into `dest`.

For ECB mode, the buffers must be 16 bytes long. For CBC mode, the buffers must be a multiple of 16 bytes, and must be equal length. For CTR mode, there are no restrictions.

**decrypt\_into** (*self*, *src*: `_typing.ReadableBuffer`, *dest*: `_typing.WritableBuffer`) → `None`

Decrypt the buffer from `src` into `dest`. For ECB mode, the buffers must be 16 bytes long. For CBC mode, the buffers must be a multiple of 16 bytes, and must be equal length. For CTR mode, there are no restrictions.

## alarm – Alarms and sleep

Provides alarms that trigger based on time intervals or on external events, such as pin changes. The program can simply wait for these alarms, or go to sleep and be awoken when they trigger.

There are two supported levels of sleep: light sleep and deep sleep.

Light sleep keeps sufficient state so the program can resume after sleeping. It does not shut down WiFi, BLE, or other communications, or ongoing activities such as audio playback. It reduces power consumption to the extent possible that leaves these continuing activities running. In some cases there may be no decrease in power consumption.

Deep sleep shuts down power to nearly all of the microcontroller including the CPU and RAM. This can save a more significant amount of power, but CircuitPython must restart `code.py` from the beginning when awakened.

For both light sleep and deep sleep, if CircuitPython is connected to a host computer, maintaining the connection takes priority and power consumption may not be reduced.

**alarm.pin – Trigger an alarm when a pin changes state.**

```
class alarm.pin.PinAlarm(pin: microcontroller.Pin, value: bool, edge: bool = False, pull: bool = False)
```

Create an alarm triggered by a `microcontroller.Pin` level. The alarm is not active until it is passed to an *alarm*-enabling function, such as `alarm.light_sleep_until_alarms()` or `alarm.exit_and_deep_sleep_until_alarms()`.

**Parameters**

- **pin** (`microcontroller.Pin`) – The pin to monitor. On some ports, the choice of pin may be limited due to hardware restrictions, particularly for deep-sleep alarms.
- **value** (`bool`) – When active, trigger when the pin value is high (`True`) or low (`False`). On some ports, multiple `PinAlarm` objects may need to have coordinated values for deep-sleep alarms.
- **edge** (`bool`) – If `True`, trigger only when there is a transition to the specified value of value. If `True`, if the alarm becomes active when the pin value already matches `value`, the alarm is not triggered: the pin must transition from `not value` to `value` to trigger the alarm. On some ports, edge-triggering may not be available, particularly for deep-sleep alarms.
- **pull** (`bool`) – Enable a pull-up or pull-down which pulls the pin to the level opposite that of `value`. For instance, if `value` is set to `True`, setting `pull` to `True` will enable a pull-down, to hold the pin low normally until an outside signal pulls it high.

**pin** :`microcontroller.Pin`

The trigger pin.

**value** :`bool`

The value on which to trigger.

**alarm.time – Trigger an alarm when the specified time is reached.**

```
class alarm.time.TimeAlarm(monotonic_time: Optional[float] = None, epoch_time: Optional[int] = None)
```

Create an alarm that will be triggered when `time.monotonic()` would equal `monotonic_time`, or when `time.time()` would equal `epoch_time`. Only one of the two arguments can be given. The alarm is not active until it is passed to an *alarm*-enabling function, such as `alarm.light_sleep_until_alarms()` or `alarm.exit_and_deep_sleep_until_alarms()`.

If the given time is in the past when sleep occurs, the alarm will be triggered immediately.

**monotonic\_time** :`float`

When this time is reached, the alarm will trigger, based on the `time.monotonic()` clock. The time may be given as `epoch_time` in the constructor, but it is returned by this property only as a `time.monotonic()` time.



**alarm.touch – Trigger an alarm when touch is detected.**

**class** `alarm.touch.TouchAlarm(*pin: microcontroller.Pin)`

Create an alarm that will be triggered when the given pin is touched. The alarm is not active until it is passed to an *alarm*-enabling function, such as `alarm.light_sleep_until_alarms()` or `alarm.exit_and_deep_sleep_until_alarms()`.

**Parameters** `pin` ([microcontroller.Pin](#)) – The pin to monitor. On some ports, the choice of pin may be limited due to hardware restrictions, particularly for deep-sleep alarms.

**pin** :[microcontroller.Pin](#)

The trigger pin.

`alarm.sleep_memory` :[SleepMemory](#)

Memory that persists during deep sleep. This object is the sole instance of `alarm.SleepMemory`.

`alarm.wake_alarm` :[\\_typing.Alarm](#)

The most recently triggered alarm. If CircuitPython was sleeping, the alarm the woke it from sleep.

`alarm.light_sleep_until_alarms(*alarms: \_typing.Alarm) → \_typing.Alarm`

Go into a light sleep until awakened one of the alarms. The alarm causing the wake-up is returned, and is also available as `alarm.wake_alarm`.

If no alarms are specified, return immediately.

**If CircuitPython is connected to a host computer, the connection will be maintained, and the microcontroller may not actually go into a light sleep.** This allows the user to interrupt an existing program with ctrl-C, and to edit the files in CIRCUITPY, which would not be possible in true light sleep. Thus, to use light sleep and save significant power,

`alarm.exit_and_deep_sleep_until_alarms(*alarms: \_typing.Alarm) → None`

Exit the program and go into a deep sleep, until awakened by one of the alarms. This function does not return.

When awakened, the microcontroller will restart and will run `boot.py` and `code.py` from the beginning.

After restart, an alarm *equivalent* to the one that caused the wake-up will be available as `alarm.wake_alarm`. Its type and/or attributes may not correspond exactly to the original alarm. For time-base alarms, currently, an `alarm.time.TimeAlarm()` is created.

If no alarms are specified, the microcontroller will deep sleep until reset.

**If CircuitPython is connected to a host computer, the connection will be maintained, and the system will not go into deep sleep.** This allows the user to interrupt an existing program with ctrl-C, and to edit the files in CIRCUITPY, which would not be possible in true deep sleep. Thus, to use deep sleep and save significant power, you will need to disconnect from the host.

Here is skeletal example that deep-sleeps and restarts every 60 seconds:

```
import alarm
import time

print("Waking up")

# Set an alarm for 60 seconds from now.
time_alarm = alarm.time.TimeAlarm(monotonic_time=time.monotonic() + 60)

# Deep sleep until the alarm goes off. Then restart the program.
alarm.exit_and_deep_sleep_until_alarms(time_alarm)
```

**class** `alarm.SleepMemory`

Store raw bytes in RAM that persists during deep sleep. The class acts as a `bytearray`. If power is lost, the memory contents are lost.

Note that this class can't be imported and used directly. The sole instance of `SleepMemory` is available at `alarm.sleep_memory`.

Usage:

```
import alarm
alarm.sleep_memory[0] = True
alarm.sleep_memory[1] = 12
```

Not used. Access the sole instance through `alarm.sleep_memory`.

`__bool__`(*self*) → *bool*

`sleep_memory` is True if its length is greater than zero. This is an easy way to check for its existence.

`__len__`(*self*) → *int*

Return the length. This is used by (`len`)

`__getitem__`(*self*, *index*: *slice*) → *bytearray*

`__getitem__`(*self*, *index*: *int*) → *int*

Returns the value at the given index.

`__setitem__`(*self*, *index*: *slice*, *value*: `_typing.ReadableBuffer`) → *None*

`__setitem__`(*self*, *index*: *int*, *value*: *int*) → *None*

Set the value at the given index.

## analogio – Analog hardware support

The `analogio` module contains classes to provide access to analog IO typically implemented with digital-to-analog (DAC) and analog-to-digital (ADC) converters.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See *Lifetime and ContextManagers* for more info.

For example:

```
import analogio
from board import *

pin = analogio.AnalogIn(A0)
print(pin.value)
pin.deinit()
```

This example will initialize the the device, read *value* and then `deinit()` the hardware. The last step is optional because CircuitPython will do it automatically after the program finishes.

**class** `analogio.AnalogIn`(*pin*: `microcontroller.Pin`)

Read analog voltage levels

Usage:

```
import analogio
from board import *
```

(continues on next page)

(continued from previous page)

```
adc = analogio.AnalogIn(A1)
val = adc.value
```

Use the `AnalogIn` on the given pin. The reference voltage varies by platform so use `reference_voltage` to read the configured setting.

**Parameters** `pin` (`Pin`) – the pin to read from

**value** :int

The value on the analog pin between 0 and 65535 inclusive (16-bit). (read-only)

Even if the underlying analog to digital converter (ADC) is lower resolution, the value is 16-bit.

**reference\_voltage** :float

The maximum voltage measurable (also known as the reference voltage) as a *float* in Volts.

**deinit** (*self*) → `None`

Turn off the `AnalogIn` and release the pin for other use.

**\_\_enter\_\_** (*self*) → `AnalogIn`

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → `None`

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**class** `analogio.AnalogOut` (*pin*: `microcontroller.Pin`)

Output analog values (a specific voltage).

Example usage:

```
import analogio
from board import *

dac = analogio.AnalogOut(A2)           # output on pin A2
dac.value = 32768                      # makes A2 1.65V
```

Use the `AnalogOut` on the given pin.

**Parameters** `pin` (`Pin`) – the pin to output to

**value** :int

The value on the analog pin between 0 and 65535 inclusive (16-bit). (write-only)

Even if the underlying digital to analog converter (DAC) is lower resolution, the value is 16-bit.

**deinit** (*self*) → `None`

Turn off the `AnalogOut` and release the pin for other use.

**\_\_enter\_\_** (*self*) → `AnalogOut`

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → `None`

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**audiobusio – Support for audio input and output over digital buses**

The `audiobusio` module contains classes to provide access to audio IO over digital buses. These protocols are used to communicate audio to other chips in the same circuit. It doesn't include audio interconnect protocols such as S/PDIF.

All libraries change hardware state and should be deinitialized when they are no longer needed. To do so, either call `deinit()` or use a context manager.

**class** `audiobusio.I2SOut` (*bit\_clock*: `microcontroller.Pin`, *word\_select*: `microcontroller.Pin`, *data*: `microcontroller.Pin`, \*, *left\_justified*: `bool`)

Output an I2S audio signal

Create a I2SOut object associated with the given pins.

**Parameters**

- **bit\_clock** (`Pin`) – The bit clock (or serial clock) pin
- **word\_select** (`Pin`) – The word select (or left/right clock) pin
- **data** (`Pin`) – The data pin
- **left\_justified** (`bool`) – True when data bits are aligned with the word select clock. False when they are shifted by one to match classic I2S protocol.

Simple 8ksp/s 440 Hz sine wave on [Metro M0 Express](#) using [UDA1334 Breakout](#):

```
import audiobusio
import audiocore
import board
import array
import time
import math

# Generate one period of sine wave.
length = 8000 // 440
sine_wave = array.array("H", [0] * length)
for i in range(length):
    sine_wave[i] = int(math.sin(math.pi * 2 * i / length) * (2 ** 15) + 2 ** 15)

sine_wave = audiocore.RawSample(sine_wave, sample_rate=8000)
i2s = audiobusio.I2SOut(board.D1, board.D0, board.D9)
i2s.play(sine_wave, loop=True)
time.sleep(1)
i2s.stop()
```

Playing a wave file from flash:

```
import board
import audioio
import audiocore
import audiobusio
import digitalio

f = open("cplay-5.1-16bit-16khz.wav", "rb")
wav = audiocore.WaveFile(f)

a = audiobusio.I2SOut(board.D1, board.D0, board.D9)
```

(continues on next page)

(continued from previous page)

```
print("playing")
a.play(wav)
while a.playing:
    pass
print("stopped")
```

**playing** :bool

True when the audio sample is being output. (read-only)

**paused** :bool

True when playback is paused. (read-only)

**deinit** (*self*) → None

Deinitialises the I2SOut and releases any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → *I2SOut*

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → NoneAutomatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.**play** (*self*, *sample*: *\_typing.AudioSample*, \*, *loop*: bool = False) → NonePlays the sample once when loop=False and continuously when loop=True. Does not block. Use *playing* to block.Sample must be an *audiocore.WaveFile*, *audiocore.RawSample*, *audiomixer.Mixer* or *audiomp3.MP3Decoder*.

The sample itself should consist of 8 bit or 16 bit samples.

**stop** (*self*) → None

Stops playback.

**pause** (*self*) → NoneStops playback temporarily while remembering the position. Use *resume* to resume playback.**resume** (*self*) → NoneResumes sample playback after *pause()*.

```
class audiobusio.PDMIn(clock_pin: microcontroller.Pin, data_pin: microcontroller.Pin, *, sample_rate: int = 16000, bit_depth: int = 8, mono: bool = True, oversample: int = 64, startup_delay: float = 0.11)
```

Record an input PDM audio stream

Create a PDMIn object associated with the given pins. This allows you to record audio signals from the given pins. Individual ports may put further restrictions on the recording parameters. The overall sample rate is determined by *sample\_rate* x *oversample*, and the total must be 1MHz or higher, so *sample\_rate* must be a minimum of 16000.

**Parameters**

- **clock\_pin** (*Pin*) – The pin to output the clock to
- **data\_pin** (*Pin*) – The pin to read the data from
- **sample\_rate** (*int*) – Target *sample\_rate* of the resulting samples. Check *sample\_rate* for actual value. Minimum *sample\_rate* is about 16000 Hz.
- **bit\_depth** (*int*) – Final number of bits per sample. Must be divisible by 8

- **mono** (`bool`) – True when capturing a single channel of audio, captures two channels otherwise
- **oversample** (`int`) – Number of single bit samples to decimate into a final sample. Must be divisible by 8
- **startup\_delay** (`float`) – seconds to wait after starting microphone clock to allow microphone to turn on. Most require only 0.01s; some require 0.1s. Longer is safer. Must be in range 0.0-1.0 seconds.

**sample\_rate** : `int`

The actual `sample_rate` of the recording. This may not match the constructed sample rate due to internal clock limitations.

**deinit** (`self`) → `None`

Deinitialises the PDMin and releases any hardware resources for reuse.

**\_\_enter\_\_** (`self`) → `PDMin`

No-op used by Context Managers.

**\_\_exit\_\_** (`self`) → `None`

Automatically deinitializes the hardware when exiting a context.

**record** (`self`, `destination`: `_typing.WriteableBuffer`, `destination_length`: `int`) → `None`

Records `destination_length` bytes of samples to destination. This is blocking.

An `IOError` may be raised when the destination is too slow to record the audio at the given rate. For internal flash, writing all 1s to the file before recording is recommended to speed up writes.

**Returns** The number of samples recorded. If this is less than `destination_length`, some samples were missed due to processing time.

## audiocore – Support for audio samples

**class** `audiocore.RawSample` (`buffer`: `_typing.ReadableBuffer`, \*, `channel_count`: `int` = 1, `sample_rate`: `int` = 8000)

A raw audio sample buffer in memory

Create a `RawSample` based on the given buffer of signed values. If `channel_count` is more than 1 then each channel's samples should alternate. In other words, for a two channel buffer, the first sample will be for channel 1, the second sample will be for channel two, the third for channel 1 and so on.

### Parameters

- **buffer** (`ReadableBuffer`) – A buffer with samples
- **channel\_count** (`int`) – The number of channels in the buffer
- **sample\_rate** (`int`) – The desired playback sample rate

Simple 8ksps 440 Hz sin wave:

```
import audiocore
import audioio
import board
import array
import time
import math

# Generate one period of sine wav.
length = 8000 // 440
```

(continues on next page)

(continued from previous page)

```

sine_wave = array.array("h", [0] * length)
for i in range(length):
    sine_wave[i] = int(math.sin(math.pi * 2 * i / length) * (2 ** 15))

dac = audioio.AudioOut(board.SPEAKER)
sine_wave = audiocore.RawSample(sine_wave)
dac.play(sine_wave, loop=True)
time.sleep(1)
dac.stop()

```

**sample\_rate** :Optional[int]

32 bit value that dictates how quickly samples are played in Hertz (cycles per second). When the sample is looped, this can change the pitch output without changing the underlying sample. This will not change the sample rate of any active playback. Call `play` again to change it.

**deinit** (*self*) → None

Deinitialises the AudioOut and releases any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → RawSample

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → None

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**class** audiocore.WaveFile (*file*: BinaryIO, *buffer*: \_typing.WriteableBuffer)

Load a wave file for audio playback

A .wav file prepped for audio playback. Only mono and stereo files are supported. Samples must be 8 bit unsigned or 16 bit signed. If a buffer is provided, it will be used instead of allocating an internal buffer.

Load a .wav file for playback with *audioio.AudioOut* or *audiobusio.I2SOut*.

**Parameters**

- **file** (*typing.BinaryIO*) – Already opened wave file
- **buffer** (*WriteableBuffer*) – Optional pre-allocated buffer, that will be split in half and used for double-buffering of the data. If not provided, two 512 byte buffers are allocated internally.

Playing a wave file from flash:

```

import board
import audiocore
import audioio
import digitalio

# Required for CircuitPlayground Express
speaker_enable = digitalio.DigitalInOut(board.SPEAKER_ENABLE)
speaker_enable.switch_to_output(value=True)

data = open("cplay-5.1-16bit-16khz.wav", "rb")
wav = audiocore.WaveFile(data)
a = audioio.AudioOut(board.A0)

print("playing")
a.play(wav)
while a.playing:

```

(continues on next page)

(continued from previous page)

```
pass
print("stopped")
```

**sample\_rate :int**

32 bit value that dictates how quickly samples are loaded into the DAC in Hertz (cycles per second). When the sample is looped, this can change the pitch output without changing the underlying sample.

**bits\_per\_sample :int**

Bits per sample. (read only)

**channel\_count :int**

Number of audio channels. (read only)

**deinit** (*self*) → *None*

Deinitialises the WaveFile and releases all memory resources for reuse.

**\_\_enter\_\_** (*self*) → *WaveFile*

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → *None*

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

## audioio – Support for audio output

The *audioio* module contains classes to provide access to audio IO.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See *Lifetime and ContextManagers* for more info.

Since CircuitPython 5, *RawSample* and *WaveFile* are moved to *audiocore*, and *Mixer* is moved to *audiomixer*.

For compatibility with CircuitPython 4.x, some builds allow the items in *audiocore* to be imported from *audioio*. This will be removed for all boards in a future build of CircuitPython.

**class** *audioio.AudioOut* (*left\_channel*: *microcontroller.Pin*, \*, *right\_channel*: *Optional[microcontroller.Pin]* = *None*, *quiescent\_value*: *int* = 32768)

Output an analog audio signal

Create a *AudioOut* object associated with the given pin(s). This allows you to play audio signals out on the given pin(s).

### Parameters

- **left\_channel** (*Pin*) – The pin to output the left channel to
- **right\_channel** (*Pin*) – The pin to output the right channel to
- **quiescent\_value** (*int*) – The output value when no signal is present. Samples should start and end with this value to prevent audible popping.

Simple 8ksps 440 Hz sin wave:

```
import audiocore
import audioio
import board
import array
```

(continues on next page)



(continued from previous page)

```

import time
import math

# Generate one period of sine wav.
length = 8000 // 440
sine_wave = array.array("H", [0] * length)
for i in range(length):
    sine_wave[i] = int(math.sin(math.pi * 2 * i / length) * (2 ** 15) + 2 ** 15)

dac = audioio.AudioOut(board.SPEAKER)
sine_wave = audiocore.RawSample(sine_wave, sample_rate=8000)
dac.play(sine_wave, loop=True)
time.sleep(1)
dac.stop()

```

Playing a wave file from flash:

```

import board
import audioio
import digitalio

# Required for CircuitPlayground Express
speaker_enable = digitalio.DigitalInOut(board.SPEAKER_ENABLE)
speaker_enable.switch_to_output(value=True)

data = open("cplay-5.1-16bit-16khz.wav", "rb")
wav = audiocore.WaveFile(data)
a = audioio.AudioOut(board.A0)

print("playing")
a.play(wav)
while a.playing:
    pass
print("stopped")

```

**playing** :bool

True when an audio sample is being output even if *paused*. (read-only)

**paused** :bool

True when playback is paused. (read-only)

**deinit** (*self*) → None

Deinitialises the AudioOut and releases any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → *AudioOut*

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → None

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**play** (*self*, *sample*: *typing.AudioSample*, \*, *loop*: bool = False) → None

Plays the sample once when loop=False and continuously when loop=True. Does not block. Use *playing* to block.

Sample must be an *audiocore.WaveFile*, *audiocore.RawSample*, *audiomixer.Mixer* or *audiomp3.MP3Decoder*.

The sample itself should consist of 16 bit samples. Microcontrollers with a lower output resolution will

use the highest order bits to output. For example, the SAMD21 has a 10 bit DAC that ignores the lowest 6 bits when playing 16 bit samples.

**stop** (*self*) → *None*

Stops playback and resets to the start of the sample.

**pause** (*self*) → *None*

Stops playback temporarily while remembering the position. Use *resume* to resume playback.

**resume** (*self*) → *None*

Resumes sample playback after *pause()*.

## audiomixer – Support for audio mixing

**class** `audiomixer.Mixer` (*voice\_count*: `int` = 2, *buffer\_size*: `int` = 1024, *channel\_count*: `int` = 2, *bits\_per\_sample*: `int` = 16, *samples\_signed*: `bool` = *True*, *sample\_rate*: `int` = 8000)

Mixes one or more audio samples together into one sample.

Create a Mixer object that can mix multiple channels with the same sample rate. Samples are accessed and controlled with the mixer's *audiomixer.MixerVoice* objects.

### Parameters

- **voice\_count** (`int`) – The maximum number of voices to mix
- **buffer\_size** (`int`) – The total size in bytes of the buffers to mix into
- **channel\_count** (`int`) – The number of channels the source samples contain. 1 = mono; 2 = stereo.
- **bits\_per\_sample** (`int`) – The bits per sample of the samples being played
- **samples\_signed** (`bool`) – Samples are signed (*True*) or unsigned (*False*)
- **sample\_rate** (`int`) – The sample rate to be used for all samples

Playing a wave file from flash:

```
import board
import audioio
import audiocore
import audiomixer
import digitalio

a = audioio.AudioOut(board.A0)
music = audiocore.WaveFile(open("cplay-5.1-16bit-16khz.wav", "rb"))
drum = audiocore.WaveFile(open("drum.wav", "rb"))
mixer = audiomixer.Mixer(voice_count=2, sample_rate=16000, channel_count=1,
                          bits_per_sample=16, samples_signed=True)

print("playing")
# Have AudioOut play our Mixer source
a.play(mixer)
# Play the first sample voice
mixer.voice[0].play(music)
while mixer.playing:
    # Play the second sample voice
    mixer.voice[1].play(drum)
    time.sleep(1)
print("stopped")
```

**playing :bool**

True when any voice is being output. (read-only)

**sample\_rate :int**

32 bit value that dictates how quickly samples are played in Hertz (cycles per second).

**voice :Tuple[MixerVoice, ...]**

A tuple of the mixer's `audiomixer.MixerVoice` object(s).

```
>>> mixer.voice
(<MixerVoice>,)
```

**deinit (self) → None**

Deinitialises the Mixer and releases any hardware resources for reuse.

**\_\_enter\_\_ (self) → Mixer**

No-op used by Context Managers.

**\_\_exit\_\_ (self) → None**

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**play (self, sample: \_typing.AudioSample, \*, voice: int = 0, loop: bool = False) → None**

Plays the sample once when loop=False and continuously when loop=True. Does not block. Use `playing` to block.

Sample must be an `audiocore.WaveFile`, `audiocore.RawSample`, `audiomixer.Mixer` or `audiomp3.MP3Decoder`.

The sample must match the Mixer's encoding settings given in the constructor.

**stop\_voice (self, voice: int = 0) → None**

Stops playback of the sample on the given voice.

**class audiomixer.MixerVoice**

Voice objects used with Mixer

Used to access and control samples with `audiomixer.Mixer`.

MixerVoice instance object(s) created by `audiomixer.Mixer`.

**level :float**

The volume level of a voice, as a floating point number between 0 and 1.

**playing :bool**

True when this voice is being output. (read-only)

**play (self, sample: \_typing.AudioSample, \*, loop: bool = False) → None**

Plays the sample once when loop=False, and continuously when loop=True. Does not block. Use `playing` to block.

Sample must be an `audiocore.WaveFile`, `audiocore.RawSample`, `audiomixer.Mixer` or `audiomp3.MP3Decoder`.

The sample must match the `audiomixer.Mixer`'s encoding settings given in the constructor.

**stop (self) → None**

Stops playback of the sample on this voice.

## audiomp3 – Support for MP3-compressed audio files

**class** `audiomp3.MP3Decoder` (*file*: `BinaryIO`, *buffer*: `_typing.WriteableBuffer`)

Load a mp3 file for audio playback

Load a .mp3 file for playback with `audioio.AudioOut` or `audiobusio.I2SOut`.

### Parameters

- **file** (`typing.BinaryIO`) – Already opened mp3 file
- **buffer** (`WriteableBuffer`) – Optional pre-allocated buffer, that will be split in half and used for double-buffering of the data. If not provided, two buffers are allocated internally. The specific buffer size required depends on the mp3 file.

Playing a mp3 file from flash:

```
import board
import audiomp3
import audioio
import digitalio

# Required for CircuitPlayground Express
speaker_enable = digitalio.DigitalInOut(board.SPEAKER_ENABLE)
speaker_enable.switch_to_output(value=True)

data = open("cplay-16bit-16khz-64kbps.mp3", "rb")
mp3 = audiomp3.MP3Decoder(data)
a = audioio.AudioOut(board.A0)

print("playing")
a.play(mp3)
while a.playing:
    pass
print("stopped")
```

**file** :`BinaryIO`

File to play back.

**sample\_rate** :`int`

32 bit value that dictates how quickly samples are loaded into the DAC in Hertz (cycles per second). When the sample is looped, this can change the pitch output without changing the underlying sample.

**bits\_per\_sample** :`int`

Bits per sample. (read only)

**channel\_count** :`int`

Number of audio channels. (read only)

**rms\_level** :`float`

The RMS audio level of a recently played moment of audio. (read only)

**deinit** (*self*) → `None`

Deinitialises the MP3 and releases all memory resources for reuse.

**\_\_enter\_\_** (*self*) → `MP3Decoder`

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → `None`

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

## audiopwmio – Audio output via digital PWM

The `audiopwmio` module contains classes to provide access to audio IO.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See [Lifetime and ContextManagers](#) for more info.

Since CircuitPython 5, `Mixer`, `RawSample` and `WaveFile` are moved to `audiocore`.

```
class audiopwmio.PWMAudioOut (left_channel:  microcontroller.Pin, *, right_channel:  Op-
                                tional[microcontroller.Pin] = None, quiescent_value:  int =
                                32768)
```

Output an analog audio signal by varying the PWM duty cycle.

Create a PWMAudioOut object associated with the given pin(s). This allows you to play audio signals out on the given pin(s). In contrast to `mod:audioio`, the pin(s) specified are digital pins, and are driven with a device-dependent PWM signal.

### Parameters

- **left\_channel** (`Pin`) – The pin to output the left channel to
- **right\_channel** (`Pin`) – The pin to output the right channel to
- **quiescent\_value** (`int`) – The output value when no signal is present. Samples should start and end with this value to prevent audible popping.

Simple 8ksps 440 Hz sin wave:

```
import audiocore
import audiopwmio
import board
import array
import time
import math

# Generate one period of sine wav.
length = 8000 // 440
sine_wave = array.array("H", [0] * length)
for i in range(length):
    sine_wave[i] = int(math.sin(math.pi * 2 * i / length) * (2 ** 15) + 2 ** 15)

dac = audiopwmio.PWMAudioOut (board.SPEAKER)
sine_wave = audiocore.RawSample(sine_wave, sample_rate=8000)
dac.play(sine_wave, loop=True)
time.sleep(1)
dac.stop()
```

Playing a wave file from flash:

```
import board
import audiocore
import audiopwmio
import digitalio

# Required for CircuitPlayground Express
speaker_enable = digitalio.DigitalInOut (board.SPEAKER_ENABLE)
speaker_enable.switch_to_output (value=True)
```

(continues on next page)

(continued from previous page)

```
data = open("cplay-5.1-16bit-16khz.wav", "rb")
wav = audiocore.WaveFile(data)
a = audiopwmio.PWMAudioOut(board.SPEAKER)

print("playing")
a.play(wav)
while a.playing:
    pass
print("stopped")
```

**playing** :boolTrue when an audio sample is being output even if *paused*. (read-only)**paused** :bool

True when playback is paused. (read-only)

**deinit** (*self*) → None

Deinitialises the PWMAudioOut and releases any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → *PWMAudioOut*

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → NoneAutomatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.**play** (*self*, *sample*: *\_typing.AudioSample*, \*, *loop*: bool = False) → NonePlays the sample once when loop=False and continuously when loop=True. Does not block. Use *playing* to block.Sample must be an *audiocore.WaveFile*, *audiocore.RawSample*, *audiomixer.Mixer* or *audiomp3.MP3Decoder*.

The sample itself should consist of 16 bit samples. Microcontrollers with a lower output resolution will use the highest order bits to output. For example, the SAMD21 has a 10 bit DAC that ignores the lowest 6 bits when playing 16 bit samples.

**stop** (*self*) → None

Stops playback and resets to the start of the sample.

**pause** (*self*) → NoneStops playback temporarily while remembering the position. Use *resume* to resume playback.**resume** (*self*) → NoneResumes sample playback after *pause()*.

## bitbangio – Digital protocols implemented by the CPU

The *bitbangio* module contains classes to provide digital bus protocol support regardless of whether the underlying hardware exists to use the protocol.

First try to use *busio* module instead which may utilize peripheral hardware to implement the protocols. Native implementations will be faster than bitbanged versions and have more capabilities.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See *Lifetime and ContextManagers* for more info.

For example:

```
import bitbangio
from board import *

i2c = bitbangio.I2C(SCL, SDA)
print(i2c.scan())
i2c.deinit()
```

This example will initialize the the device, run `scan()` and then `deinit()` the hardware. The last step is optional because CircuitPython automatically resets hardware after a program finishes.

```
class bitbangio.I2C(scl: microcontroller.Pin, sda: microcontroller.Pin, *, frequency: int = 400000,
                    timeout: int = 255)
```

Two wire serial protocol

I2C is a two-wire protocol for communicating between devices. At the physical level it consists of 2 wires: SCL and SDA, the clock and data lines respectively.

**See also:**

Using this class directly requires careful lock management. Instead, use `I2CDevice` to manage locks.

**See also:**

Using this class to directly read registers requires manual bit unpacking. Instead, use an existing driver or make one with `Register` data descriptors.

#### Parameters

- **scl** (`Pin`) – The clock pin
- **sda** (`Pin`) – The data pin
- **frequency** (`int`) – The clock frequency of the bus
- **timeout** (`int`) – The maximum clock stretching timeout in microseconds

**deinit** (`self`) → `None`

Releases control of the underlying hardware so other classes can use it.

**\_\_enter\_\_** (`self`) → `I2C`

No-op used in Context Managers.

**\_\_exit\_\_** (`self`) → `None`

Automatically deinitializes the hardware on context exit. See *Lifetime and ContextManagers* for more info.

**scan** (`self`) → `List[int]`

Scan all I2C addresses between 0x08 and 0x77 inclusive and return a list of those that respond. A device responds if it pulls the SDA line low after its address (including a read bit) is sent on the bus.

**try\_lock** (`self`) → `bool`

Attempts to grab the I2C lock. Returns True on success.

**unlock** (`self`) → `None`

Releases the I2C lock.

**readfrom\_into** (`self`, `address: int`, `buffer: _typing.WritableBuffer`, \*, `start: int = 0`, `end: Optional[int] = None`) → `None`

Read into `buffer` from the device selected by `address`. The number of bytes read will be the length of `buffer`. At least one byte must be read.

If `start` or `end` is provided, then the buffer will be sliced as if `buffer[start:end]`. This will not cause an allocation like `buf[start:end]` will so it saves memory.

**Parameters**

- **address** (`int`) – 7-bit device address
- **buffer** (`WriteableBuffer`) – buffer to write into
- **start** (`int`) – Index to start writing at
- **end** (`int`) – Index to write up to but not include

**writeto** (*self*, *address*: `int`, *buffer*: `_typing.ReadableBuffer`, \*, *start*: `int` = 0, *end*: `Optional[int]` = `None`, *stop*: `bool` = `True`) → `None`

Write the bytes from *buffer* to the device selected by *address* and then transmits a stop bit. Use *writeto\_then\_readfrom* when needing a write, no stop and repeated start before a read.

If *start* or *end* is provided, then the buffer will be sliced as if *buffer[start:end]*. This will not cause an allocation like *buffer[start:end]* will so it saves memory.

Writing a buffer or slice of length zero is permitted, as it can be used to poll for the existence of a device.

**Parameters**

- **address** (`int`) – 7-bit device address
- **buffer** (`ReadableBuffer`) – buffer containing the bytes to write
- **start** (`int`) – Index to start writing from
- **end** (`int`) – Index to read up to but not include

**writeto\_then\_readfrom** (*self*, *address*: `int`, *out\_buffer*: `_typing.ReadableBuffer`, *in\_buffer*: `_typing.ReadableBuffer`, \*, *out\_start*: `int` = 0, *out\_end*: `Optional[int]` = `None`, *in\_start*: `int` = 0, *in\_end*: `Optional[int]` = `None`) → `None`

Write the bytes from *out\_buffer* to the device selected by *address*, generate no stop bit, generate a repeated start and read into *in\_buffer*. *out\_buffer* and *in\_buffer* can be the same buffer because they are used sequentially.

If *start* or *end* is provided, then the corresponding buffer will be sliced as if *buffer[start:end]*. This will not cause an allocation like *buf[start:end]* will so it saves memory.

**Parameters**

- **address** (`int`) – 7-bit device address
- **out\_buffer** (`ReadableBuffer`) – buffer containing the bytes to write
- **in\_buffer** (`WriteableBuffer`) – buffer to write into
- **out\_start** (`int`) – Index to start writing from
- **out\_end** (`int`) – Index to read up to but not include. Defaults to `len(buffer)`
- **in\_start** (`int`) – Index to start writing at
- **in\_end** (`int`) – Index to write up to but not include. Defaults to `len(buffer)`

**class** `bitbangio.OneWire` (*pin*: `microcontroller.Pin`)

Lowest-level of the Maxim OneWire protocol

*OneWire* implements the timing-sensitive foundation of the Maxim (formerly Dallas Semi) OneWire protocol.

Protocol definition is here: <https://www.maximintegrated.com/en/app-notes/index.mvp/id/126>

Create a *OneWire* object associated with the given pin. The object implements the lowest level timing-sensitive bits of the protocol.

**Parameters** *pin* (`Pin`) – Pin to read pulses from.



Read a short series of pulses:

```
import bitbangio
import board

onewire = bitbangio.OneWire(board.D7)
onewire.reset()
onewire.write_bit(True)
onewire.write_bit(False)
print(onewire.read_bit())
```

**deinit** (*self*) → *None*

Deinitialize the OneWire bus and release any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → *OneWire*

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → *None*

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**reset** (*self*) → *bool*

Reset the OneWire bus

**read\_bit** (*self*) → *bool*

Read in a bit

**Returns** bit state read

**Return type** *bool*

**write\_bit** (*self*, *value*: *bool*) → *None*

Write out a bit based on value.

**class** `bitbangio.SPI` (*clock*: *microcontroller.Pin*, *MOSI*: *Optional[microcontroller.Pin] = None*, *MISO*: *Optional[microcontroller.Pin] = None*)

A 3-4 wire serial protocol

SPI is a serial protocol that has exclusive pins for data in and out of the main device. It is typically faster than *I2C* because a separate pin is used to select a device rather than a transmitted address. This class only manages three of the four SPI lines: `clock`, `MOSI`, `MISO`. Its up to the client to manage the appropriate select line, often abbreviated `CS` or `SS`. (This is common because multiple secondaries can share the `clock`, `MOSI` and `MISO` lines and therefore the hardware.)

Construct an SPI object on the given pins.

**See also:**

Using this class directly requires careful lock management. Instead, use `SPIDevice` to manage locks.

**See also:**

Using this class to directly read registers requires manual bit unpacking. Instead, use an existing driver or make one with `Register` data descriptors.

#### Parameters

- **clock** (*Pin*) – the pin to use for the clock.
- **MOSI** (*Pin*) – the Main Out Selected In pin.
- **MISO** (*Pin*) – the Main In Selected Out pin.

**deinit** (*self*) → *None*

Turn off the SPI bus.

**\_\_enter\_\_** (*self*) → *SPI*

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → *None*

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**configure** (*self*, \*, *baudrate*: *int* = 100000, *polarity*: *int* = 0, *phase*: *int* = 0, *bits*: *int* = 8) → *None*

Configures the SPI bus. Only valid when locked.

#### Parameters

- **baudrate** (*int*) – the clock rate in Hertz
- **polarity** (*int*) – the base state of the clock line (0 or 1)
- **phase** (*int*) – the edge of the clock that data is captured. First (0) or second (1). Rising or falling depends on clock polarity.
- **bits** (*int*) – the number of bits per word

**try\_lock** (*self*) → *bool*

Attempts to grab the SPI lock. Returns True on success.

**Returns** True when lock has been grabbed

**Return type** *bool*

**unlock** (*self*) → *None*

Releases the SPI lock.

**write** (*self*, *buf*: *typing.ReadableBuffer*) → *None*

Write the data contained in *buf*. Requires the SPI being locked. If the buffer is empty, nothing happens.

**readinto** (*self*, *buffer*: *typing.WriteableBuffer*, \*, *start*: *int* = 0, *end*: *Optional[int]* = *None*, *write\_value*: *int* = 0) → *None*

Read into buffer while writing *write\_value* for each byte read. The SPI object must be locked. If the number of bytes to read is 0, nothing happens.

#### Parameters

- **buffer** (*bytearray*) – Read data into this buffer
- **start** (*int*) – Start of the slice of *buffer* to read into: *buffer[start:end]*
- **end** (*int*) – End of the slice; this index is not included. Defaults to *len(buffer)*
- **write\_value** (*int*) – Value to write while reading.

**write\_readinto** (*self*, *buffer\_out*: *typing.ReadableBuffer*, *buffer\_in*: *typing.ReadableBuffer*, \*, *out\_start*: *int* = 0, *out\_end*: *Optional[int]* = *None*, *in\_start*: *int* = 0, *in\_end*: *Optional[int]* = *None*) → *None*

Write out the data in *buffer\_out* while simultaneously reading data into *buffer\_in*. The SPI object must be locked. The lengths of the slices defined by *buffer\_out[out\_start:out\_end]* and *buffer\_in[in\_start:in\_end]* must be equal. If buffer slice lengths are both 0, nothing happens.

#### Parameters

- **buffer\_out** (*ReadableBuffer*) – Write out the data in this buffer
- **buffer\_in** (*WriteableBuffer*) – Read data into this buffer

- **out\_start** (*int*) – Start of the slice of `buffer_out` to write out: `buffer_out[out_start:out_end]`
- **out\_end** (*int*) – End of the slice; this index is not included. Defaults to `len(buffer_out)`
- **in\_start** (*int*) – Start of the slice of `buffer_in` to read into: `buffer_in[in_start:in_end]`
- **in\_end** (*int*) – End of the slice; this index is not included. Defaults to `len(buffer_in)`

## bitmaptools – Collection of bitmap manipulation tools

`bitmaptools.rototozoom` (*dest\_bitmap: displayio.Bitmap*, *source\_bitmap: displayio.Bitmap*, \*, *ox: int*, *oy: int*, *dest\_clip0: Tuple[int, int]*, *dest\_clip1: Tuple[int, int]*, *px: int*, *py: int*, *source\_clip0: Tuple[int, int]*, *source\_clip1: Tuple[int, int]*, *angle: float*, *scale: float*, *skip\_index: int*) → *None*

Inserts the source bitmap region into the destination bitmap with rotation (angle), scale and clipping (both on source and destination bitmaps).

### Parameters

- **dest\_bitmap** (*bitmap*) – Destination bitmap that will be copied into
- **source\_bitmap** (*bitmap*) – Source bitmap that contains the graphical region to be copied
- **ox** (*int*) – Horizontal pixel location in destination bitmap where source bitmap point (px,py) is placed
- **oy** (*int*) – Vertical pixel location in destination bitmap where source bitmap point (px,py) is placed
- **dest\_clip0** (*Tuple[int, int]*) – First corner of rectangular destination clipping region that constrains region of writing into destination bitmap
- **dest\_clip1** (*Tuple[int, int]*) – Second corner of rectangular destination clipping region that constrains region of writing into destination bitmap
- **px** (*int*) – Horizontal pixel location in source bitmap that is placed into the destination bitmap at (ox,oy)
- **py** (*int*) – Vertical pixel location in source bitmap that is placed into the destination bitmap at (ox,oy)
- **source\_clip0** (*Tuple[int, int]*) – First corner of rectangular source clipping region that constrains region of reading from the source bitmap
- **source\_clip1** (*Tuple[int, int]*) – Second corner of rectangular source clipping region that constrains region of reading from the source bitmap
- **angle** (*float*) – Angle of rotation, in radians (positive is clockwise direction)
- **scale** (*float*) – Scaling factor
- **skip\_index** (*int*) – Bitmap palette index in the source that will not be copied, set to *None* to copy all pixels

`bitmaptools.fill_region` (*dest\_bitmap*: `displayio.Bitmap`, *x1*: `int`, *y1*: `int`, *x2*: `int`, *y2*: `int`, *value*: `int`)  
→ `None`

Draws the color value into the destination bitmap within the rectangular region bounded by (x1,y1) and (x2,y2), exclusive.

#### Parameters

- **dest\_bitmap** (*bitmap*) – Destination bitmap that will be written into
- **x1** (`int`) – x-pixel position of the first corner of the rectangular fill region
- **y1** (`int`) – y-pixel position of the first corner of the rectangular fill region
- **x2** (`int`) – x-pixel position of the second corner of the rectangular fill region (exclusive)
- **y2** (`int`) – y-pixel position of the second corner of the rectangular fill region (exclusive)
- **value** (`int`) – Bitmap palette index that will be written into the rectangular fill region in the destination bitmap

`bitmaptools.draw_line` (*dest\_bitmap*: `displayio.Bitmap`, *x1*: `int`, *y1*: `int`, *x2*: `int`, *y2*: `int`, *value*: `int`) →  
`None`

Draws a line into a bitmap specified two endpoints (x1,y1) and (x2,y2).

#### Parameters

- **dest\_bitmap** (*bitmap*) – Destination bitmap that will be written into
- **x1** (`int`) – x-pixel position of the line's first endpoint
- **y1** (`int`) – y-pixel position of the line's first endpoint
- **x2** (`int`) – x-pixel position of the line's second endpoint
- **y2** (`int`) – y-pixel position of the line's second endpoint
- **value** (`int`) – Bitmap palette index that will be written into the line in the destination bitmap

`bitmaptools.arrayblit` (*bitmap*: `displayio.Bitmap`, *data*: `_typing.ReadableBuffer`, *x1*: `int` = 0, *y1*: `int` = 0, *x2*: *Optional*[`int`] = `None`, *y2*: *Optional*[`int`] = `None`, *skip\_index*: *Optional*[`int`] = `None`) → `None`

Inserts pixels from *data* into the rectangle of width×height pixels with the upper left corner at (*x*, *y*)

The values from *data* are taken modulo the number of color values available in the destination bitmap.

If *x1* or *y1* are not specified, they are taken as 0. If *x2* or *y2* are not specified, or are given as -1, they are taken as the width and height of the image.

The coordinates affected by the blit are *x1* ≤ *x* < *x2* and *y1* ≤ *y* < *y2*.

*data* must contain at least as many elements as required. If it contains excess elements, they are ignored.

The blit takes place by rows, so the first elements of *data* go to the first row, the next elements to the next row, and so on.

#### Parameters

- **bitmap** (`displayio.Bitmap`) – A writable bitmap
- **data** (`ReadableBuffer`) – Buffer containing the source pixel values
- **x1** (`int`) – The left corner of the area to blit into (inclusive)
- **y1** (`int`) – The top corner of the area to blit into (inclusive)
- **x2** (`int`) – The right of the area to blit into (exclusive)
- **y2** (`int`) – The bottom corner of the area to blit into (exclusive)

- **skip\_index** (*int*) – Bitmap palette index in the source that will not be copied, set to None to copy all pixels

`bitmaptools.readinto` (*bitmap*: `displayio.Bitmap`, *file*: `BinaryIO`, *bits\_per\_pixel*: *int*, *element\_size*: *int* = 1, *reverse\_pixels\_in\_element*: *bool* = False, *swap\_bytes\_in\_element*: *bool* = False, *reverse\_rows*: *bool* = False) → None

Reads from a binary file into a bitmap.

The file must be positioned so that it consists of `bitmap.height` rows of pixel data, where each row is the smallest multiple of `element_size` bytes that can hold `bitmap.width` pixels.

The bytes in an element can be optionally swapped, and the pixels in an element can be reversed. Also, the row loading direction can be reversed, which may be required for loading certain bitmap files.

This function doesn't parse image headers, but is useful to speed up loading of uncompressed image formats such as PCF glyph data.

#### Parameters

- **bitmap** (`displayio.Bitmap`) – A writable bitmap
- **file** (`typing.BinaryIO`) – A file opened in binary mode
- **bits\_per\_pixel** (*int*) – Number of bits per pixel. Values 1, 2, 4, 8, 16, 24, and 32 are supported;
- **element\_size** (*int*) – Number of bytes per element. Values of 1, 2, and 4 are supported, except that 24 `bits_per_pixel` requires 1 byte per element.
- **reverse\_pixels\_in\_element** (*bool*) – If set, the first pixel in a word is taken from the Most Significant Bits; otherwise, it is taken from the Least Significant Bits.
- **swap\_bytes\_in\_element** (*bool*) – If the `element_size` is not 1, then reverse the byte order of each element read.
- **reverse\_rows** (*bool*) – Reverse the direction of the row loading (required for some bitmap images).

### bitops – Routines for low-level manipulation of binary data

`bitops.bit_transpose` (*input*: `_typing.ReadableBuffer`, *output*: `_typing.WritableBuffer`, *width*: *int* = 8) → `_typing.WritableBuffer`

“Transpose” a buffer by assembling each output byte with bits taken from each of `width` different input bytes.

This can be useful to convert a sequence of pixel values into a single stream of bytes suitable for sending via a parallel conversion method.

The number of bytes in the input buffer must be a multiple of the width, and the width can be any value from 2 to 8. If the width is fewer than 8, then the remaining (less significant) bits of the output are set to zero.

Let `stride = len(input) // width`. Then the first byte is made out of the most significant bits of `[input[0], input[stride], input[2*stride], ...]`. The second byte is made out of the second bits, and so on until the 8th output byte which is made of the first bits of `input[1], input[1+stride], input[2*stride], ...]`.

The required output buffer size is `len(input) * 8 // width`.

Returns the output buffer.

## board – Board specific pin names

Common container for board base pin names. These will vary from board to board so don't expect portability when using this module.

**Warning:** The board module varies by board. The APIs documented here may or may not be available on a specific board.

`board.I2C()` → *busio.I2C*

Returns the *busio.I2C* object for the board designated SDA and SCL pins. It is a singleton.

`board.SPI()` → *busio.SPI*

Returns the *busio.SPI* object for the board designated SCK, MOSI and MISO pins. It is a singleton.

`board.UART()` → *busio.UART*

Returns the *busio.UART* object for the board designated TX and RX pins. It is a singleton.

The object created uses the default parameter values for *busio.UART*. If you need to set parameters that are not changeable after creation, such as `receiver_buffer_size`, do not use `board.UART()`; instead create a *busio.UART* object explicitly with the desired parameters.

## busio – Hardware accelerated external bus access

The *busio* module contains classes to support a variety of serial protocols.

When the microcontroller does not support the behavior in a hardware accelerated fashion it may internally use a bitbang routine. However, if hardware support is available on a subset of pins but not those provided, then a `RuntimeError` will be raised. Use the *bitbangio* module to explicitly bitbang a serial protocol on any general purpose pins.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See *Lifetime and ContextManagers* for more info.

For example:

```
import busio
from board import *

i2c = busio.I2C(SCL, SDA)
print(i2c.scan())
i2c.deinit()
```

This example will initialize the the device, run `scan()` and then `deinit()` the hardware. The last step is optional because CircuitPython automatically resets hardware after a program finishes.

**class** *busio.I2C*(*scl*: *microcontroller.Pin*, *sda*: *microcontroller.Pin*, \*, *frequency*: *int* = 100000, *timeout*: *int* = 255)

Two wire serial protocol

I2C is a two-wire protocol for communicating between devices. At the physical level it consists of 2 wires: SCL and SDA, the clock and data lines respectively.

**See also:**

Using this class directly requires careful lock management. Instead, use *I2CDevice* to manage locks.

**See also:**

Using this class to directly read registers requires manual bit unpacking. Instead, use an existing driver or make one with [Register](#) data descriptors.

**Parameters**

- **scl** ([Pin](#)) – The clock pin
- **sda** ([Pin](#)) – The data pin
- **frequency** ([int](#)) – The clock frequency in Hertz
- **timeout** ([int](#)) – The maximum clock stretching timeout - (used only for `bitbangio.I2C`; ignored for `busio.I2C`)

---

**Note:** On the nRF52840, only one I2C object may be created, except on the Circuit Playground Bluefruit, which allows two, one for the onboard accelerometer, and one for offboard use.

---

**deinit** (*self*) → [None](#)

Releases control of the underlying hardware so other classes can use it.

**\_\_enter\_\_** (*self*) → [I2C](#)

No-op used in Context Managers.

**\_\_exit\_\_** (*self*) → [None](#)

Automatically deinitializes the hardware on context exit. See [Lifetime and ContextManagers](#) for more info.

**scan** (*self*) → List[[int](#)]

Scan all I2C addresses between 0x08 and 0x77 inclusive and return a list of those that respond.

**Returns** List of device ids on the I2C bus

**Return type** [list](#)

**try\_lock** (*self*) → [bool](#)

Attempts to grab the I2C lock. Returns True on success.

**Returns** True when lock has been grabbed

**Return type** [bool](#)

**unlock** (*self*) → [None](#)

Releases the I2C lock.

**readfrom\_into** (*self*, *address*: [int](#), *buffer*: [\\_typing.WriteableBuffer](#), \*, *start*: [int](#) = 0, *end*: *Optional*[[int](#)] = *None*) → [None](#)

Read into *buffer* from the device selected by *address*. The number of bytes read will be the length of *buffer*. At least one byte must be read.

If *start* or *end* is provided, then the buffer will be sliced as if `buffer[start:end]`. This will not cause an allocation like `buf[start:end]` will so it saves memory.

**Parameters**

- **address** ([int](#)) – 7-bit device address
- **buffer** ([WriteableBuffer](#)) – buffer to write into
- **start** ([int](#)) – Index to start writing at
- **end** ([int](#)) – Index to write up to but not include. Defaults to `len(buffer)`

**writeto** (*self*, *address*: `int`, *buffer*: `_typing.ReadableBuffer`, \*, *start*: `int` = 0, *end*: `Optional[int]` = `None`, *stop*: `bool` = `True`) → `None`

Write the bytes from *buffer* to the device selected by *address* and then transmit a stop bit.

If *start* or *end* is provided, then the buffer will be sliced as if `buffer[start:end]`. This will not cause an allocation like `buffer[start:end]` will so it saves memory.

Writing a buffer or slice of length zero is permitted, as it can be used to poll for the existence of a device.

#### Parameters

- **address** (`int`) – 7-bit device address
- **buffer** (`ReadableBuffer`) – buffer containing the bytes to write
- **start** (`int`) – Index to start writing from
- **end** (`int`) – Index to read up to but not include. Defaults to `len(buffer)`

**writeto\_then\_readfrom** (*self*, *address*: `int`, *out\_buffer*: `_typing.ReadableBuffer`, *in\_buffer*: `_typing.WritableBuffer`, \*, *out\_start*: `int` = 0, *out\_end*: `Optional[int]` = `None`, *in\_start*: `int` = 0, *in\_end*: `Optional[int]` = `None`) → `None`

Write the bytes from *out\_buffer* to the device selected by *address*, generate no stop bit, generate a repeated start and read into *in\_buffer*. *out\_buffer* and *in\_buffer* can be the same buffer because they are used sequentially.

if *start* or *end* is provided, then the corresponding buffer will be sliced as if `buffer[start:end]`. This will not cause an allocation like `buf[start:end]` will so it saves memory.

#### Parameters

- **address** (`int`) – 7-bit device address
- **out\_buffer** (`ReadableBuffer`) – buffer containing the bytes to write
- **in\_buffer** (`WritableBuffer`) – buffer to write into
- **out\_start** (`int`) – Index to start writing from
- **out\_end** (`int`) – Index to read up to but not include. Defaults to `len(buffer)`
- **in\_start** (`int`) – Index to start writing at
- **in\_end** (`int`) – Index to write up to but not include. Defaults to `len(buffer)`

**class** `busio.OneWire` (*pin*: `microcontroller.Pin`)

Lowest-level of the Maxim OneWire protocol

(formerly Dallas Semi) OneWire protocol.

Protocol definition is here: <https://www.maximintegrated.com/en/app-notes/index.mvp/id/126>

**class** `OneWire` (*pin*)

Create a `OneWire` object associated with the given pin. The object implements the lowest level timing-sensitive bits of the protocol.

**Parameters** *pin* (`Pin`) – Pin connected to the `OneWire` bus

Read a short series of pulses:

```
import busio
import board

onewire = busio.OneWire(board.D7)
onewire.reset()
onewire.write_bit(True)
```

(continues on next page)



(continued from previous page)

```
onewire.write_bit(False)
print(onewire.read_bit())
```

**deinit** (*self*) → *None*

Deinitialize the OneWire bus and release any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → *OneWire*

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → *None*Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.**reset** (*self*) → *bool*

Reset the OneWire bus and read presence

**Returns** False when at least one device is present**Return type** *bool***read\_bit** (*self*) → *bool*

Read in a bit

**Returns** bit state read**Return type** *bool***write\_bit** (*self*, *value*: *bool*) → *None*

Write out a bit based on value.

**class** `busio.SPI` (*clock*: *microcontroller.Pin*, *MOSI*: *Optional[microcontroller.Pin]* = *None*, *MISO*: *Optional[microcontroller.Pin]* = *None*)

A 3-4 wire serial protocol

SPI is a serial protocol that has exclusive pins for data in and out of the main device. It is typically faster than *I2C* because a separate pin is used to select a device rather than a transmitted address. This class only manages three of the four SPI lines: `clock`, `MOSI`, `MISO`. Its up to the client to manage the appropriate select line, often abbreviated `CS` or `SS`. (This is common because multiple secondaries can share the `clock`, `MOSI` and `MISO` lines and therefore the hardware.)

Construct an SPI object on the given pins.

---

**Note:** The SPI peripherals allocated in order of desirability, if possible, such as highest speed and not shared use first. For instance, on the nRF52840, there is a single 32MHz SPI peripheral, and multiple 8MHz peripherals, some of which may also be used for I2C. The 32MHz SPI peripheral is returned first, then the exclusive 8MHz SPI peripheral, and finally the shared 8MHz peripherals.

---

**See also:**Using this class directly requires careful lock management. Instead, use `SPIDevice` to manage locks.**See also:**Using this class to directly read registers requires manual bit unpacking. Instead, use an existing driver or make one with `Register` data descriptors.**Parameters**

- **clock** (*Pin*) – the pin to use for the clock.
- **MOSI** (*Pin*) – the Main Out Selected In pin.

- **MISO** (`Pin`) – the Main In Selected Out pin.

**frequency** :int

The actual SPI bus frequency. This may not match the frequency requested due to internal limitations.

**deinit** (*self*) → `None`

Turn off the SPI bus.

**\_\_enter\_\_** (*self*) → `SPI`

No-op used by Context Managers. Provided by context manager helper.

**\_\_exit\_\_** (*self*) → `None`

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**configure** (*self*, \*, *baudrate*: int = 100000, *polarity*: int = 0, *phase*: int = 0, *bits*: int = 8) → `None`

Configures the SPI bus. The SPI object must be locked.

#### Parameters

- **baudrate** (int) – the desired clock rate in Hertz. The actual clock rate may be higher or lower due to the granularity of available clock settings. Check the *frequency* attribute for the actual clock rate.
- **polarity** (int) – the base state of the clock line (0 or 1)
- **phase** (int) – the edge of the clock that data is captured. First (0) or second (1). Rising or falling depends on clock polarity.
- **bits** (int) – the number of bits per word

---

**Note:** On the SAMD21, it is possible to set the baudrate to 24 MHz, but that speed is not guaranteed to work. 12 MHz is the next available lower speed, and is within spec for the SAMD21.

---

---

**Note:** On the nRF52840, these baudrates are available: 125kHz, 250kHz, 1MHz, 2MHz, 4MHz, and 8MHz. If you pick a a baudrate other than one of these, the nearest lower baudrate will be chosen, with a minimum of 125kHz. Two SPI objects may be created, except on the Circuit Playground Bluefruit, which allows only one (to allow for an additional I2C object).

---

**try\_lock** (*self*) → `bool`

Attempts to grab the SPI lock. Returns True on success.

**Returns** True when lock has been grabbed

**Return type** `bool`

**unlock** (*self*) → `None`

Releases the SPI lock.

**write** (*self*, *buffer*: `typing.ReadableBuffer`, \*, *start*: int = 0, *end*: `Optional[int]` = None) → `None`

Write the data contained in *buffer*. The SPI object must be locked. If the buffer is empty, nothing happens.

#### Parameters

- **buffer** (`ReadableBuffer`) – Write out the data in this buffer
- **start** (int) – Start of the slice of *buffer* to write out: *buffer*[*start*:*end*]
- **end** (int) – End of the slice; this index is not included. Defaults to `len(buffer)`

**readinto** (*self*, *buffer*: `_typing.WriteableBuffer`, \*, *start*: `int` = 0, *end*: `Optional[int]` = None, *write\_value*: `int` = 0) → `None`

Read into buffer while writing *write\_value* for each byte read. The SPI object must be locked. If the number of bytes to read is 0, nothing happens.

#### Parameters

- **buffer** (`WriteableBuffer`) – Read data into this buffer
- **start** (`int`) – Start of the slice of *buffer* to read into: *buffer*[*start*:*end*]
- **end** (`int`) – End of the slice; this index is not included. Defaults to `len(buffer)`
- **write\_value** (`int`) – Value to write while reading. (Usually ignored.)

**write\_readinto** (*self*, *buffer\_out*: `_typing.ReadableBuffer`, *buffer\_in*: `_typing.WriteableBuffer`, \*, *out\_start*: `int` = 0, *out\_end*: `Optional[int]` = None, *in\_start*: `int` = 0, *in\_end*: `Optional[int]` = None) → `None`

Write out the data in *buffer\_out* while simultaneously reading data into *buffer\_in*. The SPI object must be locked. The lengths of the slices defined by *buffer\_out*[*out\_start*:*out\_end*] and *buffer\_in*[*in\_start*:*in\_end*] must be equal. If buffer slice lengths are both 0, nothing happens.

#### Parameters

- **buffer\_out** (`ReadableBuffer`) – Write out the data in this buffer
- **buffer\_in** (`WriteableBuffer`) – Read data into this buffer
- **out\_start** (`int`) – Start of the slice of *buffer\_out* to write out: *buffer\_out*[*out\_start*:*out\_end*]
- **out\_end** (`int`) – End of the slice; this index is not included. Defaults to `len(buffer_out)`
- **in\_start** (`int`) – Start of the slice of *buffer\_in* to read into: *buffer\_in*[*in\_start*:*in\_end*]
- **in\_end** (`int`) – End of the slice; this index is not included. Defaults to `len(buffer_in)`

**class** `busio.UART` (*tx*: `microcontroller.Pin`, *rx*: `microcontroller.Pin`, \*, *baudrate*: `int` = 9600, *bits*: `int` = 8, *parity*: `Optional[Parity]` = None, *stop*: `int` = 1, *timeout*: `float` = 1, *receiver\_buffer\_size*: `int` = 64)

A bidirectional serial protocol

A common bidirectional serial protocol that uses an agreed upon speed rather than a shared clock line.

#### Parameters

- **tx** (`Pin`) – the pin to transmit with, or None if this UART is receive-only.
- **rx** (`Pin`) – the pin to receive on, or None if this UART is transmit-only.
- **rts** (`Pin`) – the pin for rts, or None if rts not in use.
- **cts** (`Pin`) – the pin for cts, or None if cts not in use.
- **rs485\_dir** (`Pin`) – the output pin for rs485 direction setting, or None if rs485 not in use.
- **rs485\_invert** (`bool`) – rs485\_dir pin active high when set. Active low otherwise.
- **baudrate** (`int`) – the transmit and receive speed.
- **bits** (`int`) – the number of bits per byte, 5 to 9.
- **parity** (`Parity`) – the parity used for error checking.
- **stop** (`int`) – the number of stop bits, 1 or 2.

- **timeout** (*float*) – the timeout in seconds to wait for the first character and between subsequent characters when reading. Raises `ValueError` if timeout > 100 seconds.
- **receiver\_buffer\_size** (*int*) – the character length of the read buffer (0 to disable). (When a character is 9 bits the buffer will be 2 \* receiver\_buffer\_size bytes.)

*New in CircuitPython 4.0:* timeout has incompatibly changed units from milliseconds to seconds. The new upper limit on timeout is meant to catch mistaken use of milliseconds.

**baudrate** : *int*

The current baudrate.

**in\_waiting** : *int*

The number of bytes in the input buffer, available to be read

**timeout** : *float*

The current timeout, in seconds (float).

**deinit** (*self*) → *None*

Deinitialises the UART and releases any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → *UART*

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → *None*

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**read** (*self*, *nbytes*: *Optional[int]* = *None*) → *Optional[bytes]*

Read characters. If *nbytes* is specified then read at most that many bytes. Otherwise, read everything that arrives until the connection times out. Providing the number of bytes expected is highly recommended because it will be faster.

**Returns** Data read

**Return type** *bytes* or *None*

**readinto** (*self*, *buf*: *typing.WriteableBuffer*) → *Optional[int]*

Read bytes into the *buf*. Read at most `len(buf)` bytes.

**Returns** number of bytes read and stored into *buf*

**Return type** *int* or *None* (on a non-blocking error)

*New in CircuitPython 4.0:* No length parameter is permitted.

**readline** (*self*) → *bytes*

**Read a line, ending in a newline character, or** return *None* if a timeout occurs sooner, or return everything readable if no newline is found and `timeout=0`

**Returns** the line read

**Return type** *bytes* or *None*

**write** (*self*, *buf*: *typing.WriteableBuffer*) → *Optional[int]*

Write the buffer of bytes to the bus.

*New in CircuitPython 4.0:* *buf* must be bytes, not a string.

**return** the number of bytes written

**rtype** *int* or *None*

**reset\_input\_buffer**(*self*) → *None*

Discard any unread characters in the input buffer.

**class** busio.**Parity**

Enum-like class to define the parity used to verify correct data transfer.

**ODD** :int

Total number of ones should be odd.

**EVEN** :int

Total number of ones should be even.

## camera – Support for camera input

The *camera* module contains classes to control the camera and take pictures.

**class** camera.**Camera**

The class to control camera.

Usage:

```
import board
import sdioio
import storage
import camera

sd = sdioio.SDCard(
    clock=board.SDIO_CLOCK,
    command=board.SDIO_COMMAND,
    data=board.SDIO_DATA,
    frequency=25000000)
vfs = storage.VfsFat(sd)
storage.mount(vfs, '/sd')

cam = camera.Camera()

buffer = bytearray(512 * 1024)
file = open("/sd/image.jpg", "wb")
size = cam.take_picture(buffer, width=1920, height=1080, format=camera.
    ↳ ImageFormat.JPG)
file.write(buffer, size)
file.close()
```

Initialize camera.

**deinit**(*self*) → *None*

De-initialize camera.

**take\_picture**(*self*, *buf*: *typing*.WriteableBuffer, *format*: ImageFormat) → *int*

Take picture and save to *buf* in the given *format*. The size of the picture taken is width by height in pixels.

**Returns** the number of bytes written into *buf*

**Return type** *int*

**class** camera.**ImageFormat**

Image format

Enum-like class to define the image format.

**JPG : ImageFormat**

JPG format.

**RGB565 : ImageFormat**

RGB565 format.

## canio – CAN bus access

The `canio` module contains low level classes to support the CAN bus protocol.

CAN and Listener classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See [Lifetime and ContextManagers](#) for more info.

For example:

```
import canio
from board import *

can = canio.CAN(board.CAN_RX, board.CAN_TX, baudrate=1000000)
message = canio.Message(id=0x0408, data=b"adafruit")
can.send(message)
can.deinit()
```

This example will write the data ‘adafruit’ onto the CAN bus to any device listening for message id 0x0408.

A CAN bus involves a transceiver, which is often a separate chip with a “standby” pin. If your board has a `CAN_STANDBY` pin, ensure to set it to an output with the value `False` to enable the transceiver.

Other implementations of the CAN device may exist (for instance, attached via an SPI bus). If so their constructor arguments may differ, but otherwise we encourage implementors to follow the API that the core uses.

**class canio.BusState**

The state of the CAN bus

**ERROR\_ACTIVE :object**

The bus is in the normal (active) state

**ERROR\_WARNING :object**

The bus is in the normal (active) state, but a moderate number of errors have occurred recently.

NOTE: Not all implementations may use `ERROR_WARNING`. Do not rely on seeing `ERROR_WARNING` before `ERROR_PASSIVE`.

**ERROR\_PASSIVE :object**

The bus is in the passive state due to the number of errors that have occurred recently.

This device will acknowledge packets it receives, but cannot transmit messages. If additional errors occur, this device may progress to `BUS_OFF`. If it successfully acknowledges other packets on the bus, it can return to `ERROR_WARNING` or `ERROR_ACTIVE` and transmit packets.

**BUS\_OFF :object**

The bus has turned off due to the number of errors that have occurred recently. It must be restarted before it will send or receive packets. This device will neither send or acknowledge packets on the bus.

**class canio.CAN** (*tx*: `microcontroller.Pin`, *rx*: `microcontroller.Pin`, \*, *baudrate*: `int` = 250000, *loopback*: `bool` = `False`, *silent*: `bool` = `False`, *auto\_restart*: `bool` = `False`)

CAN bus protocol

A common shared-bus protocol. The rx and tx pins are generally connected to a transceiver which controls the H and L pins on a shared bus.

**Parameters**

- **rx** ([Pin](#)) – the pin to receive with
- **tx** ([Pin](#)) – the pin to transmit with
- **baudrate** ([int](#)) – The bit rate of the bus in Hz. All devices on the bus must agree on this value.
- **loopback** ([bool](#)) – When True the rx pin’s value is ignored, and the device receives the packets it sends.
- **silent** ([bool](#)) – When True the tx pin is always driven to the high logic level. This mode can be used to “sniff” a CAN bus without interfering.
- **auto\_restart** ([bool](#)) – If True, will restart communications after entering bus-off state

**auto\_restart** :[bool](#)

If True, will restart communications after entering bus-off state

**baudrate** :[int](#)

The baud rate (read-only)

**transmit\_error\_count** :[int](#)

The number of transmit errors (read-only). Increased for a detected transmission error, decreased for successful transmission. Limited to the range from 0 to 255 inclusive. Also called TEC.

**receive\_error\_count** :[int](#)

The number of receive errors (read-only). Increased for a detected reception error, decreased for successful reception. Limited to the range from 0 to 255 inclusive. Also called REC.

**state** :[BusState](#)

The current state of the bus. (read-only)

**loopback** :[bool](#)

True if the device was created in loopback mode, False otherwise (read-only)

**silent** :[bool](#)

True if the device was created in silent mode, False otherwise (read-only)

**restart** (*self*) → [None](#)

If the device is in the bus off state, restart it.

**listen** (*self*, *matches*: *Optional[Sequence[Match]] = None*, \*, *timeout*: [float](#) = 10) → [Listener](#)

Start receiving messages that match any one of the filters.

Creating a listener is an expensive operation and can interfere with reception of messages by other listeners.

There is an implementation-defined maximum number of listeners and limit to the complexity of the filters.

If the hardware cannot support all the requested matches, a `ValueError` is raised. Note that generally there are some number of hardware filters shared among all fifos.

A message can be received by at most one Listener. If more than one listener matches a message, it is undefined which one actually receives it.

An empty filter list causes all messages to be accepted.

Timeout dictates how long `receive()` and `next()` will block.

Platform specific notes:

SAM E5x supports two Listeners. Filter blocks are shared between the two listeners. There are 4 standard filter blocks and 4 extended filter blocks. Each block can either match 2 single addresses or a mask

of addresses. The number of filter blocks can be increased, up to a hardware maximum, by rebuilding CircuitPython, but this decreases the CircuitPython free memory even if canio is not used.

STM32F405 supports two Listeners. Filter blocks are shared between the two listeners. There are 14 filter blocks. Each block can match 2 standard addresses with mask or 1 extended address with mask.

ESP32S2 supports one Listener. There is a single filter block, which can either match a standard address with mask or an extended address with mask.

**send** (*self*, *message*: Union[RemoteTransmissionRequest, Message]) → None

Send a message on the bus with the given data and id. If the message could not be sent due to a full fifo or a bus error condition, RuntimeError is raised.

**deinit** (*self*) → None

Deinitialize this object, freeing its hardware resources

**\_\_enter\_\_** (*self*) → CAN

Returns self, to allow the object to be used in a The with statement statement for resource control

**\_\_exit\_\_** (*self*, *unused1*: Optional[Type[BaseException]], *unused2*: Optional[BaseException], *unused3*: Optional[types.TracebackType]) → None

Calls deinit()

**class** canio.Listener

Listens for CAN message

*canio.Listener* is not constructed directly, but instead by calling *canio.CAN.listen*.

In addition to using the *receive* method to retrieve a message or the *in\_waiting* method to check for an available message, a listener can be used as an iterable, yielding messages until no message arrives within *self.timeout* seconds.

**timeout** :float

**receive** (*self*) → Optional[Union[RemoteTransmissionRequest, Message]]

Reads a message, after waiting up to *self.timeout* seconds

If no message is received in time, None is returned. Otherwise, a Message or RemoteTransmissionRequest is returned.

**in\_waiting** (*self*) → int

Returns the number of messages (including remote transmission requests) waiting

**\_\_iter\_\_** (*self*) → Listener

Returns self

This method exists so that Listener can be used as an iterable

**\_\_next\_\_** (*self*) → Union[RemoteTransmissionRequest, Message]

Reads a message, after waiting up to *self.timeout* seconds

If no message is received in time, raises StopIteration. Otherwise, a Message or is returned.

This method enables the Listener to be used as an iterable, for instance in a for-loop.

**deinit** (*self*) → None

Deinitialize this object, freeing its hardware resources

**\_\_enter\_\_** (*self*) → CAN

Returns self, to allow the object to be used in a The with statement statement for resource control

**\_\_exit\_\_** (*self*, *unused1*: Optional[Type[BaseException]], *unused2*: Optional[BaseException], *unused3*: Optional[types.TracebackType]) → None

Calls deinit()



```
class canio.Match (id: int, *, mask: Optional[int] = None, extended: bool = False)
```

Describe CAN bus messages to match

Construct a Match with the given properties.

If mask is not None, then the filter is for any id which matches all the nonzero bits in mask. Otherwise, it matches exactly the given id. If extended is true then only extended ids are matched, otherwise only standard ids are matched.

```
id :int
```

The id to match

```
mask :int
```

The optional mask of ids to match

```
extended :bool
```

True to match extended ids, False to match standard ids

```
class canio.Message (id: int, data: bytes, *, extended: bool = False)
```

Construct a Message to send on a CAN bus.

#### Parameters

- **id** (**int**) – The numeric ID of the message
- **data** (**bytes**) – The content of the message
- **extended** (**bool**) – True if the message has an extended identifier, False if it has a standard identifier

In CAN, messages can have a length from 0 to 8 bytes.

```
id :int
```

The numeric ID of the message

```
data :bytes
```

The content of the message

```
extended :bool
```

True if the message's id is an extended id

```
class canio.RemoteTransmissionRequest (id: int, length: int, *, extended: bool = False)
```

Construct a RemoteTransmissionRequest to send on a CAN bus.

#### Parameters

- **id** (**int**) – The numeric ID of the requested message
- **length** (**int**) – The length of the requested message
- **extended** (**bool**) – True if the message has an extended identifier, False if it has a standard identifier

In CAN, messages can have a length from 0 to 8 bytes.

```
id :int
```

The numeric ID of the message

```
extended :bool
```

True if the message's id is an extended id

```
length :int
```

The length of the requested message.

## countio – Support for edge counting

The `countio` module contains logic to read and count edge transistions

**Warning:** This module is not available in some SAMD21 (aka M0) builds. See the [Module Support Matrix - Which Modules Are Available on Which Boards](#) for more info.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See [Lifetime and ContextManagers](#) for more info.

**class** `countio.Counter` (*pin\_a*: `microcontroller.Pin`)

Counter will keep track of the number of falling edge transistions (pulses) on a given pin

Create a Counter object associated with the given pin. It tracks the number of falling pulses relative when the object is constructed.

**Parameters** `pin_a` (`Pin`) – Pin to read pulses from.

For example:

```
import board
import countio

pin_counter = countio.Counter(board.D1)
#reset the count after 100 counts
while True:
    if pin_counter.count == 100:
        pin_counter.reset()
    print(pin_counter.count)
```

**count** :int

The current count in terms of pulses.

**deinit** (*self*) → `None`

Deinitializes the Counter and releases any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → `Counter`

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → `None`

Automatically deinitializes the hardware when exiting a context. See [Lifetime and ContextManagers](#) for more info.

**reset** (*self*) → `None`

Resets the count back to 0.

## digitalio – Basic digital pin support

The `digitalio` module contains classes to provide access to basic digital IO.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See [Lifetime and ContextManagers](#) for more info.

For example:

```
import digitalio
from board import *

pin = digitalio.DigitalInOut(D13)
print(pin.value)
```

This example will initialize the the device, read `value` and then `deinit()` the hardware.

Here is blinky:

```
import digitalio
from board import *
import time

led = digitalio.DigitalInOut(D13)
led.direction = digitalio.Direction.OUTPUT
while True:
    led.value = True
    time.sleep(0.1)
    led.value = False
    time.sleep(0.1)
```

### **class** `digitalio.DriveMode`

Defines the drive mode of a digital pin

Enum-like class to define the drive mode used when outputting digital values.

#### **PUSH\_PULL :DriveMode**

Output both high and low digital values

#### **OPEN\_DRAIN :DriveMode**

Output low digital values but go into high z for digital high. This is useful for i2c and other protocols that share a digital line.

### **class** `digitalio.DigitalInOut` (*pin*: `microcontroller.Pin`)

Digital input and output

A `DigitalInOut` is used to digitally control I/O pins. For analog control of a pin, see the `analogio.AnalogIn` and `analogio.AnalogOut` classes.

Create a new `DigitalInOut` object associated with the pin. Defaults to input with no pull. Use `switch_to_input()` and `switch_to_output()` to change the direction.

**Parameters** `pin` (`Pin`) – The pin to control

#### **direction :Direction**

The direction of the pin.

Setting this will use the defaults from the corresponding `switch_to_input()` or `switch_to_output()` method. If you want to set pull, value or drive mode prior to switching, then use those methods instead.

**value** :bool

The digital logic level of the pin.

**drive\_mode** :DriveMode

The pin drive mode. One of:

- `digitalio.DriveMode.PUSH_PULL`
- `digitalio.DriveMode.OPEN_DRAIN`

**pull** :Optional[Pull]

The pin pull direction. One of:

- `digitalio.Pull.UP`
- `digitalio.Pull.DOWN`
- `None`

Raises **AttributeError** – if *direction* is `OUTPUT`.

**deinit** (*self*) → None

Turn off the DigitalInOut and release the pin for other use.

**\_\_enter\_\_** (*self*) → `DigitalInOut`

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → None

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**switch\_to\_output** (*self*, *value*: bool = False, *drive\_mode*: DriveMode = DriveMode.PUSH\_PULL) → None

Set the drive mode and value and then switch to writing out digital values.

#### Parameters

- **value** (bool) – default value to set upon switching
- **drive\_mode** (DriveMode) – drive mode for the output

**switch\_to\_input** (*self*, *pull*: Optional[Pull] = None) → None

Set the pull and then switch to read in digital values.

**Parameters** **pull** (Pull) – pull configuration for the input

Example usage:

```
import digitalio
import board

switch = digitalio.DigitalInOut(board.SLIDE_SWITCH)
switch.switch_to_input(pull=digitalio.Pull.UP)
# Or, after switch_to_input
switch.pull = digitalio.Pull.UP
print(switch.value)
```

**class** digitalio.Direction

Defines the direction of a digital pin

Enum-like class to define which direction the digital values are going.

**INPUT** :Direction

Read digital data in

**OUTPUT :Direction**  
Write digital data out

**class digitalio.Pull**

Defines the pull of a digital input pin

Enum-like class to define the pull value, if any, used while reading digital values in.

**UP :Pull**

When the input line isn't being driven the pull up can pull the state of the line high so it reads as true.

**DOWN :Pull**

When the input line isn't being driven the pull down can pull the state of the line low so it reads as false.

## displayio – Native helpers for driving displays

The `displayio` module contains classes to manage display output including synchronizing with refresh rates and partial updating.

`displayio.release_displays()` → `None`

Releases any actively used displays so their busses and pins can be used again. This will also release the builtin display on boards that have one. You will need to reinitialize it yourself afterwards. This may take seconds to complete if an active EPaperDisplay is refreshing.

Use this once in your code.py if you initialize a display. Place it right before the initialization so the display is active as long as possible.

**class displayio.Bitmap** (*width: int, height: int, value\_count: int*)

Stores values of a certain size in a 2D array

Bitmaps can be treated as read-only buffers. If the number of bits in a pixel is 8, 16, or 32; and the number of bytes per row is a multiple of 4, then the resulting memoryview will correspond directly with the bitmap's contents. Otherwise, the bitmap data is packed into the memoryview with unspecified padding.

A read-only buffer can be used e.g., with `ulab.frombuffer` to efficiently create an array with the same content as a Bitmap; to move data efficiently from ulab back into a Bitmap, use `bitmaptools.arrayblit`.

Create a Bitmap object with the given fixed size. Each pixel stores a value that is used to index into a corresponding palette. This enables differently colored sprites to share the underlying Bitmap. `value_count` is used to minimize the memory used to store the Bitmap.

### Parameters

- **width** (`int`) – The number of values wide
- **height** (`int`) – The number of values high
- **value\_count** (`int`) – The number of possible pixel values.

**width :int**

Width of the bitmap. (read only)

**height :int**

Height of the bitmap. (read only)

**\_\_getitem\_\_** (*self, index: Union[Tuple[int, int], int]*) → `int`

Returns the value at the given index. The index can either be an x,y tuple or an int equal to `y * width + x`.

This allows you to:

```
print(bitmap[0,1])
```

**\_\_setitem\_\_** (*self*, *index*: Union[Tuple[int, int], int], *value*: int) → None

Sets the value at the given index. The index can either be an x,y tuple or an int equal to  $y * \text{width} + x$ .

This allows you to:

```
bitmap[0,1] = 3
```

**blit** (*self*, *x*: int, *y*: int, *source\_bitmap*: Bitmap, \*, *x1*: int, *y1*: int, *x2*: int, *y2*: int, *skip\_index*: int) → None

**Inserts the source\_bitmap region defined by rectangular boundaries** (x1,y1) and (x2,y2) into the bitmap at the specified (x,y) location.

#### Parameters

- **x** (int) – Horizontal pixel location in bitmap where source\_bitmap upper-left corner will be placed
- **y** (int) – Vertical pixel location in bitmap where source\_bitmap upper-left corner will be placed
- **source\_bitmap** (bitmap) – Source bitmap that contains the graphical region to be copied
- **x1** (int) – Minimum x-value for rectangular bounding box to be copied from the source bitmap
- **y1** (int) – Minimum y-value for rectangular bounding box to be copied from the source bitmap
- **x2** (int) – Maximum x-value (exclusive) for rectangular bounding box to be copied from the source bitmap
- **y2** (int) – Maximum y-value (exclusive) for rectangular bounding box to be copied from the source bitmap
- **skip\_index** (int) – bitmap palette index in the source that will not be copied, set to None to copy all pixels

**fill** (*self*, *value*: int) → None

Fills the bitmap with the supplied palette index value.

**class** displayio.ColorConverter (\*, *dither*: bool = False)

Converts one color format to another.

Create a ColorConverter object to convert color formats. Only supports RGB888 to RGB565 currently. :param bool dither: Adds random noise to dither the output image

**dither** :bool

When true the color converter dithers the output by adding random noise when truncating to display bit-depth

**convert** (*self*, *color*: int) → int

Converts the given RGB888 color to RGB565

**make\_transparent** (*self*, *pixel*: int) → None

Sets a pixel to not opaque.

**make\_opaque** (*self*, *pixel*: int) → None

Sets a pixel to opaque.

`displayio._DisplayBus`

*FourWire*, *ParallelBus* or *I2CDisplay*

```
class displayio.Display (display_bus: _DisplayBus, init_sequence: _typing.ReadableBuffer, *,
                        width: int, height: int, colstart: int = 0, rowstart: int = 0, rotation: int = 0,
                        color_depth: int = 16, grayscale: bool = False, pixels_in_byte_share_row:
                        bool = True, bytes_per_cell: int = 1, reverse_pixels_in_byte: bool
                        = False, set_column_command: int = 42, set_row_command: int =
                        43, write_ram_command: int = 44, set_vertical_scroll: int = 0, back-
                        light_pin: Optional[microcontroller.Pin] = None, brightness_command:
                        Optional[int] = None, brightness: float = 1.0, auto_brightness: bool =
                        False, single_byte_bounds: bool = False, data_as_commands: bool =
                        False, auto_refresh: bool = True, native_frames_per_second: int = 60)
```

Manage updating a display over a display bus

This initializes a display and connects it into CircuitPython. Unlike other objects in CircuitPython, Display objects live until `displayio.release_displays()` is called. This is done so that CircuitPython can use the display itself.

Most people should not use this class directly. Use a specific display driver instead that will contain the initialization sequence at minimum.

Create a Display object on the given display bus (*FourWire*, *ParallelBus* or *I2CDisplay*).

The `init_sequence` is bitpacked to minimize the ram impact. Every command begins with a command byte followed by a byte to determine the parameter count and delay. When the top bit of the second byte is 1 (0x80), a delay will occur after the command parameters are sent. The remaining 7 bits are the parameter count excluding any delay byte. The bytes following are the parameters. When the delay bit is set, a single byte after the parameters specifies the delay duration in milliseconds. The value 0xff will lead to an extra long 500 ms delay instead of 255 ms. The next byte will begin a new command definition. Here is an example:

```
init_sequence = (b"\xe1\x0f\x00\x0e\x14\x03\x11\x07\x31\xc1\x48\x08\x0f\x0c\x31\
↪x36\x0f" # Set Gamma
                b"\x11\x80\x78" # Exit Sleep then delay 0x78 (120ms)
                b"\x29\x81\xaa\x78" # Display on then delay 0x78 (120ms)
                )
display = displayio.Display(display_bus, init_sequence, width=320, height=240)
```

The first command is 0xe1 with 15 (0xf) parameters following. The second is 0x11 with 0 parameters and a 120ms (0x78) delay. The third command is 0x29 with one parameter 0xaa and a 120ms delay (0x78). Multiple byte literals (b'') are merged together on load. The parens are needed to allow byte literals on subsequent lines.

The initialization sequence should always leave the display memory access inline with the scan of the display to minimize tearing artifacts.

#### Parameters

- **display\_bus** – The bus that the display is connected to
- **init\_sequence** (`ReadableBuffer`) – Byte-packed initialization sequence.
- **width** (`int`) – Width in pixels
- **height** (`int`) – Height in pixels
- **colstart** (`int`) – The index if the first visible column
- **rowstart** (`int`) – The index if the first visible row

- **rotation** (`int`) – The rotation of the display in degrees clockwise. Must be in 90 degree increments (0, 90, 180, 270)
- **color\_depth** (`int`) – The number of bits of color per pixel transmitted. (Some displays support 18 bit but 16 is easier to transmit. The last bit is extrapolated.)
- **grayscale** (`bool`) – True if the display only shows a single color.
- **pixels\_in\_byte\_share\_row** (`bool`) – True when pixels are less than a byte and a byte includes pixels from the same row of the display. When False, pixels share a column.
- **bytes\_per\_cell** (`int`) – Number of bytes per addressable memory location when `color_depth < 8`. When greater than one, bytes share a row or column according to `pixels_in_byte_share_row`.
- **reverse\_pixels\_in\_byte** (`bool`) – Reverses the pixel order within each byte when `color_depth < 8`. Does not apply across multiple bytes even if there is more than one byte per cell (`bytes_per_cell`.)
- **reverse\_bytes\_in\_word** (`bool`) – Reverses the order of bytes within a word when `color_depth == 16`
- **set\_column\_command** (`int`) – Command used to set the start and end columns to update
- **set\_row\_command** (`int`) – Command used so set the start and end rows to update
- **write\_ram\_command** (`int`) – Command used to write pixels values into the update region. Ignored if `data_as_commands` is set.
- **set\_vertical\_scroll** (`int`) – Command used to set the first row to show
- **backlight\_pin** (`microcontroller.Pin`) – Pin connected to the display's backlight
- **brightness\_command** (`int`) – Command to set display brightness. Usually available in OLED controllers.
- **brightness** (`float`) – Initial display brightness. This value is ignored if `auto_brightness` is True.
- **auto\_brightness** (`bool`) – If True, brightness is controlled via an ambient light sensor or other mechanism.
- **single\_byte\_bounds** (`bool`) – Display column and row commands use single bytes
- **data\_as\_commands** (`bool`) – Treat all init and boundary data as SPI commands. Certain displays require this.
- **SH1107\_addressing** (`bool`) – Special quirk for SH1107, use upper/lower column set and page set
- **auto\_refresh** (`bool`) – Automatically refresh the screen
- **native\_frames\_per\_second** (`int`) – Number of display refreshes per second that occur with the given `init_sequence`.
- **backlight\_on\_high** (`bool`) – If True, pulling the backlight pin high turns the backlight on.

**auto\_refresh** :`bool`

True when the display is refreshed automatically.



**brightness :float**

The brightness of the display as a float. 0.0 is off and 1.0 is full brightness. When `auto_brightness` is True, the value of `brightness` will change automatically. If `brightness` is set, `auto_brightness` will be disabled and will be set to False.

**auto\_brightness :bool**

True when the display brightness is adjusted automatically, based on an ambient light sensor or other method. Note that some displays may have this set to True by default, but not actually implement automatic brightness adjustment. `auto_brightness` is set to False if `brightness` is set manually.

**width :int**

Gets the width of the board

**height :int**

Gets the height of the board

**rotation :int**

The rotation of the display as an int in degrees.

**bus :\_DisplayBus**

The bus being used by the display

**show** (*self*, *group*: Group) → None

Switches to displaying the given group of layers. When group is None, the default CircuitPython terminal will be shown.

**Parameters** *group* (Group) – The group to show.

**refresh** (*self*, \*, *target\_frames\_per\_second*: Optional[int] = None, *minimum\_frames\_per\_second*: int = 1) → bool

When auto refresh is off, waits for the target frame rate and then refreshes the display, returning True. If the call has taken too long since the last refresh call for the given target frame rate, then the refresh returns False immediately without updating the screen to hopefully help getting caught up.

If the time since the last successful refresh is below the minimum frame rate, then an exception will be raised. Set `minimum_frames_per_second` to 0 to disable.

When auto refresh is off, `display.refresh()` or `display.refresh(target_frames_per_second=None)` will update the display immediately.

When auto refresh is on, updates the display immediately. (The display will also update without calls to this.)

**Parameters**

- **target\_frames\_per\_second** (int) – How many times a second `refresh` should be called and the screen updated. Set to None for immediate refresh.
- **minimum\_frames\_per\_second** (int) – The minimum number of times the screen should be updated per second.

**fill\_row** (*self*, *y*: int, *buffer*: \_typing.WriteableBuffer) → \_typing.WriteableBuffer

Extract the pixels from a single row

**Parameters**

- **y** (int) – The top edge of the area
- **buffer** (WriteableBuffer) – The buffer in which to place the pixel data

```
class displayio.EPaperDisplay (display_bus: _DisplayBus, start_sequence: _typing.ReadableBuffer, stop_sequence: _typing.ReadableBuffer, *, width: int, height: int, ram_width: int, ram_height: int, colstart: int = 0, rowstart: int = 0, rotation: int = 0, set_column_window_command: Optional[int] = None, set_row_window_command: Optional[int] = None, single_byte_bounds: bool = False, write_black_ram_command: int, black_bits_inverted: bool = False, write_color_ram_command: Optional[int] = None, color_bits_inverted: bool = False, highlight_color: int = 0, refresh_display_command: int, refresh_time: float = 40, busy_pin: Optional[microcontroller.Pin] = None, busy_state: bool = True, seconds_per_frame: float = 180, always_toggle_chip_select: bool = False, grayscale: bool = False)
```

Manage updating an epaper display over a display bus

This initializes an epaper display and connects it into CircuitPython. Unlike other objects in CircuitPython, EPaperDisplay objects live until `displayio.release_displays()` is called. This is done so that CircuitPython can use the display itself.

Most people should not use this class directly. Use a specific display driver instead that will contain the startup and shutdown sequences at minimum.

Create a EPaperDisplay object on the given display bus (`displayio.FourWire` or `displayio.ParallelBus`).

The `start_sequence` and `stop_sequence` are bitpacked to minimize the ram impact. Every command begins with a command byte followed by a byte to determine the parameter count and delay. When the top bit of the second byte is 1 (0x80), a delay will occur after the command parameters are sent. The remaining 7 bits are the parameter count excluding any delay byte. The bytes following are the parameters. When the delay bit is set, a single byte after the parameters specifies the delay duration in milliseconds. The value 0xff will lead to an extra long 500 ms delay instead of 255 ms. The next byte will begin a new command definition.

#### Parameters

- **display\_bus** – The bus that the display is connected to
- **start\_sequence** (`ReadableBuffer`) – Byte-packed initialization sequence.
- **stop\_sequence** (`ReadableBuffer`) – Byte-packed initialization sequence.
- **width** (`int`) – Width in pixels
- **height** (`int`) – Height in pixels
- **ram\_width** (`int`) – RAM width in pixels
- **ram\_height** (`int`) – RAM height in pixels
- **colstart** (`int`) – The index if the first visible column
- **rowstart** (`int`) – The index if the first visible row
- **rotation** (`int`) – The rotation of the display in degrees clockwise. Must be in 90 degree increments (0, 90, 180, 270)
- **set\_column\_window\_command** (`int`) – Command used to set the start and end columns to update
- **set\_row\_window\_command** (`int`) – Command used so set the start and end rows to update

- **set\_current\_column\_command**(*int*) – Command used to set the current column location
- **set\_current\_row\_command**(*int*) – Command used to set the current row location
- **write\_black\_ram\_command**(*int*) – Command used to write pixels values into the update region
- **black\_bits\_inverted**(*bool*) – True if 0 bits are used to show black pixels. Otherwise, 1 means to show black.
- **write\_color\_ram\_command**(*int*) – Command used to write pixels values into the update region
- **color\_bits\_inverted**(*bool*) – True if 0 bits are used to show the color. Otherwise, 1 means to show color.
- **highlight\_color**(*int*) – RGB888 of source color to highlight with third ePaper color.
- **refresh\_display\_command**(*int*) – Command used to start a display refresh
- **refresh\_time**(*float*) – Time it takes to refresh the display before the stop\_sequence should be sent. Ignored when busy\_pin is provided.
- **busy\_pin**(*microcontroller.Pin*) – Pin used to signify the display is busy
- **busy\_state**(*bool*) – State of the busy pin when the display is busy
- **seconds\_per\_frame**(*float*) – Minimum number of seconds between screen refreshes
- **always\_toggle\_chip\_select**(*bool*) – When True, chip select is toggled every byte
- **grayscale**(*bool*) – When true, the color ram is the low bit of 2-bit grayscale

**time\_to\_refresh** :float

Time, in fractional seconds, until the ePaper display can be refreshed.

**busy** :bool

True when the display is refreshing. This uses the busy\_pin when available or the refresh\_time otherwise.

**width** :int

Gets the width of the display in pixels

**height** :int

Gets the height of the display in pixels

**rotation** :int

The rotation of the display as an int in degrees.

**bus** :\_DisplayBus

The bus being used by the display

**show**(*self*, *group*: [Group](#)) → None

Switches to displaying the given group of layers. When group is None, the default CircuitPython terminal will be shown.

**Parameters** *group* ([Group](#)) – The group to show.

**refresh**(*self*) → None

Refreshes the display immediately or raises an exception if too soon. Use `time.sleep(display.time_to_refresh)` to sleep until a refresh can occur.

```
class displayio.FourWire (spi_bus: busio.SPI, *, command: microcontroller.Pin, chip_select: micro-
                           controller.Pin, reset: Optional[microcontroller.Pin] = None, baudrate: int
                           = 24000000, polarity: int = 0, phase: int = 0)
```

Manage updating a display over SPI four wire protocol in the background while Python code runs. It doesn't handle display initialization.

Create a FourWire object associated with the given pins.

The SPI bus and pins are then in use by the display until `displayio.release_displays()` is called even after a reload. (It does this so CircuitPython can use the display after your code is done.) So, the first time you initialize a display bus in code.py you should call `:py:func`displayio.release_displays`` first, otherwise it will error after the first code.py run.

#### Parameters

- **spi\_bus** (`busio.SPI`) – The SPI bus that make up the clock and data lines
- **command** (`microcontroller.Pin`) – Data or command pin
- **chip\_select** (`microcontroller.Pin`) – Chip select pin
- **reset** (`microcontroller.Pin`) – Reset pin. When None only software reset can be used
- **baudrate** (`int`) – Maximum baudrate in Hz for the display on the bus
- **polarity** (`int`) – the base state of the clock line (0 or 1)
- **phase** (`int`) – the edge of the clock that data is captured. First (0) or second (1). Rising or falling depends on clock polarity.

**reset** (*self*) → `None`

Performs a hardware reset via the reset pin. Raises an exception if called when no reset pin is available.

**send** (*self*, *command*: `int`, *data*: `FourWire`, \*, *toggle\_every\_byte*: `bool` = `False`) → `None`

Sends the given command value followed by the full set of data. Display state, such as vertical scroll, set via `send` may or may not be reset once the code is done.

```
class displayio.Group (*, max_size: int = 4, scale: int = 1, x: int = 0, y: int = 0)
```

Manage a group of sprites and groups and how they are inter-related.

Create a Group of a given size and scale. Scale is in one dimension. For example, `scale=2` leads to a layer's pixel being 2x2 pixels when in the group.

#### Parameters

- **max\_size** (`int`) – Ignored. Will be removed in 7.x.
- **scale** (`int`) – Scale of layer pixels in one dimension.
- **x** (`int`) – Initial x position within the parent.
- **y** (`int`) – Initial y position within the parent.

**hidden** :`bool`

True when the Group and all of it's layers are not visible. When False, the Group's layers are visible if they haven't been hidden.

**scale** :`int`

Scales each pixel within the Group in both directions. For example, when `scale=2` each pixel will be represented by 2x2 pixels.

**x** :`int`

X position of the Group in the parent.

**y** :int

Y position of the Group in the parent.

**append** (*self*, *layer*: Union[vectorio.VectorShape, Group, TileGrid]) → None

Append a layer to the group. It will be drawn above other layers.

**insert** (*self*, *index*: int, *layer*: Union[vectorio.VectorShape, Group, TileGrid]) → None

Insert a layer into the group.

**index** (*self*, *layer*: Union[vectorio.VectorShape, Group, TileGrid]) → int

Returns the index of the first copy of layer. Raises ValueError if not found.

**pop** (*self*, *i*: int = -1) → Union[vectorio.VectorShape, Group, TileGrid]

Remove the *i*th item and return it.

**remove** (*self*, *layer*: Union[vectorio.VectorShape, Group, TileGrid]) → None

Remove the first copy of layer. Raises ValueError if it is not present.

**\_\_bool\_\_** (*self*) → bool

**\_\_len\_\_** (*self*) → int

Returns the number of layers in a Group

**\_\_getitem\_\_** (*self*, *index*: int) → Union[vectorio.VectorShape, Group, TileGrid]

Returns the value at the given index.

This allows you to:

```
print(group[0])
```

**\_\_setitem\_\_** (*self*, *index*: int, *value*: Union[vectorio.VectorShape, Group, TileGrid]) → None

Sets the value at the given index.

This allows you to:

```
group[0] = sprite
```

**\_\_delitem\_\_** (*self*, *index*: int) → None

Deletes the value at the given index.

This allows you to:

```
del group[0]
```

**sort** (*self*, *key*: function, *reverse*: bool) → None

Sort the members of the group.

**class** displayio.I2CDisplay (*i2c\_bus*: busio.I2C, \*, *device\_address*: int, *reset*: Optional[microcontroller.Pin] = None)

Manage updating a display over I2C in the background while Python code runs. It doesn't handle display initialization.

Create a I2CDisplay object associated with the given I2C bus and reset pin.

The I2C bus and pins are then in use by the display until `displayio.release_displays()` is called even after a reload. (It does this so CircuitPython can use the display after your code is done.) So, the first time you initialize a display bus in code.py you should call `py:func`displayio.release_displays`` first, otherwise it will error after the first code.py run.

#### Parameters

- **i2c\_bus** (busio.I2C) – The I2C bus that make up the clock and data lines

- **device\_address** (*int*) – The I2C address of the device
- **reset** (*microcontroller.Pin*) – Reset pin. When None only software reset can be used

**reset** (*self*) → *None*

Performs a hardware reset via the reset pin. Raises an exception if called when no reset pin is available.

**send** (*self*, *command*: *int*, *data*: *\_typing.ReadableBuffer*) → *None*

Sends the given command value followed by the full set of data. Display state, such as vertical scroll, set via send may or may not be reset once the code is done.

**class** `displayio.OnDiskBitmap` (*file*: *BinaryIO*)

Loads values straight from disk. This minimizes memory use but can lead to much slower pixel load times. These load times may result in frame tearing where only part of the image is visible.

It's easiest to use on a board with a built in display such as the [Hallowing M0 Express](#).

```
import board
import displayio
import time
import pulseio

board.DISPLAY.auto_brightness = False
board.DISPLAY.brightness = 0
splash = displayio.Group()
board.DISPLAY.show(splash)

with open("/sample.bmp", "rb") as f:
    odb = displayio.OnDiskBitmap(f)
    face = displayio.TileGrid(odb, pixel_shader=displayio.ColorConverter())
    splash.append(face)
    # Wait for the image to load.
    board.DISPLAY.refresh(target_frames_per_second=60)

    # Fade up the backlight
    for i in range(100):
        board.DISPLAY.brightness = 0.01 * i
        time.sleep(0.05)

    # Wait forever
    while True:
        pass
```

Create an `OnDiskBitmap` object with the given file.

**Parameters** **file** (*file*) – The open bitmap file

**width** :*int*

Width of the bitmap. (read only)

**height** :*int*

Height of the bitmap. (read only)

**class** `displayio.Palette` (*color\_count*: *int*)

Map a pixel `palette_index` to a full color. Colors are transformed to the display's format internally to save memory.

Create a `Palette` object to store a set number of colors.

**Parameters** **color\_count** (*int*) – The number of colors in the `Palette`

`__bool__(self) → bool`

`__len__(self) → int`

Returns the number of colors in a Palette

`__getitem__(self, index: int) → Optional[int]`

Return the pixel color at the given index as an integer.

`__setitem__(self, index: int, value: Union[int, typing.ReadableBuffer, Tuple[int, int, int]]) → None`

Sets the pixel color at the given index. The index should be an integer in the range 0 to color\_count-1.

The value argument represents a color, and can be from 0x000000 to 0xFFFFFF (to represent an RGB value). Value can be an int, bytes (3 bytes (RGB) or 4 bytes (RGB + pad byte)), bytearray, or a tuple or list of 3 integers.

This allows you to:

```
palette[0] = 0xFFFFFF          # set using an integer
palette[1] = b'\xff\xff\x00'   # set using 3 bytes
palette[2] = b'\xff\xff\x00\x00' # set using 4 bytes
palette[3] = bytearray(b'\x00\x00\xff') # set using a bytearray of 3 or 4
↳ bytes
palette[4] = (10, 20, 30)      # set using a tuple of 3 integers
```

`make_transparent(self, palette_index: int) → None`

`make_opaque(self, palette_index: int) → None`

`is_transparent(self, palette_index: int) → bool`

Returns `True` if the palette index is transparent. Returns `False` if opaque.

`class displayio.ParallelBus(*, data0: microcontroller.Pin, command: microcontroller.Pin, chip_select: microcontroller.Pin, write: microcontroller.Pin, read: microcontroller.Pin, reset: microcontroller.Pin)`

Manage updating a display over 8-bit parallel bus in the background while Python code runs. This protocol may be referred to as 8080-I Series Parallel Interface in datasheets. It doesn't handle display initialization.

Create a ParallelBus object associated with the given pins. The bus is inferred from data0 by implying the next 7 additional pins on a given GPIO port.

The parallel bus and pins are then in use by the display until `displayio.release_displays()` is called even after a reload. (It does this so CircuitPython can use the display after your code is done.) So, the first time you initialize a display bus in code.py you should call `:py:func`displayio.release_displays`` first, otherwise it will error after the first code.py run.

#### Parameters

- **data0** (`microcontroller.Pin`) – The first data pin. The rest are implied
- **command** (`microcontroller.Pin`) – Data or command pin
- **chip\_select** (`microcontroller.Pin`) – Chip select pin
- **write** (`microcontroller.Pin`) – Write pin
- **read** (`microcontroller.Pin`) – Read pin
- **reset** (`microcontroller.Pin`) – Reset pin

`reset(self) → None`

Performs a hardware reset via the reset pin. Raises an exception if called when no reset pin is available.

**send** (*self*, *command*: `int`, *data*: `_typing.ReadableBuffer`) → `None`

Sends the given command value followed by the full set of data. Display state, such as vertical scroll, set via `send` may or may not be reset once the code is done.

**class** `displayio.Shape` (*width*: `int`, *height*: `int`, \*, *mirror\_x*: `bool` = `False`, *mirror\_y*: `bool` = `False`)

Represents a shape made by defining boundaries that may be mirrored.

Create a `Shape` object with the given fixed size. Each pixel is one bit and is stored by the column boundaries of the shape on each row. Each row's boundary defaults to the full row.

#### Parameters

- **width** (`int`) – The number of pixels wide
- **height** (`int`) – The number of pixels high
- **mirror\_x** (`bool`) – When true the left boundary is mirrored to the right.
- **mirror\_y** (`bool`) – When true the top boundary is mirrored to the bottom.

**set\_boundary** (*self*, *y*: `int`, *start\_x*: `int`, *end\_x*: `int`) → `None`

Loads pre-packed data into the given row.

**class** `displayio.TileGrid` (*bitmap*: `Bitmap`, \*, *pixel\_shader*: `Union[ColorConverter, Palette]`, *width*: `int` = `1`, *height*: `int` = `1`, *tile\_width*: `Optional[int]` = `None`, *tile\_height*: `Optional[int]` = `None`, *default\_tile*: `int` = `0`, *x*: `int` = `0`, *y*: `int` = `0`)

A grid of tiles sourced out of one `bitmap`

Position a grid of tiles sourced from a `bitmap` and `pixel_shader` combination. Multiple grids can share `bitmaps` and `pixel shaders`.

A single tile grid is also known as a `Sprite`.

Create a `TileGrid` object. The `bitmap` is source for 2d pixels. The `pixel_shader` is used to convert the value and its location to a display native pixel color. This may be a simple color palette lookup, a gradient, a pattern or a color transformer.

`tile_width` and `tile_height` match the height of the `bitmap` by default.

#### Parameters

- **bitmap** (`Bitmap`) – The `bitmap` storing one or more tiles.
- **or Palette pixel\_shader** (`ColorConverter`) – The `pixel shader` that produces colors from values
- **width** (`int`) – Width of the grid in tiles.
- **height** (`int`) – Height of the grid in tiles.
- **tile\_width** (`int`) – Width of a single tile in pixels. Defaults to the full `Bitmap` and must evenly divide into the `Bitmap`'s dimensions.
- **tile\_height** (`int`) – Height of a single tile in pixels. Defaults to the full `Bitmap` and must evenly divide into the `Bitmap`'s dimensions.
- **default\_tile** (`int`) – Default tile index to show.
- **x** (`int`) – Initial x position of the left edge within the parent.
- **y** (`int`) – Initial y position of the top edge within the parent.

**hidden** :`bool`

True when the `TileGrid` is hidden. This may be `False` even when a part of a hidden `Group`.

**x** :`int`

X position of the left edge in the parent.



**y :int**

Y position of the top edge in the parent.

**flip\_x :bool**

If true, the left edge rendered will be the right edge of the right-most tile.

**flip\_y :bool**

If true, the top edge rendered will be the bottom edge of the bottom-most tile.

**transpose\_xy :bool**

If true, the TileGrid's axis will be swapped. When combined with mirroring, any 90 degree rotation can be achieved along with the corresponding mirrored version.

**pixel\_shader :Union[ColorConverter, Palette]**

The pixel shader of the tilegrid.

**\_\_getitem\_\_** (*self*, *index: Union[Tuple[int, int], int]*) → *int*

Returns the tile index at the given index. The index can either be an x,y tuple or an int equal to  $y * \text{width} + x$ .

This allows you to:

```
print(grid[0])
```

**\_\_setitem\_\_** (*self*, *index: Union[Tuple[int, int], int]*, *value: int*) → *None*

Sets the tile index at the given index. The index can either be an x,y tuple or an int equal to  $y * \text{width} + x$ .

This allows you to:

```
grid[0] = 10
```

or:

```
grid[0,0] = 10
```

## dualbank – DUALBANK Module

The *dualbank* module adds ability to update and switch between the two app partitions.

There are two identical partitions, these contain different firmware versions. Having two partitions enables rollback functionality.

The two partitions are defined as boot partition and next-update partition. Calling *dualbank.flash()* writes the next-update partition.

After the next-update partition is written a validation check is performed and on a successful validation this partition is set as the boot partition. On next reset, firmware will be loaded from this partition.

Here is the sequence of commands to follow:

```
import dualbank

dualbank.flash(buffer, offset)
dualbank.switch()
```

**dualbank.flash** (*\*buffer: \_typing.ReadableBuffer*, *offset: int = 0*) → *None*

Writes one of two app partitions at the given offset.

This can be called multiple times when flashing the firmware in small chunks.

`dualbank.switch()` → `None`

Switches the boot partition.

On next reset, firmware will be loaded from the partition just switched over to.

## fontio – Core font related data structures

**class** `fontio.BuiltinFont`

A font built into CircuitPython

Creation not supported. Available fonts are defined when CircuitPython is built. See the [Adafruit\\_CircuitPython\\_Bitmap\\_Font](#) library for dynamically loaded fonts.

**bitmap** : `displayio.Bitmap`

Bitmap containing all font glyphs starting with ASCII and followed by unicode. Use `get_glyph` in most cases. This is useful for use with `displayio.TileGrid` and `terminalio.Terminal`.

**get\_bounding\_box** (*self*) → `Tuple[int, int]`

Returns the maximum bounds of all glyphs in the font in a tuple of two values: width, height.

**get\_glyph** (*self*, *codepoint*: `int`) → `Glyph`

Returns a `fontio.Glyph` for the given codepoint or `None` if no glyph is available.

**class** `fontio.Glyph` (*bitmap*: `displayio.Bitmap`, *tile\_index*: `int`, *width*: `int`, *height*: `int`, *dx*: `int`, *dy*: `int`, *shift\_x*: `int`, *shift\_y*: `int`)

Storage of glyph info

Named tuple used to capture a single glyph and its attributes.

### Parameters

- **bitmap** – the bitmap including the glyph
- **tile\_index** – the tile index within the bitmap
- **width** – the width of the glyph’s bitmap
- **height** – the height of the glyph’s bitmap
- **dx** – x adjustment to the bitmap’s position
- **dy** – y adjustment to the bitmap’s position
- **shift\_x** – the x difference to the next glyph
- **shift\_y** – the y difference to the next glyph

## framebufferio – Native framebuffer display driving

The `framebufferio` module contains classes to manage display output including synchronizing with refresh rates and partial updating. It is used in conjunction with classes from `displayio` to actually place items on the display; and classes like `RGBMatrix` to actually drive the display.

**class** `framebufferio.FramebufferDisplay` (*framebuffer*: `typing.FrameBuffer`, \*, *rotation*: `int` = `0`, *auto\_refresh*: `bool` = `True`)

Manage updating a display with framebuffer in RAM

This initializes a display and connects it into CircuitPython. Unlike other objects in CircuitPython, Display objects live until `displayio.release_displays()` is called. This is done so that CircuitPython can use the display itself.

Create a Display object with the given framebuffer (a buffer, array, `ulab.array`, etc)

**Parameters**

- **framebuffer** (`FrameBuffer`) – The framebuffer that the display is connected to
- **auto\_refresh** (`bool`) – Automatically refresh the screen
- **rotation** (`int`) – The rotation of the display in degrees clockwise. Must be in 90 degree increments (0, 90, 180, 270)

**auto\_refresh** : `bool`

True when the display is refreshed automatically.

**brightness** : `float`

The brightness of the display as a float. 0.0 is off and 1.0 is full brightness. When `auto_brightness` is True, the value of `brightness` will change automatically. If `brightness` is set, `auto_brightness` will be disabled and will be set to False.

**auto\_brightness** : `bool`

True when the display brightness is adjusted automatically, based on an ambient light sensor or other method. Note that some displays may have this set to True by default, but not actually implement automatic brightness adjustment. `auto_brightness` is set to False if `brightness` is set manually.

**width** : `int`

Gets the width of the framebuffer

**height** : `int`

Gets the height of the framebuffer

**rotation** : `int`

The rotation of the display as an int in degrees.

**framebuffer** : `_typing.FrameBuffer`

The framebuffer being used by the display

**show** (*self*, *group*: `displayio.Group`) → `None`

Switches to displaying the given group of layers. When group is None, the default CircuitPython terminal will be shown.

**Parameters** *group* (`Group`) – The group to show.

**refresh** (*self*, \*, *target\_frames\_per\_second*: `int` = 60, *minimum\_frames\_per\_second*: `int` = 1) → `bool`

When auto refresh is off, waits for the target frame rate and then refreshes the display, returning True. If the call has taken too long since the last refresh call for the given target frame rate, then the refresh returns False immediately without updating the screen to hopefully help getting caught up.

If the time since the last successful refresh is below the minimum frame rate, then an exception will be raised. Set `minimum_frames_per_second` to 0 to disable.

When auto refresh is on, updates the display immediately. (The display will also update without calls to this.)

**Parameters**

- **target\_frames\_per\_second** (`int`) – How many times a second `refresh` should be called and the screen updated.
- **minimum\_frames\_per\_second** (`int`) – The minimum number of times the screen should be updated per second.

**fill\_row** (*self*, *y*: `int`, *buffer*: `_typing.WritableBuffer`) → `_typing.WritableBuffer`

Extract the pixels from a single row

**Parameters**

- **y** (`int`) – The top edge of the area
- **buffer** (`WriteableBuffer`) – The buffer in which to place the pixel data

## frequencyio – Support for frequency based protocols

**Warning:** This module is not available in SAMD21 builds. See the [Module Support Matrix - Which Modules Are Available on Which Boards](#) for more info.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See [Lifetime and ContextManagers](#) for more info.

For example:

```
import frequencyio
import time
from board import *

frequency = frequencyio.FrequencyIn(D13)
frequency.capture_period = 15
time.sleep(0.1)
```

This example will initialize the the device, set `capture_period`, and then sleep 0.1 seconds. CircuitPython will automatically turn off `FrequencyIn` capture when it resets all hardware after program completion. Use `deinit()` or a `with` statement to do it yourself.

**class** `frequencyio.FrequencyIn` (*pin*: `microcontroller.Pin`, *capture\_period*: `int` = 10)

Read a frequency signal

`FrequencyIn` is used to measure the frequency, in hertz, of a digital signal on an incoming pin. Accuracy has shown to be within 10%, if not better. It is recommended to utilize an average of multiple samples to smooth out readings.

Frequencies below 1KHz are not currently detectable.

`FrequencyIn` will not determine pulse width (use `PulseIn`).

Create a `FrequencyIn` object associated with the given pin.

### Parameters

- **pin** (`Pin`) – Pin to read frequency from.
- **capture\_period** (`int`) – Keyword argument to set the measurement period, in milliseconds. Default is 10ms; range is 1ms - 500ms.

Read the incoming frequency from a pin:

```
import frequencyio
import board

frequency = frequencyio.FrequencyIn(board.D11)

# Loop while printing the detected frequency
while True:
    print(frequency.value)
```

(continues on next page)

(continued from previous page)

```
# Optional clear() will reset the value
# to zero. Without this, if the incoming
# signal stops, the last reading will remain
# as the value.
frequency.clear()
```

**capture\_period :int**

The capture measurement period. Lower incoming frequencies will be measured more accurately with longer capture periods. Higher frequencies are more accurate with shorter capture periods.

---

**Note:** When setting a new `capture_period`, all previous capture information is cleared with a call to `clear()`.

---

**deinit** (*self*) → *None*

Deinitialises the `FrequencyIn` and releases any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → *FrequencyIn*

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → *None*

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**pause** (*self*) → *None*

Pause frequency capture.

**resume** (*self*) → *None*

Resumes frequency capture.

**clear** (*self*) → *None*

Clears the last detected frequency capture value.

**\_\_get\_\_** (*self*, *index*: int) → int

Returns the value of the last frequency captured.

**gamepad – Button handling in the background**

**class** `gamepad.GamePad` (*b1*: `digitalio.DigitalInOut`, *b2*: `digitalio.DigitalInOut`, *b3*: `digitalio.DigitalInOut`, *b4*: `digitalio.DigitalInOut`, *b5*: `digitalio.DigitalInOut`, *b6*: `digitalio.DigitalInOut`, *b7*: `digitalio.DigitalInOut`, *b8*: `digitalio.DigitalInOut`)

Scan buttons for presses

Usage:

```
import board
import digitalio
import gamepad
import time

B_UP = 1 << 0
B_DOWN = 1 << 1

pad = gamepad.GamePad(
    digitalio.DigitalInOut(board.D10),
```

(continues on next page)

(continued from previous page)

```

        digitalio.DigitalInOut(board.D11),
    )

y = 0
while True:
    buttons = pad.get_pressed()
    if buttons & B_UP:
        y -= 1
        print(y)
    elif buttons & B_DOWN:
        y += 1
        print(y)
    time.sleep(0.1)
    while buttons:
        # Wait for all buttons to be released.
        buttons = pad.get_pressed()
        time.sleep(0.1)

```

Initializes button scanning routines.

The `b1-b8` parameters are `DigitalInOut` objects, which immediately get switched to input with a pull-up, (unless they already were set to pull-down, in which case they remain so), and then scanned regularly for button presses. The order is the same as the order of bits returned by the `get_pressed` function. You can re-initialize it with different keys, then the new object will replace the previous one.

The basic feature required here is the ability to poll the keys at regular intervals (so that de-bouncing is consistent) and fast enough (so that we don't miss short button presses) while at the same time letting the user code run normally, call blocking functions and wait on delays.

They button presses are accumulated, until the `get_pressed` method is called, at which point the button state is cleared, and the new button presses start to be recorded.

**`get_pressed(self) → int`**

Get the status of buttons pressed since the last call and clear it.

Returns an 8-bit number, with bits that correspond to buttons, which have been pressed (or held down) since the last call to this function set to 1, and the remaining bits set to 0. Then it clears the button state, so that new button presses (or buttons that are held down) can be recorded for the next call.

**`deinit(self) → None`**

Disable button scanning.

### gamepadshift – Tracks button presses read through a shift register

**`class gamepadshift.GamePadShift`** (*clock:* `digitalio.DigitalInOut`, *data:* `digitalio.DigitalInOut`,  
*latch:* `digitalio.DigitalInOut`)

Scan buttons for presses through a shift register

Initializes button scanning routines.

The `clock`, `data` and `latch` parameters are `DigitalInOut` objects connected to the shift register controlling the buttons.

They button presses are accumulated, until the `get_pressed` method is called, at which point the button state is cleared, and the new button presses start to be recorded.

Only one gamepad (`gamepad.GamePad` or `gamepadshift.GamePadShift`) may be used at a time.

**get\_pressed**(*self*) → *int*

Get the status of buttons pressed since the last call and clear it.

Returns an 8-bit number, with bits that correspond to buttons, which have been pressed (or held down) since the last call to this function set to 1, and the remaining bits set to 0. Then it clears the button state, so that new button presses (or buttons that are held down) can be recorded for the next call.

**deinit**(*self*) → *None*

Disable button scanning.

## gnss – Global Navigation Satellite System

The *gnss* module contains classes to control the GNSS and acquire positioning information.

**class** *gnss.GNSS*(*system*: Union[*SatelliteSystem*, List[*SatelliteSystem*]])

Get updated positioning information from Global Navigation Satellite System (GNSS)

Usage:

```
import gnss
import time

nav = gnss.GNSS([gnss.SatelliteSystem.GPS, gnss.SatelliteSystem.GLONASS])
last_print = time.monotonic()
while True:
    nav.update()
    current = time.monotonic()
    if current - last_print >= 1.0:
        last_print = current
        if nav.fix is gnss.PositionFix.INVALID:
            print("Waiting for fix...")
            continue
        print("Latitude: {0:.6f} degrees".format(nav.latitude))
        print("Longitude: {0:.6f} degrees".format(nav.longitude))
```

Turn on the GNSS.

**Parameters** *system* – satellite system to use

**latitude** :float

Latitude of current position in degrees (float).

**longitude** :float

Longitude of current position in degrees (float).

**altitude** :float

Altitude of current position in meters (float).

**timestamp** :time.struct\_time

Time when the position data was updated.

**fix** :PositionFix

Fix mode.

**deinit**(*self*) → *None*

Turn off the GNSS.

**update**(*self*) → *None*

Update GNSS positioning information.

```
class gnss.PositionFix
    Position fix mode

    Enum-like class to define the position fix mode.

    INVALID :PositionFix
        No measurement.

    FIX_2D :PositionFix
        2D fix.

    FIX_3D :PositionFix
        3D fix.

class gnss.SatelliteSystem
    Satellite system type

    Enum-like class to define the satellite system type.

    GPS :SatelliteSystem
        Global Positioning System.

    GLONASS :SatelliteSystem
        GLObal NAVigation Satellite System.

    SBAS :SatelliteSystem
        Satellite Based Augmentation System.

    QZSS_L1CA :SatelliteSystem
        Quasi-Zenith Satellite System L1C/A.

    QZSS_L1S :SatelliteSystem
        Quasi-Zenith Satellite System L1S.
```

## i2cperipheral – Two wire serial protocol peripheral

The *i2cperipheral* module contains classes to support an I2C peripheral.

Example emulating a peripheral with 2 addresses (read and write):

```
import board
from i2cperipheral import I2CPeripheral

regs = [0] * 16
index = 0

with I2CPeripheral(board.SCL, board.SDA, (0x40, 0x41)) as device:
    while True:
        r = device.request()
        if not r:
            # Maybe do some housekeeping
            continue
        with r: # Closes the transfer if necessary by sending a NACK or feeding_
            ↪ dummy bytes
            if r.address == 0x40:
                if not r.is_read: # Main write which is Selected read
                    b = r.read(1)
                    if not b or b[0] > 15:
                        break
                    index = b[0]
```

(continues on next page)



(continued from previous page)

```

        b = r.read(1)
        if b:
            regs[index] = b[0]
        elif r.is_restart: # Combined transfer: This is the Main read message
            n = r.write(bytes([regs[index]]))
        #else:
        # A read transfer is not supported in this example
        # If the microcontroller tries, it will get 0xff byte(s) by the_
→ctx manager (r.close())
        elif r.address == 0x41:
            if not r.is_read:
                b = r.read(1)
                if b and b[0] == 0xde:
                    # do something
                pass

```

This example sets up an I2C device that can be accessed from Linux like this:

```

$ i2cget -y 1 0x40 0x01
0x00
$ i2cset -y 1 0x40 0x01 0xaa
$ i2cget -y 1 0x40 0x01
0xaa

```

**Warning:** I2CPeripheral makes use of clock stretching in order to slow down the host. Make sure the I2C host supports this.

Raspberry Pi in particular does not support this with its I2C hw block. This can be worked around by using the `i2c-gpio` bit banging driver. Since the RPi firmware uses the hw i2c, it's not possible to emulate a HAT eeprom.

**class** `i2cperipheral.I2CPeripheral` (*scl*: `microcontroller.Pin`, *sda*: `microcontroller.Pin`, *addresses*: `Sequence[int]`, *smbus*: `bool = False`)

Two wire serial protocol peripheral

I2C is a two-wire protocol for communicating between devices. This implements the peripheral (sensor, secondary) side.

#### Parameters

- **scl** (`Pin`) – The clock pin
- **sda** (`Pin`) – The data pin
- **addresses** (`list[int]`) – The I2C addresses to respond to (how many is hw dependent).
- **smbus** (`bool`) – Use SMBUS timings if the hardware supports it

**deinit** (*self*) → `None`

Releases control of the underlying hardware so other classes can use it.

**\_\_enter\_\_** (*self*) → `I2CPeripheral`

No-op used in Context Managers.

**\_\_exit\_\_** (*self*) → `None`

Automatically deinitializes the hardware on context exit. See *Lifetime and ContextManagers* for more info.

**request** (*self*, *timeout*: float = - 1) → *I2CPeripheralRequest*

Wait for an I2C request.

**Parameters** **timeout** (float) – Timeout in seconds. Zero means wait forever, a negative value means check once

**Returns** I2C Slave Request or None if timeout=-1 and there's no request

**Return type** *I2CPeripheralRequest*

**class** `i2cperipheral.I2CPeripheralRequest` (*peripheral*: *I2CPeripheral*, *address*: int, *is\_read*: bool, *is\_restart*: bool)

Information about an I2C transfer request This cannot be instantiated directly, but is returned by *I2CPeripheral.request()*.

**Parameters**

- **peripheral** – The I2CPeripheral object receiving this request
- **address** – I2C address
- **is\_read** – True if the main peripheral is requesting data
- **is\_restart** – Repeated Start Condition

**address** :int

The I2C address of the request.

**is\_read** :bool

The I2C main controller is reading from this peripheral.

**is\_restart** :bool

Is Repeated Start Condition.

**\_\_enter\_\_** (*self*) → *I2CPeripheralRequest*

No-op used in Context Managers.

**\_\_exit\_\_** (*self*) → None

Close the request.

**read** (*self*, *n*: int = - 1, *ack*: bool = True) → bytearray

Read data. If ack=False, the caller is responsible for calling *I2CPeripheralRequest.ack()*.

**Parameters**

- **n** – Number of bytes to read (negative means all)
- **ack** – Whether or not to send an ACK after the n'th byte

**Returns** Bytes read

**write** (*self*, *buffer*: *\_typing.ReadableBuffer*) → int

Write the data contained in buffer.

**Parameters** **buffer** (*ReadableBuffer*) – Write out the data in this buffer

**Returns** Number of bytes written

**ack** (*self*, *ack*: bool = True) → None

Acknowledge or Not Acknowledge last byte received. Use together with *I2CPeripheralRequest.read()* *ack=False*.

**Parameters** **ack** – Whether to send an ACK or NACK

## ipaddress

The `ipaddress` module provides types for IP addresses. It is a subset of CPython's `ipaddress` module.

`ipaddress.ip_address(obj: Union[int]) → IPv4Address`

Return a corresponding IP address object or raise `ValueError` if not possible.

**class** `ipaddress.IPv4Address(address: Union[int, str, bytes])`

Encapsulates an IPv4 address.

Create a new `IPv4Address` object encapsulating the address value.

The value itself can either be bytes or a string formatted address.

**packed :bytes**

The bytes that make up the address (read-only).

**version :int**

4 for IPv4, 6 for IPv6

`__eq__ (self, other: object) → bool`

Two Address objects are equal if their addresses and address types are equal.

`__hash__ (self) → int`

Returns a hash for the `IPv4Address` data.

## math – mathematical functions

The `math` module provides some basic mathematical functions for working with floating-point numbers.

**math.e :float**

base of the natural logarithm

**math.pi :float**

the ratio of a circle's circumference to its diameter

`math.acos(x: float) → float`

Return the inverse cosine of  $x$ .

`math.asin(x: float) → float`

Return the inverse sine of  $x$ .

`math.atan(x: float) → float`

Return the inverse tangent of  $x$ .

`math.atan2(y: float, x: float) → float`

Return the principal value of the inverse tangent of  $y/x$ .

`math.ceil(x: float) → int`

Return an integer, being  $x$  rounded towards positive infinity.

`math.copysign(x: float, y: float) → float`

Return  $x$  with the sign of  $y$ .

`math.cos(x: float) → float`

Return the cosine of  $x$ .

`math.degrees(x: float) → float`

Return radians  $x$  converted to degrees.

`math.exp(x: float) → float`

Return the exponential of  $x$ .

`math.fabs(x: float) → float`

Return the absolute value of `x`.

`math.floor(x: float) → int`

Return an integer, being `x` rounded towards negative infinity.

`math.fmod(x: float, y: float) → int`

Return the remainder of `x/y`.

`math.frexp(x: float) → Tuple[int, int]`

Decomposes a floating-point number into its mantissa and exponent. The returned value is the tuple `(m, e)` such that `x == m * 2**e` exactly. If `x == 0` then the function returns `(0.0, 0)`, otherwise the relation `0.5 <= abs(m) < 1` holds.

`math.isfinite(x: float) → bool`

Return `True` if `x` is finite.

`math.isinf(x: float) → bool`

Return `True` if `x` is infinite.

`math.isnan(x: float) → bool`

Return `True` if `x` is not-a-number

`math.ldexp(x: float, exp: float) → float`

Return `x * (2**exp)`.

`math.modf(x: float) → Tuple[float, float]`

Return a tuple of two floats, being the fractional and integral parts of `x`. Both return values have the same sign as `x`.

`math.pow(x: float, y: float) → float`

Returns `x` to the power of `y`.

`math.radians(x: float) → float`

Return degrees `x` converted to radians.

`math.sin(x: float) → float`

Return the sine of `x`.

`math.sqrt(x: float) → float`

Returns the square root of `x`.

`math.tan(x: float) → float`

Return the tangent of `x`.

`math.trunc(x: float) → int`

Return an integer, being `x` rounded towards 0.

`math.expm1(x: float) → float`

Return `exp(x) - 1`.

`math.log2(x: float) → float`

Return the base-2 logarithm of `x`.

`math.log10(x: float) → float`

Return the base-10 logarithm of `x`.

`math.cosh(x: float) → float`

Return the hyperbolic cosine of `x`.

`math.sinh(x: float) → float`

Return the hyperbolic sine of `x`.

`math.tanh(x: float) → float`

Return the hyperbolic tangent of `x`.

`math.acosh(x: float) → float`

Return the inverse hyperbolic cosine of `x`.

`math.asinh(x: float) → float`

Return the inverse hyperbolic sine of `x`.

`math.atanh(x: float) → float`

Return the inverse hyperbolic tangent of `x`.

`math.erf(x: float) → float`

Return the error function of `x`.

`math.erfc(x: float) → float`

Return the complementary error function of `x`.

`math.gamma(x: float) → float`

Return the gamma function of `x`.

`math.lgamma(x: float) → float`

Return the natural logarithm of the gamma function of `x`.

## memorymonitor – Memory monitoring helpers

**exception** `memorymonitor.AllocationError`

Bases: *Exception*

Catchall exception for allocation related errors.

Initialize self. See `help(type(self))` for accurate signature.

**class** `memorymonitor.AllocationAlarm(*, minimum_block_count: int = 1)`

Throw an exception when an allocation of `minimum_block_count` or more blocks occurs while active.

Track allocations:

```
import memorymonitor

aa = memorymonitor.AllocationAlarm(minimum_block_count=2)
x = 2
# Should not allocate any blocks.
with aa:
    x = 5

# Should throw an exception when allocating storage for the 20 bytes.
with aa:
    x = bytearray(20)
```

**ignore** (*self*, *count*: int) → *AllocationAlarm*

Sets the number of applicable allocations to ignore before raising the exception. Automatically set back to zero at context exit.

Use it within a `with` block:

```
# Will not alarm because the bytearray allocation will be ignored.
with aa.ignore(2):
    x = bytearray(20)
```

`__enter__(self) → AllocationAlarm`  
Enables the alarm.

`__exit__(self) → None`  
Automatically disables the allocation alarm when exiting a context. See *Lifetime and ContextManagers* for more info.

**class** `memorymonitor.AllocationSize`

Tracks the number of allocations in power of two buckets.

It will have 16 16-bit buckets to track allocation counts. It is total allocations meaning frees are ignored. Reallocated memory is counted twice, at allocation and when reallocated with the larger size.

The buckets are measured in terms of blocks which is the finest granularity of the heap. This means bucket 0 will count all allocations less than or equal to the number of bytes per block, typically 16. Bucket 2 will be less than or equal to 4 blocks. See *bytes\_per\_block* to convert blocks to bytes.

Multiple *AllocationSizes* can be used to track different code boundaries.

Track allocations:

```
import memorymonitor

mm = memorymonitor.AllocationSize()
with mm:
    print("hello world" * 3)

for bucket, count in enumerate(mm):
    print("<", 2 ** bucket, count)
```

**bytes\_per\_block :int**

Number of bytes per block

`__enter__(self) → AllocationSize`  
Clears counts and resumes tracking.

`__exit__(self) → None`  
Automatically pauses allocation tracking when exiting a context. See *Lifetime and ContextManagers* for more info.

`__len__(self) → int`  
Returns the number of allocation buckets.

This allows you to:

```
mm = memorymonitor.AllocationSize()
print(len(mm))
```

`__getitem__(self, index: int) → Optional[int]`  
Returns the allocation count for the given bucket.

This allows you to:

```
mm = memorymonitor.AllocationSize()
print(mm[0])
```

**microcontroller – Pin references and cpu functionality**

The `microcontroller` module defines the pins from the perspective of the microcontroller. See `board` for board-specific pin mappings.

`microcontroller.cpu :Processor`

CPU information and control, such as `cpu.temperature` and `cpu.frequency` (clock frequency). This object is an instance of `microcontroller.Processor`.

`microcontroller.cpus :Processor`

CPU information and control, such as `cpus[0].temperature` and `cpus[1].frequency` (clock frequency) on chips with more than 1 cpu. The index selects which cpu. This object is an instance of `microcontroller.Processor`.

`microcontroller.delay_us (delay: int) → None`

Dedicated delay method used for very short delays. **Do not** do long delays because this stops all other functions from completing. Think of this as an empty `while` loop that runs for the specified (`delay`) time. If you have other code or peripherals (e.g audio recording) that require specific timing or processing while you are waiting, explore a different avenue such as using `time.sleep()`.

`microcontroller.disable_interrupts() → None`

Disable all interrupts. Be very careful, this can stall everything.

`microcontroller.enable_interrupts() → None`

Enable the interrupts that were enabled at the last disable.

`microcontroller.on_next_reset (run_mode: RunMode) → None`

Configure the run mode used the next time the microcontroller is reset but not powered down.

**Parameters** `run_mode` (`RunMode`) – The next run mode

`microcontroller.reset() → None`

Reset the microcontroller. After reset, the microcontroller will enter the run mode last set by `on_next_reset`.

**Warning:** This may result in file system corruption when connected to a host computer. Be very careful when calling this! Make sure the device “Safely removed” on Windows or “ejected” on Mac OSX and Linux.

`microcontroller.nvm :Optional[nvm.ByteArray]`

Available non-volatile memory. This object is the sole instance of `nvm.ByteArray` when available or `None` otherwise.

**Type** `nvm.ByteArray` or `None`

`microcontroller.watchdog :Optional[watchdog.WatchDogTimer]`

Available watchdog timer. This object is the sole instance of `watchdog.WatchDogTimer` when available or `None` otherwise.

**class** `microcontroller.Pin`

Identifies an IO pin on the microcontroller.

Identifies an IO pin on the microcontroller. They are fixed by the hardware so they cannot be constructed on demand. Instead, use `board` or `microcontroller.pin` to reference the desired pin.

**class** `microcontroller.Processor`

Microcontroller CPU information and control

Usage:

```
import microcontroller
print(microcontroller.cpu.frequency)
print(microcontroller.cpu.temperature)
```

Note that on chips **with** more than one cpu (such **as** the RP2040) `microcontroller.cpu` will **return** the value **for** CPU 0. To get values **from other** CPUs use `microcontroller.cpus` indexed by the number of the desired cpu. i.e.

```
print(microcontroller.cpus[0].temperature)
print(microcontroller.cpus[1].frequency)
```

You cannot create an instance of `microcontroller.Processor`. Use `microcontroller.cpu` to access the sole instance available.

**frequency :int**

The CPU operating frequency in Hertz. (read-only)

**reset\_reason :ResetReason**

The reason the microcontroller started up from reset state.

**temperature :Optional[float]**

The on-chip temperature, in Celsius, as a float. (read-only)

Is `None` if the temperature is not available.

**uid :bytearray**

The unique id (aka serial number) of the chip as a `bytearray`. (read-only)

**voltage :Optional[float]**

The input voltage to the microcontroller, as a float. (read-only)

Is `None` if the voltage is not available.

**class microcontroller.ResetReason**

The reason the microcontroller was last reset

**POWER\_ON :object**

The microcontroller was started from power off.

**BROWNOUT :object**

The microcontroller was reset due to too low a voltage.

**SOFTWARE :object**

The microcontroller was reset from software.

**DEEP\_SLEEP\_ALARM :object**

The microcontroller was reset for deep sleep and restarted by an alarm.

**RESET\_PIN :object**

The microcontroller was reset by a signal on its reset pin. The pin might be connected to a reset button.

**WATCHDOG :object**

The microcontroller was reset by its watchdog timer.

**UNKNOWN :object**

The microcontroller restarted for an unknown reason.

**class microcontroller.RunMode**

run state of the microcontroller

Enum-like class to define the run mode of the microcontroller and CircuitPython.



**NORMAL :RunMode**

Run CircuitPython as normal.

**SAFE\_MODE :RunMode**

Run CircuitPython in safe mode. User code will not be run and the file system will be writeable over USB.

**BOOTLOADER :RunMode**

Run the bootloader.

**msgpack – Pack object in msgpack format**

The msgpack format is similar to json, except that the encoded data is binary. See <https://msgpack.org> for details. The module implements a subset of the cpython module msgpack-python.

Not implemented: 64-bit int, uint, float.

Example 1:

```
import msgpack
from io import BytesIO

b = BytesIO()
msgpack.pack({'list': [True, False, None, 1, 3.14], 'str': 'blah'}, b)
b.seek(0)
print(msgpack.unpack(b))
```

Example 2: handling objects:

```
from msgpack import pack, unpack, ExtType
from io import BytesIO

class MyClass:
    def __init__(self, val):
        self.value = val
    def __str__(self):
        return str(self.value)

data = MyClass(b'my_value')

def encoder(obj):
    if isinstance(obj, MyClass):
        return ExtType(1, obj.value)
    return f"no encoder for {obj}"

def decoder(code, data):
    if code == 1:
        return MyClass(data)
    return f"no decoder for type {code}"

buffer = BytesIO()
pack(data, buffer, default=encoder)
buffer.seek(0)
```

(continues on next page)

(continued from previous page)

```

decoded = unpack(buffer, ext_hook=decoder)
print(f"{data} -> {buffer.getvalue()} -> {decoded}")

```

`msgpack.pack(obj: object, buffer: \_typing.WriteableBuffer, *, default: Union\[Callable\[\[object\], None\], None\] = None) → None`

Output object to buffer in msgpack format.

#### Parameters

- **obj** ([object](#)) – Object to convert to msgpack format.
- **buffer** ([WriteableBuffer](#)) – buffer to write into
- **None]] default** ([Optional\[Callable\[\[object\], None\]\]](#)) – function called for python objects that do not have a representation in msgpack format.

`msgpack.unpack(buffer: \_typing.ReadableBuffer, *, ext_hook: Union\[Callable\[\[int, bytes\], object\], None\] = None, use_list: bool = True) → object`

Unpack and return one object from buffer.

#### Parameters

- **buffer** ([ReadableBuffer](#)) – buffer to read from
- **bytes], object]] ext\_hook** ([Optional\[Callable\[\[int, bytes\], object\]\]](#)) – function called for objects in msgpack ext format.
- **use\_list** ([Optional\[bool\]](#)) – return array as list or tuple (use\_list=False).

**Return object** object read from buffer.

**class** `msgpack.ExtType` (`code: int, data: bytes`)

ExtType represents ext type in msgpack.

Constructor :param int code: type code in range 0~127. :param bytes data: representation.

**code** :int

The type code, in range 0~127.

**data** :bytes

Data.

## multiterminal – Manage additional terminal sources

The `multiterminal` module allows you to configure an additional serial terminal source. Incoming characters are accepted from both the internal serial connection and the optional secondary connection.

`multiterminal.get_secondary_terminal()` → [Optional\[BinaryIO\]](#)

Returns the current secondary terminal.

`multiterminal.set_secondary_terminal(stream: BinaryIO)` → [None](#)

Read additional input from the given stream and write out back to it. This doesn't replace the core stream (usually UART or native USB) but is mixed in instead.

**Parameters** `stream` ([stream](#)) – secondary stream

`multiterminal.clear_secondary_terminal()` → [None](#)

Clears the secondary terminal.

`multiterminal.schedule_secondary_terminal_read` (*socket:* *schedule\_secondary\_terminal\_read.socket*)  
 → `None`

In cases where the underlying OS is doing task scheduling, this notifies the OS when more data is available on the socket to read. This is useful as a callback for lwip sockets.

## neopixel\_write – Low-level neopixel implementation

The `neopixel_write` module contains a helper method to write out bytes in the 800khz neopixel protocol.

For example, to turn off a single neopixel (like the status pixel on Express boards.)

```
import board
import neopixel_write
import digitalio

pin = digitalio.DigitalInOut(board.NEOPIXEL)
pin.direction = digitalio.Direction.OUTPUT
pixel_off = bytearray([0, 0, 0])
neopixel_write.neopixel_write(pin, pixel_off)
```

`neopixel_write.neopixel_write` (*digitalinout:* `digitalio.DigitalInOut`, *buf:* `typing.ReadableBuffer`) → `None`

Write buf out on the given DigitalInOut.

### Parameters

- **digitalinout** (`DigitalInOut`) – the DigitalInOut to output with
- **buf** (`ReadableBuffer`) – The bytes to clock out. No assumption is made about color order

## network – Network Interface Management

**Warning:** This module is disabled in 6.x and will be removed in 7.x. Please use networking libraries instead.

This module provides a registry of configured NICs. It is used by the ‘socket’ module to look up a suitable NIC when a socket is created.

`network.route()` → `List[object]`  
 Returns a list of all configured NICs.

## nvm – Non-volatile memory

The `nvm` module allows you to store whatever raw bytes you wish in a reserved section non-volatile memory.

Note that this module can’t be imported and used directly. The sole instance of `ByteArray` is available at `microcontroller.nvm`.

### class nvm.ByteArray

Presents a stretch of non-volatile memory as a bytearray.

Non-volatile memory is available as a byte array that persists over reloads and power cycles. Each assignment causes an erase and write cycle so it’s recommended to assign all values to change at once.

Usage:

```
import microcontroller
microcontroller.nvm[0:3] = b"\xcc\x10\x00"
```

Not currently dynamically supported. Access the sole instance through `microcontroller.nvm`.

`__bool__(self) → bool`

`__len__(self) → int`

Return the length. This is used by (`len`)

`__getitem__(self, index: slice) → bytearray`

`__getitem__(self, index: int) → int`

Returns the value at the given index.

`__setitem__(self, index: slice, value: _typing.ReadableBuffer) → None`

`__setitem__(self, index: int, value: int) → None`

Set the value at the given index.

## os – functions that an OS normally provides

The `os` module is a strict subset of the CPython `os` module. So, code written in CircuitPython will work in CPython but not necessarily the other way around.

`os.uname() → _Uname`

Returns a named tuple of operating specific and CircuitPython port specific information.

**class** `os._Uname`

Bases: `NamedTuple`

The type of values that `uname()` returns

**sysname** :str

**nodename** :str

**release** :str

**version** :str

**machine** :str

`os.chdir(path: str) → None`

Change current directory.

`os.getcwd() → str`

Get the current directory.

`os.listdir(dir: str) → str`

With no argument, list the current directory. Otherwise list the given directory.

`os.mkdir(path: str) → None`

Create a new directory.

`os.remove(path: str) → None`

Remove a file.

`os.rmdir(path: str) → None`

Remove a directory.

`os.rename(old_path: str, new_path: str) → str`

Rename a file.

`os.stat(path: str) → Tuple[int, int, int, int, int, int, int, int, int, int]`  
Get the status of a file or directory.

---

**Note:** On builds without long integers, the number of seconds for contemporary dates will not fit in a small integer. So the time fields return 946684800, which is the number of seconds corresponding to 1999-12-31.

---

`os.statvfs(path: str) → Tuple[int, int, int, int, int, int, int, int, int, int]`  
Get the status of a filesystem.

Returns a tuple with the filesystem information in the following order:

- `f_bsize` – file system block size
- `f_frsize` – fragment size
- `f_blocks` – size of fs in `f_frsize` units
- `f_bfree` – number of free blocks
- `f_bavail` – number of free blocks for unprivileged users
- `f_files` – number of inodes
- `f_ffree` – number of free inodes
- `f_favail` – number of free inodes for unprivileged users
- `f_flag` – mount flags
- `f_namemax` – maximum filename length

Parameters related to inodes: `f_files`, `f_ffree`, `f_avail` and the `f_flags` parameter may return 0 as they can be unavailable in a port-specific implementation.

`os.sync() → None`  
Sync all filesystems.

`os.urandom(size: int) → str`  
Returns a string of `size` random bytes based on a hardware True Random Number Generator. When not available, it will raise a `NotImplementedError`.

`os.sep: str`  
Separator used to delineate path components such as folder and file names.

## ps2io – Support for PS/2 protocol

The `ps2io` module contains classes to provide PS/2 communication.

**Warning:** This module is not available in some SAMD21 builds. See the [Module Support Matrix - Which Modules Are Available on Which Boards](#) for more info.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See [Lifetime and ContextManagers](#) for more info.

**class** `ps2io.Ps2(data_pin: microcontroller.Pin, clock_pin: microcontroller.Pin)`  
Communicate with a PS/2 keyboard or mouse

Ps2 implements the PS/2 keyboard/mouse serial protocol, used in legacy devices. It is similar to UART but there are only two lines (Data and Clock). PS/2 devices are 5V, so bidirectional level converters must be used to connect the I/O lines to pins of 3.3V boards.

Create a Ps2 object associated with the given pins.

#### Parameters

- **data\_pin** ([Pin](#)) – Pin tied to data wire.
- **clock\_pin** ([Pin](#)) – Pin tied to clock wire. This pin must support interrupts.

Read one byte from PS/2 keyboard and turn on Scroll Lock LED:

```
import ps2io
import board

kbd = ps2io.Ps2(board.D10, board.D11)

while len(kbd) == 0:
    pass

print(kbd.popleft())
print(kbd.sendcmd(0xed))
print(kbd.sendcmd(0x01))
```

**deinit** (*self*) → [None](#)

Deinitialises the Ps2 and releases any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → [Ps2](#)

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → [None](#)

Automatically deinitializes the hardware when exiting a context. See [Lifetime and ContextManagers](#) for more info.

**popleft** (*self*) → [int](#)

Removes and returns the oldest received byte. When buffer is empty, raises an `IndexError` exception.

**sendcmd** (*self*, *byte*: [int](#)) → [int](#)

Sends a command byte to PS/2. Returns the response byte, typically the general ack value (0xFA). Some commands return additional data which is available through [popleft\(\)](#).

Raises a `RuntimeError` in case of failure. The root cause can be found by calling [clear\\_errors\(\)](#). It is advisable to call [clear\\_errors\(\)](#) before [sendcmd\(\)](#) to flush any previous errors.

**Parameters** **byte** ([int](#)) – byte value of the command

**clear\_errors** (*self*) → [None](#)

Returns and clears a bitmap with latest recorded communication errors.

Reception errors (arise asynchronously, as data is received):

0x01: start bit not 0

0x02: timeout

0x04: parity bit error

0x08: stop bit not 1

0x10: buffer overflow, newest data discarded

Transmission errors (can only arise in the course of [sendcmd\(\)](#)):

0x100: clock pin didn't go to LO in time  
 0x200: clock pin didn't go to HI in time  
 0x400: data pin didn't ACK  
 0x800: clock pin didn't ACK  
 0x1000: device didn't respond to RTS  
 0x2000: device didn't send a response byte in time

`__bool__(self) → bool`

`__len__(self) → int`

Returns the number of received bytes in buffer, available to `popleft()`.

## pulseio – Support for individual pulse based protocols

The `pulseio` module contains classes to provide access to basic pulse IO. Individual pulses are commonly used in infrared remotes and in DHT temperature sensors.

**Warning:** PWMOut is moving to `pwmio` and will be removed from `pulseio` in CircuitPython 7.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See *Lifetime and ContextManagers* for more info.

**class** `pulseio.PulseIn` (*pin*: `microcontroller.Pin`, *maxlen*: `int` = 2, \*, *idle\_state*: `bool` = `False`)

Measure a series of active and idle pulses. This is commonly used in infrared receivers and low cost temperature sensors (DHT). The pulsed signal consists of timed active and idle periods. Unlike PWM, there is no set duration for active and idle pairs.

Create a `PulseIn` object associated with the given pin. The object acts as a read-only sequence of pulse lengths with a given max length. When it is active, new pulse lengths are added to the end of the list. When there is no more room (`len() == maxlen`) the oldest pulse length is removed to make room.

### Parameters

- **pin** (`Pin`) – Pin to read pulses from.
- **maxlen** (`int`) – Maximum number of pulse durations to store at once
- **idle\_state** (`bool`) – Idle state of the pin. At start and after `resume` the first recorded pulse will be the opposite state from idle.

Read a short series of pulses:

```
import pulseio
import board

pulses = pulseio.PulseIn(board.D7)

# Wait for an active pulse
while len(pulses) == 0:
    pass
# Pause while we do something with the pulses
pulses.pause()
```

(continues on next page)

(continued from previous page)

```
# Print the pulses. pulses[0] is an active pulse unless the length
# reached max length and idle pulses are recorded.
print(pulses)

# Clear the rest
pulses.clear()

# Resume with an 80 microsecond active pulse
pulses.resume(80)
```

**maxlen** :int

The maximum length of the `PulseIn`. When `len()` is equal to `maxlen`, it is unclear which pulses are active and which are idle.

**paused** :bool

True when pulse capture is paused as a result of `pause()` or an error during capture such as a signal that is too fast.

**deinit** (*self*) → None

Deinitialises the `PulseIn` and releases any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → *PulseIn*

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → None

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**pause** (*self*) → None

Pause pulse capture

**resume** (*self*, *trigger\_duration*: int = 0) → None

Resumes pulse capture after an optional trigger pulse.

**Warning:** Using trigger pulse with a device that drives both high and low signals risks a short. Make sure your device is open drain (only drives low) when using a trigger pulse. You most likely added a “pull-up” resistor to your circuit to do this.

**Parameters** **trigger\_duration** (int) – trigger pulse duration in microseconds

**clear** (*self*) → None

Clears all captured pulses

**popleft** (*self*) → int

Removes and returns the oldest read pulse.

**\_\_bool\_\_** (*self*) → bool**\_\_len\_\_** (*self*) → int

Returns the current pulse length

This allows you to:

```
pulses = pulseio.PulseIn(pin)
print(len(pulses))
```



`__getitem__` (*self*, *index*: `int`) → `Optional[int]`  
Returns the value at the given index or values in slice.

This allows you to:

```
pulses = pulseio.PulseIn(pin)
print(pulses[0])
```

**class** `pulseio.PulseOut` (*carrier*: `pwmio.PWMOut`)

Pulse PWM “carrier” output on and off. This is commonly used in infrared remotes. The pulsed signal consists of timed on and off periods. Unlike PWM, there is no set duration for on and off pairs.

Create a `PulseOut` object associated with the given `PWMOut` object.

**Parameters** `carrier` (`PWMOut`) – `PWMOut` that is set to output on the desired pin.

Send a short series of pulses:

```
import array
import pulseio
import pwmio
import board

# 50% duty cycle at 38kHz.
pwm = pwmio.PWMOut(board.D13, frequency=38000, duty_cycle=32768)
pulse = pulseio.PulseOut(pwm)
#
#           on   off   on   off   on
pulses = array.array('H', [65000, 1000, 65000, 65000, 1000])
pulse.send(pulses)

# Modify the array of pulses.
pulses[0] = 200
pulse.send(pulses)
```

**deinit** (*self*) → `None`

Deinitialises the `PulseOut` and releases any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → `PulseOut`

No-op used by Context Managers.

**\_\_exit\_\_** (*self*) → `None`

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**send** (*self*, *pulses*: `_typing.ReadableBuffer`) → `None`

Pulse alternating on and off durations in microseconds starting with on. `pulses` must be an `array.array` with data type ‘H’ for unsigned halfword (two bytes).

This method waits until the whole array of pulses has been sent and ensures the signal is off afterwards.

**Parameters** `pulses` (`array.array`) – pulse durations in microseconds

## pwmio – Support for PWM based protocols

The `pwmio` module contains classes to provide access to basic pulse IO.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See [Lifetime and ContextManagers](#) for more info.

For example:

```
import pwmio
import time
from board import *

pwm = pwmio.PWMOut(D13)
pwm.duty_cycle = 2 ** 15
time.sleep(0.1)
```

This example will initialize the the device, set `duty_cycle`, and then sleep 0.1 seconds. CircuitPython will automatically turn off the PWM when it resets all hardware after program completion. Use `deinit()` or a `with` statement to do it yourself.

**class** `pwmio.PWMOut` (*pin*: `microcontroller.Pin`, \*, *duty\_cycle*: `int` = 0, *frequency*: `int` = 500, *variable\_frequency*: `bool` = False)

Output a Pulse Width Modulated signal on a given pin.

Create a PWM object associated with the given pin. This allows you to write PWM signals out on the given pin. Frequency is fixed after init unless `variable_frequency` is True.

---

**Note:** When `variable_frequency` is True, further PWM outputs may be limited because it may take more internal resources to be flexible. So, when outputting both fixed and flexible frequency signals construct the fixed outputs first.

---

### Parameters

- **pin** (`Pin`) – The pin to output to
- **duty\_cycle** (`int`) – The fraction of each pulse which is high. 16-bit
- **frequency** (`int`) – The target frequency in Hertz (32-bit)
- **variable\_frequency** (`bool`) – True if the frequency will change over time

Simple LED fade:

```
import pwmio
import board

pwm = pwmio.PWMOut(board.D13)      # output on D13
pwm.duty_cycle = 2 ** 15           # Cycles the pin with 50% duty cycle (half of
↪ 2 ** 16) at the default 500hz
```

PWM at specific frequency (servos and motors):

```
import pwmio
import board
```

(continues on next page)

(continued from previous page)

```
pwm = pwmio.PWMOut(board.D13, frequency=50)
pwm.duty_cycle = 2 ** 15 # Cycles the pin with 50% duty cycle.
↪ (half of 2 ** 16) at 50hz
```

Variable frequency (usually tones):

```
import pwmio
import board
import time

pwm = pwmio.PWMOut(board.D13, duty_cycle=2 ** 15, frequency=440, variable_
↪ frequency=True)
time.sleep(0.2)
pwm.frequency = 880
time.sleep(0.1)
```

#### **duty\_cycle :int**

16 bit value that dictates how much of one cycle is high (1) versus low (0). 0xffff will always be high, 0 will always be low and 0x7fff will be half high and then half low.

Depending on how PWM is implemented on a specific board, the internal representation for duty cycle might have less than 16 bits of resolution. Reading this property will return the value from the internal representation, so it may differ from the value set.

#### **frequency :int**

32 bit value that dictates the PWM frequency in Hertz (cycles per second). Only writeable when constructed with `variable_frequency=True`.

Depending on how PWM is implemented on a specific board, the internal value for the PWM's duty cycle may need to be recalculated when the frequency changes. In these cases, the duty cycle is automatically recalculated from the original duty cycle value. This should happen without any need to manually re-set the duty cycle.

#### **deinit (self) → None**

Deinitialises the PWMOut and releases any hardware resources for reuse.

#### **\_\_enter\_\_ (self) → PWMOut**

No-op used by Context Managers.

#### **\_\_exit\_\_ (self) → None**

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

## random – pseudo-random numbers and choices

The `random` module is a strict subset of the CPython `random` module. So, code written in CircuitPython will work in CPython but not necessarily the other way around.

Like its CPython cousin, CircuitPython's random seeds itself on first use with a true random from `os.urandom()` when available or the uptime otherwise. Once seeded, it will be deterministic, which is why its bad for cryptography.

**Warning:** Numbers from this module are not cryptographically strong! Use bytes from `os.urandom` directly for true randomness.

`random._T`

`random.seed(seed: int) → None`

Sets the starting seed of the random number generation. Further calls to `random` will return deterministic results afterwards.

`random.getrandbits(k: int) → int`

Returns an integer with  $k$  random bits.

`random.randrange(stop: Tuple[int, int, int]) → int`

Returns a randomly selected integer from `range(start, stop, step)`.

`random.randint(a: int, b: int) → int`

Returns a randomly selected integer between  $a$  and  $b$  inclusive. Equivalent to `randrange(a, b + 1, 1)`

`random.choice(seq: Sequence[_T]) → _T`

Returns a randomly selected element from the given sequence. Raises `IndexError` when the sequence is empty.

`random.random() → float`

Returns a random float between 0 and 1.0.

`random.uniform(a: float, b: float) → float`

Returns a random float between  $a$  and  $b$ . It may or may not be inclusive depending on float rounding.

## rgbmatrix – Low-level routines for bitbanged LED matrices

```
class rgbmatrix.RGBMatrix(*, width: int, bit_depth: int, rgb_pins: Sequence[digitalio.DigitalInOut],
                           addr_pins: Sequence[digitalio.DigitalInOut], clock_pin: dig-
                           italio.DigitalInOut, latch_pin: digitalio.DigitalInOut, out-
                           put_enable_pin: digitalio.DigitalInOut, doublebuffer: bool = True,
                           framebuffer: Optional[_typing.WriteableBuffer] = None, height: int =
                           0, tile: int = 1, serpentine: bool = True)
```

Displays an in-memory framebuffer to a HUB75-style RGB LED matrix.

Create a `RGBMatrix` object with the given attributes. The height of the display is determined by the number of `rgb` and address pins and the number of tiles: `len(rgb_pins) // 3 * 2 ** len(addr_pins) * abs(tile)`. With 6 RGB pins, 4 address lines, and a single matrix, the display will be 32 pixels tall. If the optional height parameter is specified and is not 0, it is checked against the calculated height.

Up to 30 RGB pins and 8 address pins are supported.

The RGB pins must be within a single “port” and performance and memory usage are best when they are all within “close by” bits of the port. The clock pin must also be on the same port as the RGB pins. See the documentation of the underlying `protomatter C` library for more information. Generally, Adafruit’s interface boards are designed so that these requirements are met when matched with the intended microcontroller board. For instance, the Feather M4 Express works together with the RGB Matrix Feather.

The framebuffer is in “RGB565” format.

“RGB565” means that it is organized as a series of 16-bit numbers where the highest 5 bits are interpreted as red, the next 6 as green, and the final 5 as blue. The object can be any buffer, but `array.array` and `ulab.array` objects are most often useful. To update the content, modify the framebuffer and call `refresh`.

If a framebuffer is not passed in, one is allocated and initialized to all black. In any case, the framebuffer can be retrieved by passing the `RGBMatrix` object to `memoryview()`.

If `doublebuffer` is `False`, some memory is saved, but the display may flicker during updates.

A `RGBMatrix` is often used in conjunction with a `framebufferio.FramebufferDisplay`.

**brightness :float**

In the current implementation, 0.0 turns the display off entirely and any other value up to 1.0 turns the display on fully.

**width :int**

The width of the display, in pixels

**height :int**

The height of the display, in pixels

**deinit** (*self*) → *None*

Free the resources (pins, timers, etc.) associated with this `rgbmatrix` instance. After deinitialization, no further operations may be performed.

**refresh** (*self*) → *None*

Transmits the color data in the buffer to the pixels so that they are shown.

## rotaryio – Support for reading rotation sensors

The `rotaryio` module contains classes to read different rotation encoding schemes. See [Wikipedia's Rotary Encoder page](#) for more background.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See [Lifetime and ContextManagers](#) for more info.

**class** `rotaryio.IncrementalEncoder` (*pin\_a*: `microcontroller.Pin`, *pin\_b*: `microcontroller.Pin`)

`IncrementalEncoder` determines the relative rotational position based on two series of pulses.

Create an `IncrementalEncoder` object associated with the given pins. It tracks the positional state of an incremental rotary encoder (also known as a quadrature encoder.) Position is relative to the position when the object is constructed.

### Parameters

- **pin\_a** (`Pin`) – First pin to read pulses from.
- **pin\_b** (`Pin`) – Second pin to read pulses from.

For example:

```
import rotaryio
import time
from board import *

enc = rotaryio.IncrementalEncoder(D1, D2)
last_position = None
while True:
    position = enc.position
    if last_position == None or position != last_position:
        print(position)
    last_position = position
```

**position :int**

The current position in terms of pulses. The number of pulses per rotation is defined by the specific hardware.

**deinit** (*self*) → *None*

Deinitializes the `IncrementalEncoder` and releases any hardware resources for reuse.

**\_\_enter\_\_** (*self*) → *IncrementalEncoder*

No-op used by Context Managers.

`__exit__(self) → None`

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

## **rp2pio – Hardware interface to RP2 series’ programmable IO (PIO) peripheral.**

`rp2pio.pins_are_sequential(pins: List[microcontroller.Pin]) → bool`

Return True if the pins have sequential GPIO numbers, False otherwise

```
class rp2pio.StateMachine(program: _typing.ReadableBuffer, frequency: int, *, init: Optional[_typing.ReadableBuffer] = None, first_out_pin: Optional[microcontroller.Pin] = None, out_pin_count: int = 1, initial_out_pin_state: int = 0, initial_out_pin_direction: int = 4294967295, first_in_pin: Optional[microcontroller.Pin] = None, in_pin_count: int = 1, pull_in_pin_up: int = 0, pull_in_pin_down: int = 0, first_set_pin: Optional[microcontroller.Pin] = None, set_pin_count: int = 1, initial_set_pin_state: int = 0, initial_set_pin_direction: int = 31, first_sideset_pin: Optional[microcontroller.Pin] = None, sideset_pin_count: int = 1, initial_sideset_pin_state: int = 0, initial_sideset_pin_direction: int = 31, exclusive_pin_use: bool = True, auto_pull: bool = False, pull_threshold: int = 32, out_shift_right: bool = True, wait_for_txstall: bool = True, auto_push: bool = False, push_threshold: int = 32, in_shift_right: bool = True)
```

A single PIO StateMachine

The programmable I/O peripheral on the RP2 series of microcontrollers is unique. It is a collection of generic state machines that can be used for a variety of protocols. State machines may be independent or coordinated. Program memory and IRQs are shared between the state machines in a particular PIO instance. They are independent otherwise.

This class is designed to facilitate sharing of PIO resources. By default, it is assumed that the state machine is used on its own and can be placed in either PIO. State machines with the same program will be placed in the same PIO if possible. To ensure multiple state machines share a PIO use the `colocate` kwarg during construction and create them one after another.

Construct a StateMachine object on the given pins with the given program.

### **Parameters**

- **program** (*ReadableBuffer*) – the program to run with the state machine
- **frequency** (*int*) – the target clock frequency of the state machine. Actual may be less.
- **init** (*ReadableBuffer*) – a program to run once at start up. This is run after program is started so instructions may be intermingled
- **first\_out\_pin** (*Pin*) – the first pin to use with the OUT instruction
- **out\_pin\_count** (*int*) – the count of consecutive pins to use with OUT starting at `first_out_pin`
- **initial\_out\_pin\_state** (*int*) – the initial output value for out pins starting at `first_out_pin`
- **initial\_out\_pin\_direction** (*int*) – the initial output direction for out pins starting at `first_out_pin`
- **first\_in\_pin** (*Pin*) – the first pin to use with the IN instruction

- **in\_pin\_count** (`int`) – the count of consecutive pins to use with IN starting at `first_in_pin`
- **pull\_in\_pin\_up** (`int`) – a 1-bit in this mask sets pull up on the corresponding in pin
- **pull\_in\_pin\_down** (`int`) – a 1-bit in this mask sets pull up on the corresponding in pin. Setting both pulls enables a “bus keep” function, i.e. a weak pull to whatever is current high/low state of GPIO.
- **first\_set\_pin** (`Pin`) – the first pin to use with the SET instruction
- **set\_pin\_count** (`int`) – the count of consecutive pins to use with SET starting at `first_set_pin`
- **initial\_set\_pin\_state** (`int`) – the initial output value for set pins starting at `first_set_pin`
- **initial\_set\_pin\_direction** (`int`) – the initial output direction for set pins starting at `first_set_pin`
- **first\_sideset\_pin** (`Pin`) – the first pin to use with a side set
- **sideset\_pin\_count** (`int`) – the count of consecutive pins to use with a side set starting at `first_sideset_pin`
- **initial\_sideset\_pin\_state** (`int`) – the initial output value for sideset pins starting at `first_sideset_pin`
- **initial\_sideset\_pin\_direction** (`int`) – the initial output direction for sideset pins starting at `first_sideset_pin`
- **exclusive\_pin\_use** (`bool`) – When True, do not share any pins with other state machines. Pins are never shared with other peripherals
- **auto\_pull** (`bool`) – When True, automatically load data from the tx FIFO into the output shift register (OSR) when an OUT instruction shifts more than `pull_threshold` bits
- **pull\_threshold** (`int`) – Number of bits to shift before loading a new value into the OSR from the tx FIFO
- **out\_shift\_right** (`bool`) – When True, data is shifted out the right side (LSB) of the OSR. It is shifted out the left (MSB) otherwise. NOTE! This impacts data alignment when the number of bytes is not a power of two (1, 2 or 4 bytes).
- **wait\_for\_txstall** (`bool`) – When True, writing data out will block until the TX FIFO and OSR are empty and an instruction is stalled waiting for more data. When False, data writes won’t wait for the OSR to empty (only the TX FIFO) so make sure you give enough time before deiniting or stopping the state machine.
- **auto\_push** (`bool`) – When True, automatically save data from input shift register (ISR) into the rx FIFO when an IN instruction shifts more than `push_threshold` bits
- **push\_threshold** (`int`) – Number of bits to shift before saving the ISR value to the RX FIFO
- **in\_shift\_right** (`bool`) – When True, data is shifted into the right side (LSB) of the ISR. It is shifted into the left (MSB) otherwise. NOTE! This impacts data alignment when the number of bytes is not a power of two (1, 2 or 4 bytes).

#### **frequency :int**

The actual state machine frequency. This may not match the frequency requested due to internal limitations.

**rxstall** :bool

True when the state machine has stalled due to a full RX FIFO since the last `clear_rxfifo` call.

**in\_waiting** :int

The number of words available to read into

**deinit** (*self*) → None

Turn off the state machine and release its resources.

**\_\_enter\_\_** (*self*) → *StateMachine*

No-op used by Context Managers. Provided by context manager helper.

**\_\_exit\_\_** (*self*) → None

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

**restart** (*self*) → None

Resets this state machine, runs any init and enables the clock.

**run** (*self*, *instructions*: *\_typing.ReadableBuffer*) → None

Runs all given instructions. They will likely be interleaved with in-memory instructions. Make sure this doesn't wait for input!

This can be used to output internal state to the RX FIFO and then read with `readinto`.

**stop** (*self*) → None

Stops the state machine clock. Use `restart` to enable it.

**write** (*self*, *buffer*: *\_typing.ReadableBuffer*, \*, *start*: int = 0, *end*: *Optional[int]* = None) → None

Write the data contained in *buffer* to the state machine. If the buffer is empty, nothing happens.

#### Parameters

- **buffer** (*ReadableBuffer*) – Write out the data in this buffer
- **start** (int) – Start of the slice of *buffer* to write out: *buffer[start:end]*
- **end** (int) – End of the slice; this index is not included. Defaults to `len(buffer)`

**readinto** (*self*, *buffer*: *\_typing.WritableBuffer*, \*, *start*: int = 0, *end*: *Optional[int]* = None) → None

**Read into buffer. If the number of bytes to read is 0, nothing happens. The buffer** include any data added to the fifo even if it was added before this was called.

#### Parameters

- **buffer** (*WritableBuffer*) – Read data into this buffer
- **start** (int) – Start of the slice of *buffer* to read into: *buffer[start:end]*
- **end** (int) – End of the slice; this index is not included. Defaults to `len(buffer)`

**write\_readinto** (*self*, *buffer\_out*: *\_typing.ReadableBuffer*, *buffer\_in*: *\_typing.WritableBuffer*, \*, *out\_start*: int = 0, *out\_end*: *Optional[int]* = None, *in\_start*: int = 0, *in\_end*: *Optional[int]* = None) → None

Write out the data in *buffer\_out* while simultaneously reading data into *buffer\_in*. The lengths of the slices defined by *buffer\_out[out\_start:out\_end]* and *buffer\_in[in\_start:in\_end]* may be different. The function will return once both are filled. If buffer slice lengths are both 0, nothing happens.

#### Parameters

- **buffer\_out** (*ReadableBuffer*) – Write out the data in this buffer
- **buffer\_in** (*WritableBuffer*) – Read data into this buffer



- **out\_start** (*int*) – Start of the slice of `buffer_out` to write out: `buffer_out[out_start:out_end]`
- **out\_end** (*int*) – End of the slice; this index is not included. Defaults to `len(buffer_out)`
- **in\_start** (*int*) – Start of the slice of `buffer_in` to read into: `buffer_in[in_start:in_end]`
- **in\_end** (*int*) – End of the slice; this index is not included. Defaults to `len(buffer_in)`

**clear\_rxfifo** (*self*) → *None*  
 Clears any unread bytes in the rxfifo.

## rtc – Real Time Clock

The `rtc` module provides support for a Real Time Clock. You can access and manage the RTC using `rtc.RTC`. It also backs the `time.time()` and `time.localtime()` functions using the onboard RTC if present.

`rtc.set_time_source(rtc: RTC)` → *None*

Sets the RTC time source used by `time.localtime()`. The default is `rtc.RTC`, but it's useful to use this to override the time source for testing purposes. For example:

```
import rtc
import time

class RTC(object):
    @property
    def datetime(self):
        return time.struct_time((2018, 3, 17, 21, 1, 47, 0, 0, 0))

r = RTC()
rtc.set_time_source(r)
```

**class** `rtc.RTC`  
 Real Time Clock

This class represents the onboard Real Time Clock. It is a singleton and will always return the same instance.

**datetime** : `time.struct_time`

The current date and time of the RTC as a `time.struct_time`.

This must be set to the current date and time whenever the board loses power:

```
import rtc
import time

r = rtc.RTC()
r.datetime = time.struct_time((2019, 5, 29, 15, 14, 15, 0, -1, -1))
```

Once set, the RTC will automatically update this value as time passes. You can read this property to get a snapshot of the current time:

```
current_time = r.datetime
print(current_time)
# struct_time(tm_year=2019, tm_month=5, ...)
```

**calibration :int**

The RTC calibration value as an *int*.

A positive value speeds up the clock and a negative value slows it down. Range and value is hardware specific, but one step is often approximately 1 ppm:

```
import rtc
import time

r = rtc.RTC()
r.calibration = 1
```

## samd – SAMD implementation settings

**class** `samd.Clock`

Identifies a clock on the microcontroller.

They are fixed by the hardware so they cannot be constructed on demand. Instead, use `samd.clock` to reference the desired clock.

**enabled :bool**

Is the clock enabled? (read-only)

**parent :Union[Clock, None]**

Clock parent. (read-only)

**frequency :int**

Clock frequency in Herz. (read-only)

**calibration :int**

Clock calibration. Not all clocks can be calibrated.

## sdcardio – Interface to an SD card via the SPI bus

**class** `sdcardio.SDCard` (*bus*: `busio.SPI`, *cs*: `microcontroller.Pin`, *baudrate*: `int = 8000000`)

SD Card Block Interface

Controls an SD card over SPI. This built-in module has higher read performance than the library `adafruit_sdcard`, but it is only compatible with `busio.SPI`, not `bitbangio.SPI`. Usually an `SDCard` object is used with `storage.VfsFat` to allow file I/O to an SD card.

Construct an SPI SD Card object with the given properties

### Parameters

- **spi** (`busio.SPI`) – The SPI bus
- **cs** (`microcontroller.Pin`) – The chip select connected to the card
- **baudrate** (`int`) – The SPI data rate to use after card setup

Note that during detection and configuration, a hard-coded low baudrate is used. Data transfers use the specified baudrate (rounded down to one that is supported by the microcontroller)

---

**Important:** If the same SPI bus is shared with other peripherals, it is important that the SD card be initialized before accessing any other peripheral on the bus. Failure to do so can prevent the SD card from being recognized until it is powered off or re-inserted.

---

Example usage:

```
import os

import board
import sdcardio
import storage

sd = sdcardio.SDCard(board.SPI(), board.SD_CS)
vfs = storage.VfsFat(sd)
storage.mount(vfs, '/sd')
os.listdir('/sd')
```

**count** (*self*) → *int*

Returns the total number of sectors

Due to technical limitations, this is a function and not a property.

**Returns** The number of 512-byte blocks, as a number

**deinit** (*self*) → *None*

Disable permanently.

**Returns** *None*

**readblocks** (*self*, *start\_block*: *int*, *buf*: *typing.WriteableBuffer*) → *None*

Read one or more blocks from the card

**Parameters**

- **start\_block** (*int*) – The block to start reading from
- **buf** (*WriteableBuffer*) – The buffer to write into. Length must be multiple of 512.

**Returns** *None*

**writeblocks** (*self*, *start\_block*: *int*, *buf*: *typing.ReadableBuffer*) → *None*

Write one or more blocks to the card

**Parameters**

- **start\_block** (*int*) – The block to start writing from
- **buf** (*ReadableBuffer*) – The buffer to read from. Length must be multiple of 512.

**Returns** *None*

## sdioio – Interface to an SD card via the SDIO bus

**class** `sdioio.SDCard` (*clock*: `microcontroller.Pin`, *command*: `microcontroller.Pin`, *data*: *Sequence[microcontroller.Pin]*, *frequency*: *int*)

SD Card Block Interface with SDIO

Controls an SD card over SDIO. SDIO is a parallel protocol designed for SD cards. It uses a clock pin, a command pin, and 1 or 4 data pins. It can be operated at a high frequency such as 25MHz. Usually an `SDCard` object is used with `storage.VfsFat` to allow file I/O to an SD card.

Construct an SDIO SD Card object with the given properties

**Parameters**

- **clock** (*Pin*) – the pin to use for the clock.
- **command** (*Pin*) – the pin to use for the command.

- **data** – A sequence of pins to use for data.
- **frequency** – The frequency of the bus in Hz

Example usage:

```
import os

import board
import sdioio
import storage

sd = sdioio.SDCard(
    clock=board.SDIO_CLOCK,
    command=board.SDIO_COMMAND,
    data=board.SDIO_DATA,
    frequency=25000000)
vfs = storage.VfsFat(sd)
storage.mount(vfs, '/sd')
os.listdir('/sd')
```

**configure** (*self*, *frequency*: `int = 0`, *width*: `int = 0`) → `None`

Configures the SDIO bus.

#### Parameters

- **frequency** (`int`) – the desired clock rate in Hertz. The actual clock rate may be higher or lower due to the granularity of available clock settings. Check the *frequency* attribute for the actual clock rate.
- **width** (`int`) – the number of data lines to use. Must be 1 or 4 and must also not exceed the number of data lines at construction

---

**Note:** Leaving a value unspecified or 0 means the current setting is kept

---

**count** (*self*) → `int`

Returns the total number of sectors

Due to technical limitations, this is a function and not a property.

**Returns** The number of 512-byte blocks, as a number

**readblocks** (*self*, *start\_block*: `int`, *buf*: `_typing.WriteableBuffer`) → `None`

Read one or more blocks from the card

#### Parameters

- **start\_block** (`int`) – The block to start reading from
- **buf** (`WriteableBuffer`) – The buffer to write into. Length must be multiple of 512.

**Returns** `None`

**writeblocks** (*self*, *start\_block*: `int`, *buf*: `_typing.ReadableBuffer`) → `None`

Write one or more blocks to the card

#### Parameters

- **start\_block** (`int`) – The block to start writing from
- **buf** (`ReadableBuffer`) – The buffer to read from. Length must be multiple of 512.

**Returns** `None`

**property frequency** (*self*) → *int*

The actual SDIO bus frequency. This may not match the frequency requested due to internal limitations.

**property width** (*self*) → *int*

The actual SDIO bus width, in bits

**deinit** (*self*) → *None*

Disable permanently.

**Returns** *None*

**\_\_enter\_\_** (*self*) → *SDCard*

No-op used by Context Managers. Provided by context manager helper.

**\_\_exit\_\_** (*self*) → *None*

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

## sharpdisplay – Support for Sharp Memory Display framebuffer

## socket – TCP, UDP and RAW socket support

**Warning:** This module is disabled in 6.x and will be removed in 7.x. Please use networking libraries instead. (Native networking will provide a socket compatible class.)

Create TCP, UDP and RAW sockets for communicating over the Internet.

**class** `socket.socket` (*family*: *int* = *AF\_INET*, *type*: *int* = *SOCK\_STREAM*, *proto*: *int* = *IPPROTO\_TCP*)

Create a new socket

**Parameters**

- **family** (*int*) – *AF\_INET* or *AF\_INET6*
- **type** (*int*) – *SOCK\_STREAM*, *SOCK\_DGRAM* or *SOCK\_RAW*
- **proto** (*int*) – *IPPROTO\_TCP*, *IPPROTO\_UDP* or *IPPROTO\_RAW* (ignored)

**AF\_INET** :*int*

**AF\_INET6** :*int*

**SOCK\_STREAM** :*int*

**SOCK\_DGRAM** :*int*

**SOCK\_RAW** :*int*

**IPPROTO\_TCP** :*int*

**bind** (*self*, *address*: *Tuple*[*str*, *int*]) → *None*

Bind a socket to an address

**Parameters** **address** (*tuple* (*str*, *int*)) – tuple of (remote\_address, remote\_port)

**listen** (*self*, *backlog*: *int*) → *None*

Set socket to listen for incoming connections

**Parameters** **backlog** (*int*) – length of backlog queue for waiting connections

**accept** (*self*) → Tuple[*socket*, *str*]

Accept a connection on a listening socket of type SOCK\_STREAM, creating a new socket of type SOCK\_STREAM. Returns a tuple of (new\_socket, remote\_address)

**connect** (*self*, *address*: Tuple[*str*, *int*]) → None

Connect a socket to a remote address

**Parameters** **address** (tuple (*str*, *int*)) – tuple of (remote\_address, remote\_port)

**send** (*self*, *bytes*: *typing.ReadableBuffer*) → *int*

Send some bytes to the connected remote address. Suits sockets of type SOCK\_STREAM

**Parameters** **bytes** (*ReadableBuffer*) – some bytes to send

**recv\_into** (*self*, *buffer*: *typing.WriteableBuffer*, *bufsize*: *int*) → *int*

Reads some bytes from the connected remote address, writing into the provided buffer. If bufsize <= len(buffer) is given, a maximum of bufsize bytes will be read into the buffer. If no valid value is given for bufsize, the default is the length of the given buffer.

Suits sockets of type SOCK\_STREAM Returns an int of number of bytes read.

**Parameters**

- **buffer** (*WriteableBuffer*) – buffer to receive into
- **bufsize** (*int*) – optionally, a maximum number of bytes to read.

**recv** (*self*, *bufsize*: *int*) → *bytes*

Reads some bytes from the connected remote address. Suits sockets of type SOCK\_STREAM Returns a bytes() of length <= bufsize

**Parameters** **bufsize** (*int*) – maximum number of bytes to receive

**sendto** (*self*, *bytes*: *typing.ReadableBuffer*, *address*: Tuple[*str*, *int*]) → *int*

Send some bytes to a specific address. Suits sockets of type SOCK\_DGRAM

**Parameters**

- **bytes** (*ReadableBuffer*) – some bytes to send
- **address** (tuple (*str*, *int*)) – tuple of (remote\_address, remote\_port)

**recvfrom** (*self*, *bufsize*: *int*) → Tuple[*bytes*, Tuple[*str*, *int*]]

Reads some bytes from the connected remote address. Suits sockets of type SOCK\_STREAM

Returns a tuple containing \* a bytes() of length <= bufsize \* a remote\_address, which is a tuple of ip address and port number

**Parameters** **bufsize** (*int*) – maximum number of bytes to receive

**setsockopt** (*self*, *level*: *int*, *optname*: *int*, *value*: *int*) → None

Sets socket options

**settimeout** (*self*, *value*: *int*) → None

Set the timeout value for this socket.

**Parameters** **value** (*int*) – timeout in seconds. 0 means non-blocking. None means block indefinitely.

**setblocking** (*self*, *flag*: *bool*) → Optional[*int*]

Set the blocking behaviour of this socket.

**Parameters** **flag** (*bool*) – False means non-blocking, True means block indefinitely.

`socket.getaddrinfo(host: str, port: int) → Tuple[int, int, int, str, str]`

Gets the address information for a hostname and port

Returns the appropriate family, socket type, socket protocol and address information to call `socket.socket()` and `socket.connect()` with, as a tuple.

## socketpool

The `socketpool` module provides sockets through a pool. The pools themselves act like CPython's `socket` module.

**class** `socketpool.Socket`

TCP, UDP and RAW socket. Cannot be created directly. Instead, call `SocketPool.socket()`.

Provides a subset of CPython's `socket.socket` API. It only implements the versions of `recv` that do not allocate bytes objects.

`__enter__(self) → Socket`

No-op used by Context Managers.

`__exit__(self) → None`

Automatically closes the Socket when exiting a context. See *Lifetime and ContextManagers* for more info.

`accept(self) → Tuple[Socket, Tuple[str, int]]`

Accept a connection on a listening socket of type `SOCK_STREAM`, creating a new socket of type `SOCK_STREAM`. Returns a tuple of (new\_socket, remote\_address)

`bind(self, address: Tuple[str, int]) → None`

Bind a socket to an address

**Parameters** `address` (~tuple) – tuple of (remote\_address, remote\_port)

`close(self) → None`

Closes this Socket and makes its resources available to its SocketPool.

`connect(self, address: Tuple[str, int]) → None`

Connect a socket to a remote address

**Parameters** `address` (~tuple) – tuple of (remote\_address, remote\_port)

`listen(self, backlog: int) → None`

Set socket to listen for incoming connections

**Parameters** `backlog` (~int) – length of backlog queue for waiting connetions

`recvfrom_into(self, buffer: _typing.WritableBuffer) → Tuple[int, Tuple[str, int]]`

Reads some bytes from a remote address.

Returns a tuple containing \* the number of bytes received into the given buffer \* a remote\_address, which is a tuple of ip address and port number

**Parameters** `buffer` (object) – buffer to read into

`recv_into(self, buffer: _typing.WritableBuffer, bufsize: int) → int`

Reads some bytes from the connected remote address, writing into the provided buffer. If `bufsize <= len(buffer)` is given, a maximum of `bufsize` bytes will be read into the buffer. If no valid value is given for `bufsize`, the default is the length of the given buffer.

Suits sockets of type `SOCK_STREAM` Returns an int of number of bytes read.

**Parameters**

- `buffer` (bytearray) – buffer to receive into

- **bufsize** (*int*) – optionally, a maximum number of bytes to read.

**send** (*self*, *bytes*: *\_typing.ReadableBuffer*) → *int*

Send some bytes to the connected remote address. Suits sockets of type SOCK\_STREAM

**Parameters** *bytes* (~*bytes*) – some bytes to send

**sendto** (*self*, *bytes*: *\_typing.ReadableBuffer*, *address*: *Tuple[str, int]*) → *int*

Send some bytes to a specific address. Suits sockets of type SOCK\_DGRAM

**Parameters**

- **bytes** (~*bytes*) – some bytes to send
- **address** (~*tuple*) – tuple of (remote\_address, remote\_port)

**setblocking** (*self*, *flag*: *bool*) → *Optional[int]*

Set the blocking behaviour of this socket.

**Parameters** *flag* (~*bool*) – False means non-blocking, True means block indefinitely.

**settimeout** (*self*, *value*: *int*) → *None*

Set the timeout value for this socket.

**Parameters** *value* (~*int*) – timeout in seconds. 0 means non-blocking. None means block indefinitely.

**\_\_hash\_\_** (*self*) → *int*

Returns a hash for the Socket.

**class** `socketpool.SocketPool`

A pool of socket resources available for the given radio. Only one SocketPool can be created for each radio.

SocketPool should be used in place of CPython's socket which provides a pool of sockets provided by the underlying OS.

**AF\_INET** :*int*

**AF\_INET6** :*int*

**SOCK\_STREAM** :*int*

**SOCK\_DGRAM** :*int*

**SOCK\_RAW** :*int*

**socket** (*self*, *family*: *int* = *AF\_INET*, *type*: *int* = *SOCK\_STREAM*) → *Socket*

Create a new socket

**Parameters**

- **family** (~*int*) – AF\_INET or AF\_INET6
- **type** (~*int*) – SOCK\_STREAM, SOCK\_DGRAM or SOCK\_RAW

`socketpool.getaddrinfo` (*host*: *str*, *port*: *int*, *family*: *int* = 0, *type*: *int* = 0, *proto*: *int* = 0, *flags*: *int* = 0) → *Tuple[int, int, int, str, Tuple[str, int]]*

Gets the address information for a hostname and port

Returns the appropriate family, socket type, socket protocol and address information to call `socket.socket()` and `socket.connect()` with, as a tuple.



**ssl**

The `ssl` module provides SSL contexts to wrap sockets in.

`ssl.create_default_context()` → *SSLContext*

Return the default SSLContext.

**class ssl.SSLContext**

Settings related to SSL that can be applied to a socket by wrapping it. This is useful to provide SSL certificates to specific connections rather than all of them.

`ssl.wrap_socket(sock: socketpool.Socket, *, server_side: bool = False, server_hostname: Optional[str] = None)` → *SSLSocket*

Wraps the socket into a socket-compatible class that handles SSL negotiation. The socket must be of type SOCK\_STREAM.

**class ssl.SSLSocket**

Implements TLS security on a subset of `socketpool.Socket` functions. Cannot be created directly. Instead, call `wrap_socket` on an existing socket object.

Provides a subset of CPython's `ssl.SSLSocket` API. It only implements the versions of `recv` that do not allocate bytes objects.

`__enter__(self)` → *SSLSocket*

No-op used by Context Managers.

`__exit__(self)` → *None*

Automatically closes the Socket when exiting a context. See *Lifetime and ContextManagers* for more info.

`accept(self)` → *Tuple[SSLSocket, Tuple[str, int]]*

Accept a connection on a listening socket of type SOCK\_STREAM, creating a new socket of type SOCK\_STREAM. Returns a tuple of (new\_socket, remote\_address)

`bind(self, address: Tuple[str, int])` → *None*

Bind a socket to an address

**Parameters** `address` (~*tuple*) – tuple of (remote\_address, remote\_port)

`close(self)` → *None*

Closes this Socket

`connect(self, address: Tuple[str, int])` → *None*

Connect a socket to a remote address

**Parameters** `address` (~*tuple*) – tuple of (remote\_address, remote\_port)

`listen(self, backlog: int)` → *None*

Set socket to listen for incoming connections

**Parameters** `backlog` (~*int*) – length of backlog queue for waiting connetions

`recv_into(self, buffer: typing.WriteableBuffer, bufsize: int)` → *int*

Reads some bytes from the connected remote address, writing into the provided buffer. If `bufsize <= len(buffer)` is given, a maximum of `bufsize` bytes will be read into the buffer. If no valid value is given for `bufsize`, the default is the length of the given buffer.

Suits sockets of type SOCK\_STREAM Returns an int of number of bytes read.

**Parameters**

- **buffer** (*bytearray*) – buffer to receive into
- **bufsize** (*int*) – optionally, a maximum number of bytes to read.

**send** (*self*, *bytes*: `_typing.ReadableBuffer`) → *int*

Send some bytes to the connected remote address. Suits sockets of type SOCK\_STREAM

**Parameters** **bytes** (`~bytes`) – some bytes to send

**settimeout** (*self*, *value*: *int*) → *None*

Set the timeout value for this socket.

**Parameters** **value** (`~int`) – timeout in seconds. 0 means non-blocking. None means block indefinitely.

**setblocking** (*self*, *flag*: *bool*) → *Optional[int]*

Set the blocking behaviour of this socket.

**Parameters** **flag** (`~bool`) – False means non-blocking, True means block indefinitely.

**\_\_hash\_\_** (*self*) → *int*

Returns a hash for the Socket.

## storage – Storage management

The `storage` provides storage management functionality such as mounting and unmounting which is typically handled by the operating system hosting Python. CircuitPython does not have an OS, so this module provides this functionality directly.

`storage.mount` (*filesystem*: `VfsFat`, *mount\_path*: *str*, \*, *readonly*: *bool* = *False*) → *None*

Mounts the given filesystem object at the given path.

This is the CircuitPython analog to the UNIX `mount` command.

**Parameters** **readonly** (*bool*) – True when the filesystem should be readonly to CircuitPython.

`storage.umount` (*mount*: *Union[str, VfsFat]*) → *None*

Unmounts the given filesystem object or if *mount* is a path, then unmount the filesystem mounted at that location.

This is the CircuitPython analog to the UNIX `umount` command.

`storage.remount` (*mount\_path*: *str*, *readonly*: *bool* = *False*, \*, *disable\_concurrent\_write\_protection*: *bool* = *False*) → *None*

Remounts the given path with new parameters.

### Parameters

- **readonly** (*bool*) – True when the filesystem should be readonly to CircuitPython.
- **disable\_concurrent\_write\_protection** (*bool*) – When True, the check that makes sure the underlying filesystem data is written by one computer is disabled. Disabling the protection allows CircuitPython and a host to write to the same filesystem with the risk that the filesystem will be corrupted.

`storage.getmount` (*mount\_path*: *str*) → *VfsFat*

Retrieves the mount object associated with the mount path

`storage.erase_filesystem` () → *None*

Erase and re-create the CIRCUITPY filesystem.

On boards that present USB-visible CIRCUITPY drive (e.g., SAMD21 and SAMD51), then call `microcontroller.reset()` to restart CircuitPython and have the host computer remount CIRCUITPY.

This function can be called from the REPL when CIRCUITPY has become corrupted.

**Warning:** All the data on CIRCUITPY will be lost, and CircuitPython will restart on certain boards.

**class** `storage.VfsFat` (*block\_device*: `str`)

Create a new VfsFat filesystem around the given block device.

**Parameters** `block_device` – Block device the the filesystem lives on

**label** : `str`

The filesystem label, up to 11 case-insensitive bytes. Note that this property can only be set when the device is writable by the microcontroller.

**mkfs** (*self*) → `None`

Format the block device, deleting any data that may have been there

**open** (*self*, *path*: `str`, *mode*: `str`) → `None`

Like builtin `open()`

**ilistdir** (*self*, *path*: `str`) → `Iterator[Union[Tuple[AnyStr, int, int, int], Tuple[AnyStr, int, int]]]`

Return an iterator whose values describe files and folders within `path`

**mkdir** (*self*, *path*: `str`) → `None`

Like `os.mkdir`

**rmdir** (*self*, *path*: `str`) → `None`

Like `os.rmdir`

**stat** (*self*, *path*: `str`) → `Tuple[int, int, int, int, int, int, int, int, int, int]`

Like `os.stat`

**statvfs** (*self*, *path*: `int`) → `Tuple[int, int, int, int, int, int, int, int, int, int]`

Like `os.statvfs`

**mount** (*self*, *readonly*: `bool`, *mkfs*: `VfsFat`) → `None`

Don't call this directly, call `storage.mount`.

**umount** (*self*) → `None`

Don't call this directly, call `storage.umount`.

## struct – Manipulation of c-style data

This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: `struct`.

Supported size/byte order prefixes: `@`, `<`, `>`, `!`.

Supported format codes: `b`, `B`, `x`, `h`, `H`, `i`, `I`, `l`, `L`, `q`, `Q`, `s`, `P`, `f`, `d` (the latter 2 depending on the floating-point support).

`struct.calcsize` (*fmt*: `str`) → `int`

Return the number of bytes needed to store the given `fmt`.

`struct.pack` (*fmt*: `str`, *\*values*: `Any`) → `bytes`

Pack the values according to the format string `fmt`. The return value is a bytes object encoding the values.

`struct.pack_into` (*fmt*: `str`, *buffer*: `typing.WriteableBuffer`, *offset*: `int`, *\*values*: `Any`) → `None`

Pack the values according to the format string `fmt` into a buffer starting at `offset`. `offset` may be negative to count from the end of buffer.

`struct.unpack` (*fmt*: `str`, *data*: `typing.ReadableBuffer`) → `Tuple[Any, ...]`

Unpack from the data according to the format string `fmt`. The return value is a tuple of the unpacked values. The buffer size must match the size required by the format.

`struct.unpack_from(fmt: str, data: \_typing.ReadableBuffer, offset: int = 0) → Tuple[Any, ...]`

Unpack from the data starting at offset according to the format string `fmt`. `offset` may be negative to count from the end of buffer. The return value is a tuple of the unpacked values. The buffer size must be at least as big as the size required by the form.

## **supervisor – Supervisor settings**

`supervisor.runtime :Runtime`

Runtime information, such as `runtime.serial_connected` (USB serial connection status). This object is the sole instance of `supervisor.Runtime`.

`supervisor.enable_autoreload() → None`

Enable autoreload based on USB file write activity.

`supervisor.disable_autoreload() → None`

Disable autoreload based on USB file write activity until `enable_autoreload` is called.

`supervisor.set_rgb_status_brightness(brightness: int) → None`

Set brightness of status neopixel from 0-255 `set_rgb_status_brightness` is called.

`supervisor.reload() → None`

Reload the main Python code and run it (equivalent to hitting Ctrl-D at the REPL).

`supervisor.set_next_stack_limit(size: int) → None`

Set the size of the stack for the next vm run. If its too large, the default will be used.

**class** `supervisor.RunReason`

The reason that CircuitPython started running.

**STARTUP :object**

CircuitPython started the microcontroller started up. See `microcontroller.Processor.reset_reason` for more detail on why the microcontroller was started.

**AUTO\_RELOAD :object**

CircuitPython restarted due to an external write to the filesystem.

**SUPERVISOR\_RELOAD :object**

CircuitPython restarted due to a call to `supervisor.reload()`.

**REPL\_RELOAD :object**

CircuitPython started due to the user typing CTRL-D in the REPL.

**class** `supervisor.Runtime`

Current status of runtime objects.

Usage:

```
import supervisor
if supervisor.runtime.serial_connected:
    print("Hello World!")
```

You cannot create an instance of `supervisor.Runtime`. Use `supervisor.runtime` to access the sole instance available.

**usb\_connected :bool**

Returns the USB enumeration status (read-only).

**serial\_connected :bool**

Returns the USB serial communication status (read-only).

**serial\_bytes\_available** :int

Returns the whether any bytes are available to read on the USB serial input. Allows for polling to see whether to call the built-in input() or wait. (read-only)

**run\_reason** :RunReason

Returns why CircuitPython started running this particular time.

## terminalio – Displays text in a TileGrid

The `terminalio` module contains classes to display a character stream on a display. The built in font is available as `terminalio.FONT`.

`terminalio.FONT` :fontio.BuiltinFont

The built in font

**class** `terminalio.Terminal` (*tilegrid*: displayio.TileGrid, *font*: fontio.BuiltinFont)

Display a character stream with a TileGrid

Terminal manages tile indices and cursor position based on VT100 commands. The font should be a `fontio.BuiltinFont` and the TileGrid's bitmap should match the font's bitmap.

**write** (*self*, *buf*: typing.ReadableBuffer) → Optional[int]

Write the buffer of bytes to the bus.

**Returns** the number of bytes written

**Return type** int or None

## time – time and timing related functions

The `time` module is a strict subset of the CPython `time` module. So, code written in MicroPython will work in CPython but not necessarily the other way around.

`time.monotonic`() → float

Returns an always increasing value of time with an unknown reference point. Only use it to compare against other values from `monotonic`.

**Returns** the current monotonic time

**Return type** float

`time.sleep` (*seconds*: float) → None

Sleep for a given number of seconds.

**Parameters** *seconds* (float) – the time to sleep in fractional seconds

**class** `time.struct_time` (*time\_tuple*: Tuple[int, int, int, int, int, int, int, int, int, int])

Structure used to capture a date and time. Note that it takes a tuple!

**Parameters** *time\_tuple* (tuple) – Tuple of time info: (tm\_year, tm\_mon, tm\_mday, tm\_hour, tm\_min, tm\_sec, tm\_wday, tm\_yday, tm\_isdst)

- tm\_year: the year, 2017 for example
- tm\_mon: the month, range [1, 12]
- tm\_mday: the day of the month, range [1, 31]
- tm\_hour: the hour, range [0, 23]
- tm\_min: the minute, range [0, 59]

- `tm_sec`: the second, range [0, 61]
- `tm_wday`: the day of the week, range [0, 6], Monday is 0
- `tm_yday`: the day of the year, range [1, 366], -1 indicates not known
- `tm_isdst`: 1 when in daylight savings, 0 when not, -1 if unknown.

`time.time()` → *int*

Return the current time in seconds since since Jan 1, 1970.

**Returns** the current time

**Return type** *int*

`time.monotonic_ns()` → *int*

Return the time of the monotonic clock, cannot go backward, in nanoseconds.

**Returns** the current time

**Return type** *int*

`time.localtime(secs: int)` → *struct\_time*

Convert a time expressed in seconds since Jan 1, 1970 to a `struct_time` in local time. If `secs` is not provided or `None`, the current time as returned by `time()` is used. The earliest date for which it can generate a time is Jan 1, 2000.

**Returns** the current time

**Return type** *time.struct\_time*

`time.mktime(t: struct_time)` → *int*

This is the inverse function of `localtime()`. Its argument is the `struct_time` or full 9-tuple (since the `dst` flag is needed; use -1 as the `dst` flag if it is unknown) which expresses the time in local time, not UTC. The earliest date for which it can generate a time is Jan 1, 2000.

**Returns** seconds

**Return type** *int*

## **touchio – Touch related IO**

The `touchio` module contains classes to provide access to touch IO typically accelerated by hardware on the onboard microcontroller.

All classes change hardware state and should be deinitialized when they are no longer needed if the program continues after use. To do so, either call `deinit()` or use a context manager. See [Lifetime and ContextManagers](#) for more info.

For example:

```
import touchio
from board import *

touch_pin = touchio.TouchIn(D6)
print(touch_pin.value)
```

This example will initialize the the device, and print the *value*.

**class** `touchio.TouchIn(pin: microcontroller.Pin)`

Read the state of a capacitive touch sensor

Usage:

```
import touchio
from board import *

touch = touchio.TouchIn(A1)
while True:
    if touch.value:
        print("touched!")
```

Use the `TouchIn` on the given pin.

**Parameters** `pin` (`Pin`) – the pin to read from

**value** :`bool`

Whether the touch pad is being touched or not. (read-only)

True when `raw_value > threshold`.

**raw\_value** :`int`

The raw touch measurement as an `int`. (read-only)

**threshold** :`Optional[int]`

Minimum `raw_value` needed to detect a touch (and for `value` to be `True`).

When the `TouchIn` object is created, an initial `raw_value` is read from the pin, and then `threshold` is set to be 100 + that value.

You can adjust `threshold` to make the pin more or less sensitive:

```
import board
import touchio

touch = touchio.TouchIn(board.A1)
touch.threshold = 7300
```

**deinit** (`self`) → `None`

Deinitialises the `TouchIn` and releases any hardware resources for reuse.

**\_\_enter\_\_** (`self`) → `TouchIn`

No-op used by Context Managers.

**\_\_exit\_\_** (`self`) → `None`

Automatically deinitializes the hardware when exiting a context. See *Lifetime and ContextManagers* for more info.

## uheap – Heap size analysis

`uheap.info` (`object: info.object`) → `int`

Prints memory debugging info for the given object and returns the estimated size.

## ulab – Manipulate numeric data similar to numpy

`ulab` is a numpy-like module for micropython, meant to simplify and speed up common mathematical operations on arrays. The primary goal was to implement a small subset of numpy that might be useful in the context of a microcontroller. This means low-level data processing of linear (array) and two-dimensional (matrix) data.

`ulab` is adapted from micropython-ulab, and the original project's documentation can be found at <https://micropython-ulab.readthedocs.io/en/latest/>

`ulab` is modeled after numpy, and aims to be a compatible subset where possible. Numpy's documentation can be found at <https://docs.scipy.org/doc/numpy/index.html>

### `ulab.approx` – Numerical approximation methods

`ulab.approx.bisect` (*fun: Callable[[float], float], a: float, b: float, \*, xtol: float = 2.4e-07, maxiter: int = 100*)  $\rightarrow$  *float*

#### Parameters

- **f** (*callable*) – The function to bisect
- **a** (*float*) – The left side of the interval
- **b** (*float*) – The right side of the interval
- **xtol** (*float*) – The tolerance value
- **maxiter** (*float*) – The maximum number of iterations to perform

Find a solution (zero) of the function  $f(x)$  on the interval  $(a..b)$  using the bisection method. The result is accurate to within `xtol` unless more than `maxiter` steps are required.

`ulab.approx.fmin` (*fun: Callable[[float], float], x0: float, \*, xatol: float = 2.4e-07, fatol: float = 2.4e-07, maxiter: int = 200*)  $\rightarrow$  *float*

#### Parameters

- **f** (*callable*) – The function to bisect
- **x0** (*float*) – The initial x value
- **xatol** (*float*) – The absolute tolerance value
- **fatol** (*float*) – The relative tolerance value

Find a minimum of the function  $f(x)$  using the downhill simplex method. The located  $x$  is within `fxtol` of the actual minimum, and  $f(x)$  is within `fatol` of the actual minimum unless more than `maxiter` steps are required.

`ulab.approx.interp` (*x: ulab.array, xp: ulab.array, fp: ulab.array, \*, left: Optional[float] = None, right: Optional[float] = None*)  $\rightarrow$  *ulab.array*

#### Parameters

- **x** (*ulab.array*) – The x-coordinates at which to evaluate the interpolated values.
- **xp** (*ulab.array*) – The x-coordinates of the data points, must be increasing
- **fp** (*ulab.array*) – The y-coordinates of the data points, same length as `xp`
- **left** – Value to return for  $x < xp[0]$ , default is `fp[0]`.
- **right** – Value to return for  $x > xp[-1]$ , default is `fp[-1]`.



Returns the one-dimensional piecewise linear interpolant to a function with given discrete data points (xp, fp), evaluated at x.

```
ulab.approx.newton (fun: Callable[[float], float], x0: float, *, xtol: float = 2.4e-07, rtol: float = 0.0,
                    maxiter: int = 50) → float
```

#### Parameters

- **f** (callable) – The function to bisect
- **x0** (float) – The initial x value
- **xtol** (float) – The absolute tolerance value
- **rtol** (float) – The relative tolerance value
- **maxiter** (float) – The maximum number of iterations to perform

Find a solution (zero) of the function  $f(x)$  using Newton's Method. The result is accurate to within  $xtol * rtol * |f(x)|$  unless more than **maxiter** steps are required.

```
ulab.approx.trapz (y: ulab.array, x: Optional[ulab.array] = None, dx: float = 1.0) → float
```

#### Parameters

- **ulab.array y** (1D) – the values of the dependent variable
- **ulab.array x** (1D) – optional, the coordinates of the independent variable. Defaults to uniformly spaced values.
- **dx** (float) – the spacing between sample points, if x=None

Returns the integral of  $y(x)$  using the trapezoidal rule.

### ulab.compare – Comparison functions

```
ulab.compare.clip (x1: Union[ulab.array, float], x2: Union[ulab.array, float], x3: Union[ulab.array,
float]) → ulab.array
```

Constrain the values from  $x1$  to be between  $x2$  and  $x3$ .  $x2$  is assumed to be less than or equal to  $x3$ .

Arguments may be ulab arrays or numbers. All array arguments must be the same size. If the inputs are all scalars, a single scalar is returned.

Shorthand for `ulab.maximum(x2, ulab.minimum(x1, x3))`

```
ulab.compare.equal (x1: Union[ulab.array, float], x2: Union[ulab.array, float]) → List[bool]
```

Return an array of bool which is true where  $x1[i] == x2[i]$  and false elsewhere

```
ulab.compare.not_equal (x1: Union[ulab.array, float], x2: Union[ulab.array, float]) → List[bool]
```

Return an array of bool which is false where  $x1[i] == x2[i]$  and true elsewhere

```
ulab.compare.maximum (x1: Union[ulab.array, float], x2: Union[ulab.array, float]) → ulab.array
```

Compute the element by element maximum of the arguments.

Arguments may be ulab arrays or numbers. All array arguments must be the same size. If the inputs are both scalars, a number is returned

```
ulab.compare.minimum (x1: Union[ulab.array, float], x2: Union[ulab.array, float]) → ulab.array
```

Compute the element by element minimum of the arguments.

Arguments may be ulab arrays or numbers. All array arguments must be the same size. If the inputs are both scalars, a number is returned

## `ulab.fft` – Frequency-domain functions

`ulab.fft.fft` (*r*: `ulab.array`, *c*: `Optional[ulab.array]` = `None`) → `Tuple[ulab.array, ulab.array]`

### Parameters

- **r** (`ulab.array`) – A 1-dimension array of values whose size is a power of 2
- **c** (`ulab.array`) – An optional 1-dimension array of values whose size is a power of 2, giving the complex part of the value

**Return tuple (r, c)** The real and complex parts of the FFT

Perform a Fast Fourier Transform from the time domain into the frequency domain

See also `~ulab.extras.spectrum`, which computes the magnitude of the fft, rather than separately returning its real and imaginary parts.

`ulab.fft.ifft` (*r*: `ulab.array`, *c*: `Optional[ulab.array]` = `None`) → `Tuple[ulab.array, ulab.array]`

### Parameters

- **r** (`ulab.array`) – A 1-dimension array of values whose size is a power of 2
- **c** (`ulab.array`) – An optional 1-dimension array of values whose size is a power of 2, giving the complex part of the value

**Return tuple (r, c)** The real and complex parts of the inverse FFT

Perform an Inverse Fast Fourier Transform from the frequency domain into the time domain

`ulab.fft.spectrogram` (*r*: `ulab.array`) → `ulab.array`

**Parameters** **r** (`ulab.array`) – A 1-dimension array of values whose size is a power of 2

Computes the spectrum of the input signal. This is the absolute value of the (complex-valued) fft of the signal. This function is similar to `scipy.signal.spectrogram`.

## `ulab.filter` – Filtering functions

`ulab.filter.convolve` (*a*: `ulab.array`, *v*: `ulab.array`) → `ulab.array`

### Parameters

- **a** (`ulab.array`) –
- **v** (`ulab.array`) –

Returns the discrete, linear convolution of two one-dimensional sequences. The result is always an array of float. Only the `full` mode is supported, and the `mode` named parameter of `numpy` is not accepted. Note that all other modes can be had by slicing a `full` result.

Convolution filters can implement high pass, low pass, band pass, etc., filtering operations. Convolution filters are typically constructed ahead of time. This can be done using desktop python with `scipy`, or on web pages such as <https://fiiir.com/>

Convolution is most time-efficient when both inputs are of float type.

`ulab.filter.sosfilt` (*sos*: `ulab._ArrayLike`, *x*: `ulab._ArrayLike`) → `ulab.array`

`ulab.filter.sosfilt` (*sos*: `ulab._ArrayLike`, *x*: `ulab._ArrayLike`, \*, *zi*: `ulab.array`) → `Tuple[ulab.array, ulab.array]`

### Parameters

- **sos** (`ulab.array`) – Array of second-order filter coefficients, must have shape (n\_sections, 6). Each row corresponds to a second-order section, with the first three columns providing the numerator coefficients and the last three providing the denominator coefficients.
- **x** (`ulab.array`) – The data to be filtered
- **zi** (`ulab.array`) – Optional initial conditions for the filter

**Returns** If `zi` is not specified, the filter result alone is returned. If `zi` is specified, the return value is a 2-tuple of the filter result and the final filter conditions.

Filter data along one dimension using cascaded second-order sections.

Filter a data sequence, `x`, using a digital IIR filter defined by `sos`.

The filter function is implemented as a series of second-order filters with direct-form II transposed structure. It is designed to minimize numerical precision errors for high-order filters.

Filter coefficients can be generated by using scipy's filter generators such as `signal.ellip(..., output='sos')`.

## `ulab.linalg` – Linear algebra functions

`ulab.linalg.cholesky` (`A`: `ulab.array`) → `ulab.array`

**Parameters** `A` (`array`) – a positive definite, symmetric square matrix

**Return** `~ulab.array L` a square root matrix in the lower triangular form

**Raises** `ValueError` – If the input does not fulfill the necessary conditions

The returned matrix satisfies the equation  $m=LL^*$

`ulab.linalg.det` (`m`: `ulab.array`) → `float`

**Param** `m`, a square matrix

**Return** `float` The determinant of the matrix

Computes the eigenvalues and eigenvectors of a square matrix

`ulab.linalg.dot` (`m1`: `ulab.array`, `m2`: `ulab.array`) → Union[`ulab.array`, `float`]

**Parameters**

- **m1** (`array`) – a matrix, or a vector
- **m2** (`array`) – a matrix, or a vector

Computes the product of two matrices, or two vectors. In the latter case, the inner product is returned.

`ulab.linalg.eig` (`m`: `ulab.array`) → Tuple[`ulab.array`, `ulab.array`]

**Parameters** `m` – a square matrix

**Return** tuple (eigenvectors, eigenvalues)

Computes the eigenvalues and eigenvectors of a square matrix

`ulab.linalg.inv` (`m`: `ulab.array`) → `ulab.array`

**Parameters** `m` (`array`) – a square matrix

**Returns** The inverse of the matrix, if it exists

**Raises** `ValueError` – if the matrix is not invertible

Computes the inverse of a square matrix

`ulab.linalg.norm(x: ulab.array) → float`

**Parameters** **x** (*array*) – a vector or a matrix

Computes the 2-norm of a vector or a matrix, i.e.,  $\sqrt{\text{sum}(x*x)}$ , however, without the RAM overhead.

`ulab.linalg.trace(m: ulab.array) → float`

**Parameters** **m** – a square matrix

Compute the trace of the matrix, the sum of its diagonal elements.

## **ulab.numerical – Numerical and Statistical functions**

Most of these functions take an “axis” argument, which indicates whether to operate over the flattened array (None), or a particular axis (integer).

`ulab.numerical.argmax(array: ulab._ArrayLike, *, axis: Optional[int] = None) → int`

Return the index of the maximum element of the 1D array

`ulab.numerical.argmin(array: ulab._ArrayLike, *, axis: Optional[int] = None) → int`

Return the index of the minimum element of the 1D array

`ulab.numerical.argsort(array: ulab.array, *, axis: int = -1) → ulab.array`

Returns an array which gives indices into the input array from least to greatest.

`ulab.numerical.cross(a: ulab.array, b: ulab.array) → ulab.array`

Return the cross product of two vectors of length 3

`ulab.numerical.diff(array: ulab.array, *, n: int = 1, axis: int = -1) → ulab.array`

Return the numerical derivative of successive elements of the array, as an array. axis=None is not supported.

`ulab.numerical.flip(array: ulab.array, *, axis: Optional[int] = None) → ulab.array`

Returns a new array that reverses the order of the elements along the given axis, or along all axes if axis is None.

`ulab.numerical.max(array: ulab._ArrayLike, *, axis: Optional[int] = None) → float`

Return the maximum element of the 1D array

`ulab.numerical.mean(array: ulab._ArrayLike, *, axis: Optional[int] = None) → float`

Return the mean element of the 1D array, as a number if axis is None, otherwise as an array.

`ulab.numerical.median(array: ulab.array, *, axis: int = -1) → ulab.array`

Find the median value in an array along the given axis, or along all axes if axis is None.

`ulab.numerical.min(array: ulab._ArrayLike, *, axis: Optional[int] = None) → float`

Return the minimum element of the 1D array

`ulab.numerical.roll(array: ulab.array, distance: int, *, axis: Optional[int] = None) → None`

Shift the content of a vector by the positions given as the second argument. If the axis keyword is supplied, the shift is applied to the given axis. The array is modified in place.

`ulab.numerical.sort(array: ulab.array, *, axis: int = -1) → ulab.array`

Sort the array along the given axis, or along all axes if axis is None. The array is modified in place.

`ulab.numerical.std(array: ulab._ArrayLike, *, axis: Optional[int] = None, ddof: int = 0) → float`

Return the standard deviation of the array, as a number if axis is None, otherwise as an array.

`ulab.numerical.sum(array: ulab._ArrayLike, *, axis: Optional[int] = None) → Union[float, int, ulab.array]`

Return the sum of the array, as a number if axis is None, otherwise as an array.

## ulab.poly – Polynomial functions

`ulab.poly.polyfit` (*y*: `ulab._ArrayLike`, *degree*: `int`) → `ulab.array`

`ulab.poly.polyfit` (*x*: `ulab._ArrayLike`, *y*: `ulab._ArrayLike`, *degree*: `int`) → `ulab.array`

Return a polynomial of given degree that approximates the function  $f(x)=y$ . If *x* is not supplied, it is the `range(len(y))`.

`ulab.poly.polyval` (*p*: `ulab._ArrayLike`, *x*: `ulab._ArrayLike`) → `ulab.array`

Evaluate the polynomial *p* at the points *x*. *x* must be an array.

## ulab.user – This module should hold arbitrary user-defined functions.

## ulab.vector – Element-by-element functions

These functions can operate on numbers, 1-D iterables, 1-D arrays, or 2-D arrays by applying the function to every element in the array. This is typically much more efficient than expressing the same operation as a Python loop.

`ulab.vector.acos` (*a*: `ulab._ArrayLike`) → `ulab.array`

Computes the inverse cosine function

`ulab.vector.acosh` (*a*: `ulab._ArrayLike`) → `ulab.array`

Computes the inverse hyperbolic cosine function

`ulab.vector.asin` (*a*: `ulab._ArrayLike`) → `ulab.array`

Computes the inverse sine function

`ulab.vector.asinh` (*a*: `ulab._ArrayLike`) → `ulab.array`

Computes the inverse hyperbolic sine function

`ulab.vector.around` (*a*: `ulab._ArrayLike`, \*, *decimals*: `int` = 0) → `ulab.array`

Returns a new float array in which each element is rounded to *decimals* places.

`ulab.vector.atan` (*a*: `ulab._ArrayLike`) → `ulab.array`

Computes the inverse tangent function; the return values are in the range  $[-\pi/2, \pi/2]$ .

`ulab.vector.arctan2` (*ya*: `ulab._ArrayLike`, *xa*: `ulab._ArrayLike`) → `ulab.array`

Computes the inverse tangent function of *y/x*; the return values are in the range  $[-\pi, \pi]$ .

`ulab.vector.atanh` (*a*: `ulab._ArrayLike`) → `ulab.array`

Computes the inverse hyperbolic tangent function

`ulab.vector.ceil` (*a*: `ulab._ArrayLike`) → `ulab.array`

Rounds numbers up to the next whole number

`ulab.vector.cos` (*a*: `ulab._ArrayLike`) → `ulab.array`

Computes the cosine function

`ulab.vector.cosh` (*a*: `ulab._ArrayLike`) → `ulab.array`

Computes the hyperbolic cosine function

`ulab.vector.degrees` (*a*: `ulab._ArrayLike`) → `ulab.array`

Converts angles from radians to degrees

`ulab.vector.erf` (*a*: `ulab._ArrayLike`) → `ulab.array`

Computes the error function, which has applications in statistics

`ulab.vector.erfc` (*a*: `ulab._ArrayLike`) → `ulab.array`

Computes the complementary error function, which has applications in statistics

`ulab.vector.exp(a: ulab._ArrayLike) → ulab.array`  
Computes the exponent function.

`ulab.vector.expm1(a: ulab._ArrayLike) → ulab.array`  
Computes  $e^x - 1$ . In certain applications, using this function preserves numeric accuracy better than the `exp` function.

`ulab.vector.floor(a: ulab._ArrayLike) → ulab.array`  
Rounds numbers up to the next whole number

`ulab.vector.gamma(a: ulab._ArrayLike) → ulab.array`  
Computes the gamma function

`ulab.vector.lgamma(a: ulab._ArrayLike) → ulab.array`  
Computes the natural log of the gamma function

`ulab.vector.log(a: ulab._ArrayLike) → ulab.array`  
Computes the natural log

`ulab.vector.log10(a: ulab._ArrayLike) → ulab.array`  
Computes the log base 10

`ulab.vector.log2(a: ulab._ArrayLike) → ulab.array`  
Computes the log base 2

`ulab.vector.radians(a: ulab._ArrayLike) → ulab.array`  
Converts angles from degrees to radians

`ulab.vector.sin(a: ulab._ArrayLike) → ulab.array`  
Computes the sine function

`ulab.vector.sinh(a: ulab._ArrayLike) → ulab.array`  
Computes the hyperbolic sine

`ulab.vector.sqrt(a: ulab._ArrayLike) → ulab.array`  
Computes the square root

`ulab.vector.tan(a: ulab._ArrayLike) → ulab.array`  
Computes the tangent

`ulab.vector.tanh(a: ulab._ArrayLike) → ulab.array`  
Computes the hyperbolic tangent

`ulab.vector.vectorize(f: Union[Callable[[int], float], Callable[[float], float]], *, otypes: Optional[ulab._DType] = None) → Callable[[ulab._ArrayLike], ulab.array]`

#### Parameters

- **f** (*callable*) – The function to wrap
- **otypes** – List of array types that may be returned by the function. None is interpreted to mean the return value is float.

Wrap a Python function `f` so that it can be applied to arrays. The callable must return only values of the types specified by `otypes`, or the result is undefined.

`ulab._DType`  
`ulab.int8, ulab.uint8, ulab.int16, ulab.uint16, ulab.float` or `ulab.bool`

`ulab._float`  
Type alias of the builtin float

`ulab._bool`  
Type alias of the builtin bool

`ulab._Index`

```
class ulab.array(values: Union[array, Iterable[Union[_float, _bool, Iterable[Any]]], *, dtype:
    _DType = ulab.float)
    1- and 2- dimensional array
```

#### Parameters

- **values** (*sequence*) – Sequence giving the initial content of the array.
- **dtype** (*\_DType*) – The type of array values, `ulab.int8`, `ulab.uint8`, `ulab.int16`, `ulab.uint16`, `ulab.float` or `ulab.bool`

The `values` sequence can either be another `~ulab.array`, sequence of numbers (in which case a 1-dimensional array is created), or a sequence where each subsequence has the same length (in which case a 2-dimensional array is created).

Passing a `ulab.array` and a different dtype can be used to convert an array from one dtype to another.

In many cases, it is more convenient to create an array from a function like `zeros` or `linspace`.

`ulab.array` implements the buffer protocol, so it can be used in many places an `array.array` can be used.

**shape** :`Tuple[int, ...]`

The size of the array, a tuple of length 1 or 2

**size** :`int`

The number of elements in the array

**itemsize** :`int`

The size of a single item in the array

**strides** :`Tuple[int, ...]`

Tuple of bytes to step in each dimension, a tuple of length 1 or 2

**copy** (*self*) → `array`

Return a copy of the array

**dtype** (*self*) → `_DType`

Returns the dtype of the array

**flatten** (*self*, \*, *order*: `str` = 'C') → `array`

**Parameters** **order** – Whether to flatten by rows ('C') or columns ('F')

Returns a new `ulab.array` object which is always 1 dimensional. If order is 'C' (the default), then the data is ordered in rows; If it is 'F', then the data is ordered in columns. "C" and "F" refer to the typical storage organization of the C and Fortran languages.

**reshape** (*self*, *shape*: `Tuple[int, ...]`) → `array`

Returns an array containing the same data with a new shape.

**sort** (*self*, \*, *axis*: `Optional[int]` = 1) → `None`

**Parameters** **axis** – Whether to sort elements within rows (0), columns (1), or elements (None)

**tobytes** (*self*) → `bytearray`

Return the raw data bytes in the array

**transpose** (*self*) → `array`

Swap the rows and columns of a 2-dimensional array

**\_\_add\_\_** (*self*, *other*: `Union[array, _float]`) → `array`

Adds corresponding elements of the two arrays, or adds a number to all elements of the array. If both arguments are arrays, their sizes must match.

`__radd__ (self, other: \_float) → array`

`__sub__ (self, other: Union[array, \_float]) → array`

Subtracts corresponding elements of the two arrays, or subtracts a number from all elements of the array.  
If both arguments are arrays, their sizes must match.

`__rsub__ (self, other: \_float) → array`

`__mul__ (self, other: Union[array, \_float]) → array`

Multiplies corresponding elements of the two arrays, or multiplies all elements of the array by a number.  
If both arguments are arrays, their sizes must match.

`__rmul__ (self, other: \_float) → array`

`__div__ (self, other: Union[array, \_float]) → array`

Multiplies corresponding elements of the two arrays, or divides all elements of the array by a number. If  
both arguments are arrays, their sizes must match.

`__rdiv__ (self, other: \_float) → array`

`__pow__ (self, other: Union[array, \_float]) → array`

Computes the power ( $x^y$ ) of corresponding elements of the the two arrays, or one number and one array.  
If both arguments are arrays, their sizes must match.

`__rpow__ (self, other: \_float) → array`

`__inv__ (self) → array`

`__neg__ (self) → array`

`__pos__ (self) → array`

`__abs__ (self) → array`

`__len__ (self) → int`

`__lt__ (self, other: Union[array, \_float]) → array`

Return `self < value`.

`__le__ (self, other: Union[array, \_float]) → array`

Return `self <= value`.

`__gt__ (self, other: Union[array, \_float]) → array`

Return `self > value`.

`__ge__ (self, other: Union[array, \_float]) → array`

Return `self >= value`.

`__iter__ (self) → Union[Iterator[array], Iterator[\_float]]`

`__getitem__ (self, index: \_Index) → Union[array, \_float]`

Retrieve an element of the array.

`__setitem__ (self, index: \_Index, value: Union[array, \_float]) → None`

Set an element of the array.

`ulab.\_ArrayLike`

`ulab.array, List[float], Tuple[float] or range`

`ulab.int8 : \_DType`

Type code for signed integers in the range -128 .. 127 inclusive, like the 'b' typecode of `array.array`

`ulab.int16 : \_DType`

Type code for signed integers in the range -32768 .. 32767 inclusive, like the 'h' typecode of `array.array`



`ulab.float :_DType`  
Type code for floating point values, like the 'f' typecode of `array.array`

`ulab.uint8 :_DType`  
Type code for unsigned integers in the range 0 .. 255 inclusive, like the 'H' typecode of `array.array`

`ulab.uint16 :_DType`  
Type code for unsigned integers in the range 0 .. 65535 inclusive, like the 'h' typecode of `array.array`

`ulab.bool :_DType`  
Type code for boolean values

`ulab.get_printoptions ()` → Dict[str, int]  
Get printing options

`ulab.set_printoptions (threshold: Optional[int] = None, edgeitems: Optional[int] = None)` → None  
Set printing options

`ulab.ndinfo (array: array)` → None

`ulab.arange (stop: _float, step: _float = 1, *, dtype: _DType = ulab.float)` → array

`ulab.arange (start: _float, stop: _float, step: _float = 1, *, dtype: _DType = ulab.float)` → array  
Return a new 1-D array with elements ranging from start to stop, with step size step.

`ulab.concatenate (arrays: Tuple[array], *, axis: int = 0)` → array  
Join a sequence of arrays along an existing axis.

`ulab.diag (a: array, *, k: int = 0)` → array  
Return specified diagonals.

`ulab.eye (size: int, *, M: Optional[int] = None, k: int = 0, dtype: _DType = ulab.float)` → array  
Return a new square array of size, with the diagonal elements set to 1 and the other elements set to 0.

`ulab.full (shape: Union[int, Tuple[int, ...]], fill_value: Union[_float, _bool], *, dtype: _DType = ulab.float)` → array  
Return a new array of the given shape with all elements set to 0.

`ulab.linspace (start: _float, stop: _float, *, dtype: _DType = ulab.float, num: int = 50, endpoint: _bool = True, retstep: _bool = False)` → array  
Return a new 1-D array with num elements ranging from start to stop linearly.

`ulab.logspace (start: _float, stop: _float, *, dtype: _DType = ulab.float, num: int = 50, endpoint: _bool = True, base: _float = 10.0)` → array  
Return a new 1-D array with num evenly spaced elements on a log scale. The sequence starts at base \*\* start, and ends with base \*\* stop.

`ulab.ones (shape: Union[int, Tuple[int, ...]], *, dtype: _DType = ulab.float)` → array  
Return a new array of the given shape with all elements set to 1.

`ulab.zeros (shape: Union[int, Tuple[int, ...]], *, dtype: _DType = ulab.float)` → array  
Return a new array of the given shape with all elements set to 0.

`ulab.frombuffer (buffer: _typing.ReadableBuffer, dtype: _DType = ulab.float, count: int = -1, offset: int = 0)` → array  
Interpret a buffer as a 1-dimensional array.

## usb\_cdc – USB CDC Serial streams

The `usb_cdc` module allows access to USB CDC (serial) communications.

`usb_cdc.serials : Tuple[Serial, ...]`

Tuple of all CDC streams. Each item is a `Serial`. `serials[0]` is the USB REPL connection. `serials[1]` is a second USB serial connection, unconnected to the REPL.

**class** `usb_cdc.Serial`

Receives cdc commands over USB

You cannot create an instance of `usb_cdc.Serial`. The available instances are in the `usb_cdc.serials` tuple.

**connected** : `bool`

True if this Serial is connected to a host. (read-only)

---

**Note:** The host is considered to be connected if it is asserting DTR (Data Terminal Ready). Most terminal programs and `pyserial` assert DTR when opening a serial connection. However, the C# `SerialPort` API does not. You must set `SerialPort.DtrEnable`.

---

**in\_waiting** : `int`

Returns the number of bytes waiting to be read on the USB serial input. (read-only)

**out\_waiting** : `int`

Returns the number of bytes waiting to be written on the USB serial output. (read-only)

**timeout** : `Optional[float]`

The initial value of `timeout` is `None`. If `None`, wait indefinitely to satisfy the conditions of a read operation. If 0, do not wait. If > 0, wait only `timeout` seconds.

**write\_timeout** : `Optional[float]`

The initial value of `write_timeout` is `None`. If `None`, wait indefinitely to finish writing all the bytes passed to `write()`. If 0, do not wait. If > 0, wait only `write_timeout` seconds.

**read** (*self*, *size*: `int` = 1) → `bytes`

Read at most `size` bytes. If `size` exceeds the internal buffer size only the bytes in the buffer will be read. If `timeout` is > 0 or `None`, and fewer than `size` bytes are available, keep waiting until the timeout expires or `size` bytes are available.

**Returns** Data read

**Return type** `bytes`

**readinto** (*self*, *buf*: `_typing.WriteableBuffer`) → `int`

Read bytes into the `buf`. If `nbytes` is specified then read at most that many bytes, subject to `timeout`. Otherwise, read at most `len(buf)` bytes.

**Returns** number of bytes read and stored into `buf`

**Return type** `bytes`

**readline** (*self*, *size*: `int` = -1) → `Optional[bytes]`

Read a line ending in a newline character (“\n”), including the newline. Return everything readable if no newline is found and `timeout` is 0. Return `None` in case of error.

This is a binary stream: the newline character “\n” cannot be changed. If the host computer transmits “\r” it will also be included as part of the line.

**Parameters** `size` (`int`) – maximum number of characters to read. -1 means as many as possible.

**Returns** the line read

**Return type** *bytes* or *None*

**readlines** (*self*) → List[Optional[*bytes*]]

Read multiple lines as a list, using *readline()*.

**Warning:** If *timeout* is *None*, *readlines()* will never return, because there is no way to indicate end of stream.

**Returns** a list of the line read

**Return type** *list*

**write** (*self*, *buf*: *\_typing.ReadableBuffer*) → *int*

Write as many bytes as possible from the buffer of bytes.

**Returns** the number of bytes written

**Return type** *int*

**flush** (*self*) → *None*

Force out any unwritten bytes, waiting until they are written.

**reset\_input\_buffer** (*self*) → *None*

Clears any unread bytes.

**reset\_output\_buffer** (*self*) → *None*

Clears any unwritten bytes.

## usb\_hid – USB Human Interface Device

The *usb\_hid* module allows you to output data as a HID device.

**usb\_hid.devices** :*Tuple*[*Device*, ...]

Tuple of all active HID device interfaces.

**class** *usb\_hid.Device*

HID Device

Usage:

```
import usb_hid

mouse = usb_hid.devices[0]

mouse.send_report()
```

Not currently dynamically supported.

**last\_received\_report** :*bytes*

The HID OUT report as a *bytes*. (read-only). *None* if nothing received.

**usage\_page** :*int*

The usage page of the device as an *int*. Can be thought of a category. (read-only)

**usage** :*int*

The functionality of the device as an *int*. (read-only)

For example, Keyboard is 0x06 within the generic desktop usage page 0x01. Mouse is 0x02 within the same usage page.

**send\_report** (*self*, *buf*: `_typing.ReadableBuffer`) → `None`

Send a HID report.

## usb\_midi – MIDI over USB

The `usb_midi` module contains classes to transmit and receive MIDI messages over USB.

`usb_midi.ports` : `Tuple[Union[PortIn, PortOut], ...]`

Tuple of all MIDI ports. Each item is either `PortIn` or `PortOut`.

**class** `usb_midi.PortIn`

Receives midi commands over USB

You cannot create an instance of `usb_midi.PortIn`.

`PortIn` objects are constructed for every corresponding entry in the USB descriptor and added to the `usb_midi.ports` tuple.

**read** (*self*, *nbytes*: `Optional[int]` = `None`) → `Optional[bytes]`

Read characters. If *nbytes* is specified then read at most that many bytes. Otherwise, read everything that arrives until the connection times out. Providing the number of bytes expected is highly recommended because it will be faster.

**Returns** Data read

**Return type** `bytes` or `None`

**readinto** (*self*, *buf*: `_typing.WritableBuffer`, *nbytes*: `Optional[int]` = `None`) → `Optional[bytes]`

Read bytes into the *buf*. If *nbytes* is specified then read at most that many bytes. Otherwise, read at most `len(buf)` bytes.

**Returns** number of bytes read and stored into *buf*

**Return type** `bytes` or `None`

**class** `usb_midi.PortOut`

Sends midi messages to a computer over USB

You cannot create an instance of `usb_midi.PortOut`.

`PortOut` objects are constructed for every corresponding entry in the USB descriptor and added to the `usb_midi.ports` tuple.

**write** (*self*, *buf*: `_typing.ReadableBuffer`) → `Optional[int]`

Write the buffer of bytes to the bus.

**Returns** the number of bytes written

**Return type** `int` or `None`

## ustack – Stack information and analysis

`ustack.max_stack_usage()` → *int*

Return the maximum excursion of the stack so far.

`ustack.stack_size()` → *int*

Return the size of the entire stack. Same as in `micropython.mem_info()`, but returns a value instead of just printing it.

`ustack.stack_usage()` → *int*

Return how much stack is currently in use. Same as `micropython.stack_use()`; duplicated here for convenience.

## vectorio – Lightweight 2d shapes for displays

**class** `vectorio.Circle` (*radius: int*)

Circle is positioned on screen by its center point.

**Parameters** **radius** – The radius of the circle in pixels

**radius** :*int*

The radius of the circle in pixels.

**class** `vectorio.Polygon` (*points: List[Tuple[int, int]]*)

Represents a closed shape by ordered vertices

**Parameters** **points** – Vertices for the polygon

**points** :*List[Tuple[int, int]]*

Set a new look and shape for this polygon

**class** `vectorio.Rectangle` (*width: int, height: int*)

Represents a rectangle by defining its bounds

**Parameters**

- **width** – The number of pixels wide
- **height** – The number of pixels high

**class** `vectorio.VectorShape` (*shape: Union[Polygon, Rectangle, Circle], pixel\_shader: Union[displayio.ColorConverter, displayio.Palette], x: int = 0, y: int = 0*)

Binds a vector shape to a location and pixel color

**Parameters**

- **shape** – The shape to draw.
- **pixel\_shader** – The pixel shader that produces colors from values
- **x** – Initial x position of the center axis of the shape within the parent.
- **y** – Initial y position of the center axis of the shape within the parent.

**x** :*int*

X position of the center point of the shape in the parent.

**y** :*int*

Y position of the center point of the shape in the parent.

**pixel\_shader** :*Union[displayio.ColorConverter, displayio.Palette]*

The pixel shader of the shape.

## watchdog – Watchdog Timer

The `watchdog` module provides support for a Watchdog Timer. This timer will reset the device if it hasn't been fed after a specified amount of time. This is useful to ensure the board has not crashed or locked up. Note that on some platforms the watchdog timer cannot be disabled once it has been enabled.

The `WatchDogTimer` is used to restart the system when the application crashes and ends up into a non recoverable state. Once started it cannot be stopped or reconfigured in any way. After enabling, the application must “feed” the watchdog periodically to prevent it from expiring and resetting the system.

Example usage:

```
from microcontroller import watchdog as w
from watchdog import WatchDogMode
w.timeout=2.5 # Set a timeout of 2.5 seconds
w.mode = WatchDogMode.RAISE
w.feed()
```

**class** `watchdog.WatchDogMode`

run state of the watchdog timer

Enum-like class to define the run mode of the watchdog timer.

**RAISE** :`WatchDogMode`

Raise an exception when the `WatchDogTimer` expires.

**RESET** :`WatchDogMode`

Reset the system if the `WatchDogTimer` expires.

**class** `watchdog.WatchDogTimer`

Timer that is used to detect code lock ups and automatically reset the microcontroller when one is detected.

A lock up is detected when the watchdog hasn't been fed after a given duration. So, make sure to call `feed` within the timeout.

Not currently dynamically supported. Access the sole instance through `microcontroller.watchdog`.

**timeout** :`float`

The maximum number of seconds that can elapse between calls to `feed()`

**mode** :`WatchDogMode`

The current operating mode of the `WatchDogTimer` `watchdog.WatchDogMode`.

Setting a `WatchDogMode` activates the `WatchDog`:

```
import microcontroller
import watchdog

w = microcontroller.watchdog
w.timeout = 5
w.mode = watchdog.WatchDogMode.RAISE
```

Once set, the `WatchDogTimer` will perform the specified action if the timer expires.

**feed** (*self*) → `None`

Feed the watchdog timer. This must be called regularly, otherwise the timer will expire.

**deinit** (*self*) → *None*

Stop the watchdog timer. This may raise an error if the watchdog timer cannot be disabled on this platform.

## wifi

The *wifi* module provides necessary low-level functionality for managing wifi connections. Use *socketpool* for communicating over the network.

**wifi.radio** :*Radio*

Wifi radio used to manage both station and AP modes. This object is the sole instance of *wifi.Radio*.

**class** *wifi.Network*

A wifi network provided by a nearby access point.

You cannot create an instance of *wifi.Network*. They are returned by *wifi.Radio.start\_scanning\_networks*.

**ssid** :*str*

String id of the network

**bssid** :*bytes*

BSSID of the network (usually the AP's MAC address)

**rssi** :*int*

Signal strength of the network

**channel** :*int*

Channel number the network is operating on

**country** :*str*

String id of the country code

**authmode** :*str*

String id of the authmode

**class** *wifi.Radio*

Native wifi radio.

This class manages the station and access point functionality of the native Wifi radio.

You cannot create an instance of *wifi.Radio*. Use *wifi.radio* to access the sole instance available.

**enabled** :*bool*

True when the wifi radio is enabled. If you set the value to *False*, any open sockets will be closed.

**mac\_address** :*bytes*

MAC address of the wifi radio. (read-only)

**hostname** :*\_typing.ReadableBuffer*

Hostname for wifi interface. When the hostname is altered after interface started/connected the changes would only be reflected once the interface restarts/reconnects.

**ipv4\_gateway** :*Optional[ipaddress.IPv4Address]*

IP v4 Address of the gateway when connected to an access point. None otherwise.

**ipv4\_subnet** :*Optional[ipaddress.IPv4Address]*

IP v4 Address of the subnet when connected to an access point. None otherwise.

**ipv4\_address** :*Optional[ipaddress.IPv4Address]*

IP v4 Address of the radio when connected to an access point. None otherwise.

**ipv4\_dns** :Optional[[ipaddress.IPv4Address](#)]

IP v4 Address of the DNS server in use when connected to an access point. None otherwise.

**ap\_info** :Optional[[Network](#)]

Network object containing BSSID, SSID, authmode, channel, country and RSSI when connected to an access point. None otherwise.

**start\_scanning\_networks** (*self*, \*, *start\_channel*: [int](#) = 1, *stop\_channel*: [int](#) = 11) → [Iterable](#)[[Network](#)]

Scans for available wifi networks over the given channel range. Make sure the channels are allowed in your country.

**stop\_scanning\_networks** (*self*) → [None](#)

Stop scanning for Wifi networks and free any resources used to do it.

**connect** (*self*, *ssid*: [\\_typing.ReadableBuffer](#), *password*: [\\_typing.ReadableBuffer](#) = b'', \*, *channel*: [Optional](#)[[int](#)] = 0, *bssid*: [Optional](#)[[\\_typing.ReadableBuffer](#)] = b'', *timeout*: [Optional](#)[[float](#)] = [None](#)) → [bool](#)

Connects to the given ssid and waits for an ip address. Reconnections are handled automatically once one connection succeeds.

By default, this will scan all channels and connect to the access point (AP) with the given *ssid* and greatest signal strength (*rss*i).

If *channel* is given, the scan will begin with the given channel and connect to the first AP with the given *ssid*. This can speed up the connection time significantly because a full scan doesn't occur.

If *bssid* is given, the scan will start at the first channel or the one given and connect to the AP with the given *bssid* and *ssid*.

**ping** (*self*, *ip*: [ipaddress.IPv4Address](#), \*, *timeout*: [Optional](#)[[float](#)] = 0.5) → [float](#)

Ping an IP to test connectivity. Returns echo time in seconds. Returns None when it times out.

**class** [wifi.ScannedNetworks](#)

Iterates over all [wifi.Network](#) objects found while scanning. This object is always created by a [wifi.Radio](#): it has no user-visible constructor.

Cannot be instantiated directly. Use [wifi.Radio.start\\_scanning\\_networks](#).

**\_\_iter\_\_** (*self*) → [Iterator](#)[[Network](#)]

Returns itself since it is the iterator.

**\_\_next\_\_** (*self*) → [Network](#)

Returns the next [wifi.Network](#). Raises [StopIteration](#) if scanning is finished and no other results are available.

**wiznet** – Support for WizNet hardware, including the WizNet 5500 Ethernet adaptor.

**Warning:** This module is disabled in 6.x and will be removed in 7.x. Please use networking libraries instead.

**class** [wiznet.WIZNET5K](#) (*spi*: [busio.SPI](#), *cs*: [microcontroller.Pin](#), *rst*: [microcontroller.Pin](#), *dhcp*: [bool](#) = [True](#))

Wrapper for Wiznet 5500 Ethernet interface

Create a new WIZNET5500 interface using the specified pins

**Parameters**

- **spi** ([SPI](#)) – spi bus to use



- **cs** (`Pin`) – pin to use for Chip Select
  - **rst** (`Pin`) – pin to use for Reset (optional)
  - **dhcp** (`bool`) – boolean flag, whether to start DHCP automatically (optional, keyword only, default `True`)
- The reset pin is optional: if supplied it is used to reset the wiznet board before initialization.
  - The SPI bus will be initialized appropriately by this library.
  - At present, the WIZNET5K object is a singleton, so only one WizNet interface is supported at a time.

**connected** :`bool`

(boolean, readonly) is this device physically connected?

**dhcp** :`bool`

(boolean, readwrite) is DHCP active on this device?

- set to `True` to activate DHCP, `False` to turn it off

**ifconfig** (*self*, *params*: *Optional*[*Tuple*[*str*, *str*, *str*, *str*]] = *None*) → *Optional*[*Tuple*[*str*, *str*, *str*, *str*]]

Called without parameters, returns a tuple of (ip\_address, subnet\_mask, gateway\_address, dns\_server)

Or can be called with the same tuple to set those parameters. Setting ifconfig parameters turns DHCP off, if it was on.

### **help()** – Built-in method to provide helpful information

**help** (*object*=*None*)

Prints a help method about the given object. When *object* is none, prints general port information.

## 1.8.2 Supported Ports

CircuitPython supports a number of microcontroller families. Support quality for each varies depending on the active contributors for each port.

Adafruit sponsored developers are actively contributing to atmel-samd, mimxrt10xx, nrf and stm ports. They also maintain the other ports in order to ensure the boards build. Additional testing is limited.

### **SAMD21 and SAMD51**

This port supports many development boards that utilize SAMD21 and SAMD51 chips. See <https://circuitpython.org/downloads> for all supported boards.

### **Building**

For build instructions see this guide: <https://learn.adafruit.com/building-circuitpython/>

## Debugging

For debugging instructions see this guide: <https://learn.adafruit.com/debugging-the-samd21-with-gdb>

## Port Specific modules

### CircuitPython port to Spresense

This directory contains the port of CircuitPython to Spresense. It is a compact development board based on Sony's power-efficient multicore microcontroller CXD5602.

Board features:

- Integrated GPS
  - The embedded GNSS with support for GPS, QZSS and GLONASS enables applications where tracking is required.
- Hi-res audio output and multi mic inputs
  - Advanced 192kHz/24 bit audio codec and amplifier for audio output, and support for up to 8 mic input channels.
- Multicore microcontroller
  - Spresense is powered by Sony's CXD5602 microcontroller (ARM® Cortex®-M4F × 6 cores), with a clock speed of 156 MHz.

Currently, Spresense port does not support Audio and Multicore.

Refer to [developer.sony.com/develop/spresense/](https://developer.sony.com/develop/spresense/) for further information about this board.

## Prerequisites

### Linux

Add user to dialout group:

```
$ sudo usermod -a -G dialout <user-name>
```

### Windows

Download and install USB serial driver

- CP210x USB to serial driver for Windows 7/8/8.1
- CP210x USB to serial driver for Windows 10

## macOS

Download and install USB serial driver

- [CP210x USB to serial driver for Mac OS X](#)

## Build instructions

Pull all submodules into your clone:

```
$ git submodule update --init --recursive
```

Build the MicroPython cross-compiler:

```
$ make -C mpy-cross
```

Change directory to cxd56:

```
$ cd ports/cxd56
```

To build circuitpython image run:

```
$ make BOARD=spresense
```

## USB connection

Connect the `Spresense main board` to the PC via the USB cable.

## Flash the bootloader

The correct bootloader is required for the Spresense board to function.

Bootloader information:

- The bootloader has to be flashed the very first time the board is used.
- You have to accept the End User License Agreement to be able to download and use the Spresense bootloader binary.

Download the spresense binaries zip archive from: [Spresense firmware v2-0-002](#)

Extract spresense binaries in your PC to `ports/spresense/spresense-exported-sdk/firmware/`

To flash the bootloader run the command:

```
$ make BOARD=spresense flash-bootloader
```

## Flash the circuitpython image

To flash the firmware run the command:

```
$ make BOARD=spresense flash
```

## Accessing the board

Connect the Spresense extension board to the PC via the USB cable.

Once built and deployed, access the CircuitPython REPL (the Python prompt) via USB. You can run:

```
$ screen /dev/ttyACM0 115200
```

## Circuitpython on ESP32-S2

This port adds the ESP32-S2 line of modules from Espressif to Circuitpython. ESP32-S2 modules are low power, single-core Wi-Fi microcontroller SoCs designed for IoT applications.

### How this port is organized:

- **bindings/** contains some required bindings to the ESP-IDF for exceptions and memory.
- **boards/** contains the configuration files for each development board and breakout available on the port.
- **common-hal/** contains the port-specific module implementations, used by shared-module and shared-bindings.
- **esp-idf/** contains the Espressif IoT development framework installation, including all the drivers for the port.
- **modules/** contains information specific to certain ESP32-S2 hardware modules, such as the pins used for flash and RAM on the WROVER and WROOM.
- **peripherals/** contains peripheral setup files and peripheral mapping information, sorted by family and sub-variant. Most files in this directory can be generated with the python scripts in **tools/**.
- **supervisor/** contains port-specific implementations of internal flash, serial and USB, as well as the **port.c** file, which initializes the port at startup.
- **tools/** includes useful python scripts for debugging and other purposes.

At the root level, refer to **mpconfigboard.h** and **mpconfigport.mk** for port specific settings and a list of enabled circuitpython modules.

## Connecting to the ESP32-S2

The USB port built into ESP32-S2 boards such as the Saola is not the native USB of the board, but a debugging and programming interface. The actual ESP32-S2 native USB which exposes the Circuitpython drive and CDC connection is located on IO pins 19 and 20:

GPIO	USB
20	D+ (green)
19	D- (white)
GND	GND (black)
5V	+5V (red)

Connect these pins using a [USB adapter](#) or [breakout cable](#) to access the Circuitpython drive.

## Building and flashing

Before building or flashing the ESP32-S2, you must [install the esp-idf](#). This must be re-done every time the esp-idf is updated, but not every time you build. Run `cd ports/esp32s2` from `circuitpython/` to move to the esp32s2 port root, and run:

```
./esp-idf/install.sh
```

After this initial installation, you must add the esp-idf tools to your path. You must also do this **any time you open a new bash environment for building or flashing**:

```
. esp-idf/export.sh
```

When Circuitpython updates the ESP-IDF to a new release, you may need to run this installation process again. The exact commands used may also vary based on your bash environment.

Building boards such as the Saola is typically done through `make flash`. The default port is `tty.SLAB_USBtoUART`, which will only work on certain Mac setups. On most machines, both Mac and Linux, you will need to set the port yourself by running `ls /dev/tty.usb*` and selecting the one that only appears when your development board is plugged in. An example make command with the port setting is as follows:

```
make BOARD=espressif_saola_1_wrover flash PORT=/dev/tty.usbserial-1421120
```

## Debugging

The ESP32-S2 supports JTAG debugging over OpenOCD using a JLink or other probe hardware. The official tutorials can be found on the Espressif website [here](#), but they are mostly for the ESP32-S2 Kaluga, which has built-in debugging.

OpenOCD is automatically installed and added to your bash environment during the esp-idf installation and setup process. You can double check that it is installed by using `openocd --version`, as per the tutorial. Attach the JTAG probe pins according to the [instructions for JTAG debugging](#) on boards that do not contain an integrated debugger.

Once the debugger is connected physically, you must run OpenOCD with attached configuration files specifying the **interface** (your debugger probe) and either a **target** or a **board** (targets are for SoCs only, and can be used when a full board configuration file doesn't exist). You can find the path location of these files by checking the `OPENOCD_SCRIPTS` environmental variable by running `echo $OPENOCD_SCRIPTS` in bash. Interfaces will be in the `interface/` directory, and targets and boards in the `target/` and `board/` directories, respectively.

**Note:** Unfortunately, there are no board files for the esp32-s2 other than the Kaluga, and the included `target/esp32s2.cfg` target file will not work by default on the Jlink for boards like the Saola 1, as the default speed is incorrect. In addition, these files are covered under the GPL and cannot be included in Circuitpython. Thus, you must make a copy of the `esp32s2.cfg` file yourself and add the following line manually, under `transport select jtag` at the start of the file:

```
adapter_khz 1000
```

Once this is complete, your final OpenOCD command may look something like this:

```
openocd -f interface/jlink.cfg -f SOMEPATH/copied-esp32s2-saola-1.cfg
```

Where `SOMEPATH` is the location of your copied configuration file (this can be placed in the `port/boards` directory with a prefix to ignore it with `.gitignore`, for instance). Interface, target and board config files sourced from

espressif only need their paths from the \$OPENOCD\_SCRIPTS location, you don't need to include their full path. Once OpenOCD is running, connect to GDB with:

```
xtensa-esp32s2-elf-gdb build-espressif_saola_1_wrover/firmware.elf
```

And follow the Espressif GDB tutorial [instructions for connecting](#), or add them to your gdbinit:

```
target remote :3333
set remote hardware-watchpoint-limit 2
mon reset halt
flushregs
thb app_main
c
```

## LiteX (FPGA)

LiteX is a Python-based System on a Chip (SoC) designer for open source supported Field Programmable Gate Array (FPGA) chips. This means that the CPU core(s) and peripherals are not defined by the physical chip. Instead, they are loaded as separate “gateway”. Once this gateway is loaded, CircuitPython can be loaded on top of it to work as expected.

## Installation

You'll need dfu-util to install CircuitPython on the Fomu.

Make sure the foboot bootloader is updated. Instructions are here: <https://github.com/im-tomu/fomu-workshop/blob/master/docs/bootloader.rst>

Once you've updated the bootloader, you should know how to use dfu-util. It's pretty easy!

To install CircuitPython do:

```
dfu-util -D adafruit-circuitpython-fomu-en_US-<version>.dfu
```

It will install and then restart. CIRCUITPY should appear as it usually does and work the same.

## CircuitPython Port To The NXP i.MX RT10xx Series

This is a port of CircuitPython to the i.MX RT10xx series of chips.

## CircuitPython Port To The Nordic Semiconductor nRF52 Series

This is a port of CircuitPython to the Nordic Semiconductor nRF52 series of chips.

**NOTE:** There are board-specific READMEs that may be more up to date than the generic board-neutral documentation below.

## Flash

Some boards have UF2 bootloaders and can simply be flashed in the normal way, by copying firmware.uf2 to the BOOT drive.

For some boards, you can use the `flash` target:

```
make BOARD=pca10056 flash
```

## Segger Targets

Install the necessary tools to flash and debug using Segger:

[JLink Download](#)

[nrfjprog linux-32bit Download](#)

[nrfjprog linux-64bit Download](#)

[nrfjprog osx Download](#)

[nrfjprog win32 Download](#)

note: On Linux it might be required to link SEGGER's `libjlinkarm.so` inside `nrfjprog`'s folder.

## DFU Targets

run follow command to install `adafruit-nrfutil` from PyPi

```
$ pip3 install --user adafruit-nrfutil
```

**make flash** and **make sd** will not work with DFU targets. Hence, **dfu-gen** and **dfu-flash** must be used instead.

- `dfu-gen`: Generates a Firmware zip to be used by the DFU flash application.
- `dfu-flash`: Triggers the DFU flash application to upload the firmware from the generated Firmware zip file.

When enabled you have different options to test it:

- [NUS Console for Linux](#) (recommended)
- [WebBluetooth REPL](#) (experimental)

## RP2040

This port supports many development boards that utilize RP2040 chips. See <https://circuitpython.org/downloads> for all supported boards.

## Building

For build instructions see this guide: <https://learn.adafruit.com/building-circuitpython/>

## Port Specific modules

### Circuitpython on STM32

This port brings the ST Microelectronics STM32 series of MCUs to Circuitpython. STM32 chips have a wide range of capability, from <\$1 low power STM32F0s to dual-core STM32H7s running at 400+ MHz. Currently, only the F4, F7, and H7 families are supported, powered by the ARM Cortex M4 and M7 processors.

Refer to the ST Microelectronics website for more information on features sorted by family and individual chip lines: [st.com/en/microcontrollers-microprocessors/stm32-high-performance-mcus.html](http://st.com/en/microcontrollers-microprocessors/stm32-high-performance-mcus.html)

STM32 SoCs vary product-by-product in clock speed, peripheral capability, pin assignments, and their support within this port. Refer to **mpconfigport.mk** for a full list of enabled modules sorted by family.

### How this port is organized:

- **boards/** contains the configuration files for each development board and breakout available on the port, as well as system files and both shared and SoC-specific linker files. Board configuration includes a pin mapping of the board, oscillator information, board-specific build flags, and setup for OLED or TFT screens where applicable.
- **common-hal/** contains the port-specific module implementations, used by shared-module and shared-bindings.
- **packages/** contains package-specific pin bindings (LQFP100, BGA216, etc)
- **peripherals/** contains peripheral setup files and peripheral mapping information, sorted by family and sub-variant. Most files in this directory can be generated with the python scripts in **tools/**.
- **st-driver/** submodule for ST HAL and LL files generated via CubeMX. Shared with TinyUSB.
- **supervisor/** contains port-specific implementations of internal flash, serial and USB, as well as the **port.c** file, which initializes the port at startup.
- **tools/** python scripts for generating peripheral and pin mapping files in **peripherals/** and **board/**.

At the root level, refer to **mpconfigboard.h** and **mpconfigport.mk** for port specific settings and a list of enabled modules.

## Build instructions

Ensure your clone of Circuitpython is ready to build by following the [guide on the Adafruit Website](#). This includes installing the toolchain, synchronizing submodules, and running `mpy-cross`.

Once the one-time build tasks are complete, you can build at any time by navigating to the port directory:

```
$ cd ports/stm
```

To build for a specific circuitpython board, run:

```
$ make BOARD=feather_stm32f405_express
```

You may also build with certain flags available in the makefile, depending on your board and development goals. The following flags would enable debug information and correct flash locations for a pre-flashed UF2 bootloader:



```
$ make BOARD=feather_stm32f405_express DEBUG=1 UF2_BOOTLOADER=1
```

## USB connection

Connect your development board of choice to the host PC via the USB cable. Note that for most ST development boards such as the Nucleo and Discovery series, you must use a secondary OTG USB connector to access circuitpython, as the primary USB connector will be connected to a built-in ST-Link debugger rather than the chip itself.

In many cases, this ST-Link USB connector will **still need to be connected to power** for the chip to turn on - refer to your specific product manual for details.

## Flash the bootloader

Most ST development boards come with a built-in STLink programming and debugging probe accessible via USB. This programmer may show up as an MBED drive on the host PC, enabling simple drag and drop programming with a .bin file, or they may require a tool like [OpenOCD](#) or [StLink-org/stlink](#) to run flashing and debugging commands.

Many hobbyist and 3rd party development boards also expose SWD pins. These can be used with a cheap [stlink](#) debugger or other common programmers.

For non-ST products or users without a debugger, all STM32 boards in the high performance families (F4, F7 and H7) include a built-in DFU bootloader stored in ROM. This bootloader is accessed by ensuring the BOOT0 pin is held to a logic 1 and the BOOT1 pin is held to a logic 0 when the chip is reset ([ST Appnote AN2606](#)). Most chips hold BOOT low by default, so this can usually be achieved by running a jumper wire from 3.3V power to the BOOT0 pin, if it is exposed, or by flipping the appropriate switch or button as the chip is reset. Once the chip is started in DFU mode, BOOT0 no longer needs to be held high and can be released. An example is available in the [Feather STM32F405 guide](#).

Windows users will need to install [stm32cubeprog](#), while Mac and Linux users will need to install dfu-util with `brew install dfu-util` or `sudo apt-get install dfu-util`. More details are available in the [Feather F405 guide](#).

## Flashing the circuitpython image with DFU-Util

Ensure the board is in dfu mode by following the steps in the previous section. Then run:

```
$ make BOARD=feather_stm32F405_express flash
```

Alternatively, you can navigate to the build directory and run the raw dfu-util command:

```
dfu-util -a 0 --dfuse-address 0x08000000 -D firmware.bin
```

## Accessing the board

Connecting the board to the PC via the USB cable will allow code to be uploaded to the CIRCUITPY volume.

Circuitpython exposes a CDC virtual serial connection for REPL access and debugging. Connecting to it from OSX will look something like this:

```
screen /dev/tty.usbmodem14111201 115200
```

You may also use a program like [mu](#) to assist with REPL access.

### 1.8.3 Troubleshooting

From time to time, an error occurs when working with CircuitPython. Here are a variety of errors that can happen, what they mean and how to fix them.

#### File system issues

If your host computer starts complaining that your CIRCUITPY drive is corrupted or files cannot be overwritten or deleted, then you will have to erase it completely. When CircuitPython restarts it will create a fresh empty CIRCUITPY filesystem.

This often happens on Windows when the CIRCUITPY disk is not safely ejected before being reset by the button or being disconnected from USB. This can also happen on Linux and Mac OSX but it's less likely.

**Caution:** To erase and re-create CIRCUITPY (for example, to correct a corrupted filesystem), follow one of the procedures below. It's important to note that **any files stored on the CIRCUITPY drive will be erased**.

**For boards with CIRCUITPY stored on a separate SPI flash chip, such as Feather M0 Express, Metro M0 Express and Circuit Playground Express:**

1. Download the appropriate flash .erase uf2 from [the Adafruit\\_SPIFlash repo](#).
2. Double-click the reset button.
3. Copy the appropriate .uf2 to the xxxBOOT drive.
4. The on-board NeoPixel will turn blue, indicating the erase has started.
5. After about 15 seconds, the NexaPixel will start flashing green. If it flashes red, the erase failed.
6. Double-click again and load the appropriate [CircuitPython .uf2](#).

**For boards without SPI flash, such as Feather M0 Proto, Gemma M0 and, Trinket M0:**

1. Download the appropriate erase .uf2 from [the Learn repo](#).
2. Double-click the reset button.
3. Copy the appropriate .uf2 to the xxxBOOT drive.
4. The boot LED will start pulsing again, and the xxxBOOT drive will appear again.
5. Load the appropriate [CircuitPython .uf2](#).

#### ValueError: Incompatible .mpy file.

This error occurs when importing a module that is stored as a mpy binary file (rather than a py text file) that was generated by a different version of CircuitPython than the one it's being loaded into. Most versions are compatible but, rarely they aren't. In particular, the mpy binary format changed between CircuitPython versions 1.x and 2.x, and will change again between 2.x and 3.x.

So, for instance, if you just upgraded to CircuitPython 2.x from 1.x you'll need to download a newer version of the library that triggered the error on `import`. They are all available in the [Adafruit bundle](#) and the [Community bundle](#). Make sure to download a version with 2.0.0 or higher in the filename.

## 1.8.4 Additional CircuitPython Libraries and Drivers on GitHub

These are libraries and drivers available in separate GitHub repos. They are designed for use with CircuitPython and may or may not work with [MicroPython](#).

### Adafruit CircuitPython Library Bundle

We provide a bundle of all our libraries to ease installation of drivers and their dependencies. The bundle is primarily geared to the Adafruit Express line of boards which feature a relatively large external flash. With Express boards, it's easy to copy them all onto the filesystem. However, if you don't have enough space simply copy things over as they are needed.

- The Adafruit bundles are available on GitHub: [https://github.com/adafruit/Adafruit\\_CircuitPython\\_Bundle/releases](https://github.com/adafruit/Adafruit_CircuitPython_Bundle/releases).
- Documentation for the bundle, which includes links to documentation for all libraries, is available here: <https://circuitpython.readthedocs.io/projects/bundle/en/latest/>.

### CircuitPython Community Library Bundle

This bundle contains non-Adafruit sponsored libraries, that are written and submitted by members of the community.

- The Community bundles are available on GitHub: [https://github.com/adafruit/CircuitPython\\_Community\\_Bundle/releases](https://github.com/adafruit/CircuitPython_Community_Bundle/releases).
- Documentation is not available on ReadTheDocs at this time. See each library for any included documentation.

## 1.8.5 Design Guide

This guide covers a variety of development practices for CircuitPython core and library APIs. These APIs are both [built-into CircuitPython](#) and those that are [distributed on GitHub](#) and in the [Adafruit](#) and [Community](#) bundles. Consistency with these practices ensures that beginners can learn a pattern once and apply it throughout the CircuitPython ecosystem.

### Start libraries with the cookiecutter

Cookiecutter is a tool that lets you bootstrap a new repo based on another repo. We've made one [here](#) for CircuitPython libraries that include configs for Travis CI and ReadTheDocs along with a setup.py, license, code of conduct and readme.

Cookiecutter will provide a series of prompts relating to the library and then create a new directory with all of the files. See [the CircuitPython cookiecutter README](#) for more details.

### Module Naming

Adafruit funded libraries should be under the [adafruit organization](#) and have the format `Adafruit_CircuitPython_<name>` and have a corresponding `adafruit_<name>` directory (aka package) or `adafruit_<name>.py` file (aka module).

If the name would normally have a space, such as “Thermal Printer”, use an underscore instead (“Thermal\_Printer”). This underscore will be used everywhere even when the separation between “adafruit” and “circuitpython” is done with a -. Use the underscore in the cookiecutter prompts.

Community created libraries should have the repo format `CircuitPython_<name>` and not have the `adafruit_` module or package prefix.

Both should have the CircuitPython repository topic on GitHub.

## Terminology

As our Code of Conduct states, we strive to use “welcoming and inclusive language.” Whether it is in documentation or in code, the words we use matter. This means we disfavor language that due to historical and social context can make community members and potential community members feel unwelcome.

There are specific terms to avoid except where technical limitations require it. While specific cases may call for other terms, consider using these suggested terms first:

Preferred	Deprecated
Main (device)	Master
Peripheral	Slave
Sensor	
Secondary (device)	
Denylist	Blacklist
Allowlist	Whitelist

Note that “technical limitations” refers e.g., to the situation where an upstream library or URL has to contain those substrings in order to work. However, when it comes to documentation and the names of parameters and properties in CircuitPython, we will use alternate terms even if this breaks tradition with past practice.

## Lifetime and ContextManagers

A driver should be initialized and ready to use after construction. If the device requires deinitialization, then provide it through `deinit()` and also provide `__enter__` and `__exit__` to create a context manager usable with `with`.

For example, a user can then use `deinit()`:

```
import digitalio
import board
import time

led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT

for i in range(10):
    led.value = True
    time.sleep(0.5)

    led.value = False
    time.sleep(0.5)
led.deinit()
```

This will deinit the underlying hardware at the end of the program as long as no exceptions occur.

Alternatively, using a `with` statement ensures that the hardware is deinitialized:

```
import digitalio
import board
import time
```

(continues on next page)

(continued from previous page)

```

with digitalio.DigitalInOut(board.D13) as led:
    led.direction = digitalio.Direction.OUTPUT

    for i in range(10):
        led.value = True
        time.sleep(0.5)

    led.value = False
    time.sleep(0.5)

```

Python's `with` statement ensures that the deinit code is run regardless of whether the code within the `with` statement executes without exceptions.

For small programs like the examples this isn't a major concern because all user usable hardware is reset after programs are run or the REPL is run. However, for more complex programs that may use hardware intermittently and may also handle exceptions on their own, deinitializing the hardware using a `with` statement will ensure hardware isn't enabled longer than needed.

## Verify your device

Whenever possible, make sure device you are talking to is the device you expect. If not, raise a `RuntimeError`. Beware that I2C addresses can be identical on different devices so read registers you know to make sure they match your expectation. Validating this upfront will help catch mistakes.

## Getters/Setters

When designing a driver for a device, use properties for device state and use methods for sequences of abstract actions that the device performs. State is a property of the device as a whole that exists regardless of what the code is doing. This includes things like temperature, time, sound, light and the state of a switch. For a more complete list see the sensor properties bullet below.

Another way to separate state from actions is that state is usually something the user can sense themselves by sight or feel for example. Actions are something the user can watch. The device does this and then this.

Making this separation clear to the user will help beginners understand when to use what.

Here is more info on properties from [Python](#).

## Exceptions and asserts

Raise an appropriate [Exception](#), along with a useful message, whenever a critical test or other condition fails.

Example:

```

if not 0 <= pin <= 7:
    raise ValueError("Pin number must be 0-7.")

```

If memory is constrained and a more compact method is needed, use [The `assert` statement](#) instead.

Example:

```

assert 0 <= pin <= 7, "Pin number must be 0-7."

```

## Design for compatibility with CPython

CircuitPython is aimed to be one's first experience with code. It will be the first step into the world of hardware and software. To ease one's exploration out from this first step, make sure that functionality shared with CPython shares the same API. It doesn't need to be the full API it can be a subset. However, do not add non-CPython APIs to the same modules. Instead, use separate non-CPython modules to add extra functionality. By distinguishing API boundaries at modules you increase the likelihood that incorrect expectations are found on import and not randomly during runtime.

When adding a new module for additional functionality related to a CPython module do NOT simply prefix it with `u`. This is not a large enough differentiation from CPython. This is the MicroPython convention and they use `u*` modules interchangeably with the CPython name. This is confusing. Instead, think up a new name that is related to the extra functionality you are adding.

For example, storage mounting and unmounting related functions were moved from `uos` into a new `storage` module. Terminal related functions were moved into `multiterminal`. These names better match their functionality and do not conflict with CPython names. Make sure to check that you don't conflict with CPython libraries too. That way we can port the API to CPython in the future.

## Example

When adding extra functionality to CircuitPython to mimic what a normal operating system would do, either copy an existing CPython API (for example file writing) or create a separate module to achieve what you want. For example, mounting and unmount drives is not a part of CPython so it should be done in a module, such as a new `storage` module, that is only available in CircuitPython. That way when someone moves the code to CPython they know what parts need to be adapted.

## Document inline

Whenever possible, document your code right next to the code that implements it. This makes it more likely to stay up to date with the implementation itself. Use Sphinx's automodule to format these all nicely in ReadTheDocs. The cookiecutter helps set these up.

Use [Sphinx flavor rST](#) for markup.

Lots of documentation is a good thing but it can take a lot of space. To minimize the space used on disk and on load, distribute the library as both `.py` and `.mpy`, MicroPython and CircuitPython's bytecode format that omits comments.

## Module description

After the license comment:

```
"""
`<module name>` - <Short description>
=====
<Longer description.>
"""
```

## Class description

At the class level document what class does and how to initialize it:

```
class DS3231:
    """DS3231 real-time clock.

    :param ~busio.I2C i2c_bus: The I2C bus the DS3231 is connected to.
    :param int address: The I2C address of the device.
    """

    def __init__(self, i2c_bus, address=0x40):
        self._i2c = i2c_bus
```

Renders as:

```
class DS3231 (i2c_bus, address=64)
    DS3231 real-time clock.
```

### Parameters

- **i2c\_bus** ([I2C](#)) – The I2C bus the DS3231 is connected to.
- **address** ([int](#)) – The I2C address of the device.

## Attributes

Attributes are state on objects. (See [Getters/Setters](#) above for more discussion about when to use them.) They can be defined internally in a number of different ways. Each approach is enumerated below with an explanation of where the comment goes.

Regardless of how the attribute is implemented, it should have a short description of what state it represents including the type, possible values and/or units. It should be marked as (read-only) or (write-only) at the end of the first line for attributes that are not both readable and writable.

## Instance attributes

Comment comes from after the assignment:

```
def __init__(self, drive_mode):
    self.drive_mode = drive_mode
    """
    The pin drive mode. One of:

    - `digitalio.DriveMode.PUSH_PULL`
    - `digitalio.DriveMode.OPEN_DRAIN`
    """
```

Renders as:

### drive\_mode

The pin drive mode. One of:

- `digitalio.DriveMode.PUSH_PULL`
- `digitalio.DriveMode.OPEN_DRAIN`

## Property description

Comment comes from the getter:

```
@property
def datetime(self):
    """The current date and time as a `time.struct_time`."""
    return self.datetime_register

@datetime.setter
def datetime(self, value):
    pass
```

Renders as:

### **datetime**

The current date and time as a `time.struct_time`.

Read-only example:

```
@property
def temperature(self):
    """
    The current temperature in degrees Celsius. (read-only)

    The device may require calibration to get accurate readings.
    """
    return self._read(TEMPERATURE)
```

Renders as:

### **temperature**

The current temperature in degrees Celsius. (read-only)

The device may require calibration to get accurate readings.

## Data descriptor description

Comment is after the definition:

```
lost_power = i2c_bit.RWBit(0x0f, 7)
"""True if the device has lost power since the time was set."""
```

Renders as:

### **lost\_power**

True if the device has lost power since the time was set.



## Method description

First line after the method definition:

```
def turn_right(self, degrees):
    """Turns the bot ``degrees`` right.

    :param float degrees: Degrees to turn right
    """
```

Renders as:

**turn\_right** (*degrees*)  
 Turns the bot degrees right.

**Parameters** **degrees** (*float*) – Degrees to turn right

## Use BusDevice

**BusDevice** is an awesome foundational library that manages talking on a shared I2C or SPI device for you. The devices manage locking which ensures that a transfer is done as a single unit despite CircuitPython internals and, in the future, other Python threads. For I2C, the device also manages the device address. The SPI device, manages baudrate settings, chip select line and extra post-transaction clock cycles.

## I2C Example

```
from adafruit_bus_device import i2c_device

DEVICE_DEFAULT_I2C_ADDR = 0x42

class Widget:
    """A generic widget."""

    def __init__(self, i2c, address=DEVICE_DEFAULT_I2C_ADDR):
        self.i2c_device = i2c_device.I2CDevice(i2c, address)
        self.buf = bytearray(1)

    @property
    def register(self):
        """Widget's one register."""
        with self.i2c_device as i2c:
            i2c.writeto(b'0x00')
            i2c.readfrom_into(self.buf)
        return self.buf[0]
```

## SPI Example

```
from adafruit_bus_device import spi_device

class SPIWidget:
    """A generic widget with a weird baudrate."""

    def __init__(self, spi, chip_select):
        # chip_select is a pin reference such as board.D10.
        self.spi_device = spi_device.SPIDevice(spi, chip_select, baudrate=12345)
        self.buf = bytearray(1)

    @property
    def register(self):
        """Widget's one register."""
        with self.spi_device as spi:
            spi.write(b'0x00')
            spi.readinto(self.buf)
        return self.buf[0]
```

## Use composition

When writing a driver, take in objects that provide the functionality you need rather than taking their arguments and constructing them yourself or subclassing a parent class with functionality. This technique is known as composition and leads to code that is more flexible and testable than traditional inheritance.

### See also:

[Wikipedia](#) has more information on “dependency inversion”.

For example, if you are writing a driver for an I2C device, then take in an I2C object instead of the pins themselves. This allows the calling code to provide any object with the appropriate methods such as an I2C expansion board.

Another example is to expect a *DigitalInOut* for a pin to toggle instead of a *Pin* from *board*. Taking in the *Pin* object alone would limit the driver to pins on the actual microcontroller instead of pins provided by another driver such as an IO expander.

## Lots of small modules

CircuitPython boards tend to have a small amount of internal flash and a small amount of ram but large amounts of external flash for the file system. So, create many small libraries that can be loaded as needed instead of one large file that does everything.

## Speed second

Speed isn’t as important as API clarity and code size. So, prefer simple APIs like properties for state even if it sacrifices a bit of speed.

## Avoid allocations in drivers

Although Python doesn't require managing memory, it's still a good practice for library writers to think about memory allocations. Avoid them in drivers if you can because you never know how much something will be called. Fewer allocations means less time spent cleaning up. So, where you can, prefer bytearray buffers that are created in `__init__` and used throughout the object with methods that read or write into the buffer instead of creating new objects. Unified hardware API classes such as `busio.SPI` are design to read and write to subsections of buffers.

It's ok to allocate an object to return to the user. Just beware of causing more than one allocation per call due to internal logic.

**However**, this is a memory tradeoff so do not do it for large or rarely used buffers.

## Examples

### `struct.pack`

Use `struct.pack_into` instead of `struct.pack`.

### Use of MicroPython `const ()`

The MicroPython `const ()` feature, as discussed in [this forum post](#), and in [this issue thread](#), provides some optimizations that can be useful on smaller, memory constrained devices. However, when using `const ()`, keep in mind these general guide lines:

- Always use via an import, ex: `from micropython import const`
- Limit use to global (module level) variables only.
- If user will not need access to variable, prefix name with a leading underscore, ex: `__SOME_CONST`.

## Sensor properties and units

The [Adafruit Unified Sensor Driver Arduino library](#) has a [great list](#) of measurements and their units. Use the same ones including the property name itself so that drivers can be used interchangeably when they have the same properties.

Property name	Python type	Units
acceleration	(float, float, float)	x, y, z meter per second per second
magnetic	(float, float, float)	x, y, z micro-Tesla (uT)
orientation	(float, float, float)	x, y, z degrees
gyro	(float, float, float)	x, y, z radians per second
temperature	float	degrees centigrade
CO2	float	measured CO2 in ppm
eCO2	float	equivalent/estimated CO2 in ppm (estimated from some other measurement)
TVOC	float	Total Volatile Organic Compounds in ppb
distance	float	centimeters
proximity	int	non-unit-specific proximity values (monotonic but not actual distance)
light	float	non-unit-specific light levels (should be monotonic but is not lux)
lux	float	SI lux
pressure	float	hectopascal (hPa)
relative_humidity	float	percent
current	float	milliamps (mA)
voltage	float	volts (V)
color	int	RGB, eight bits per channel (0xff0000 is red)
alarm	(time.struct, str)	Sample alarm time and string to characterize frequency such as “hourly”
datetime	time.struct	date and time
duty_cycle	int	16-bit PWM duty cycle (regardless of output resolution)
frequency	int	Hertz
value	bool	Digital logic
value	int	16-bit Analog value, unit-less
weight	float	grams (g)
sound_level	float	non-unit-specific sound level (monotonic but not actual decibels)

## Adding native modules

The Python API for a new module should be defined and documented in `shared-bindings` and define an underlying C API. If the implementation is port-agnostic or relies on underlying APIs of another module, the code should live in `shared-module`. If it is port specific then it should live in `common-hal` within the port’s folder. In either case, the file and folder structure should mimic the structure in `shared-bindings`.

To test your native modules or core enhancements, follow these [Adafruit Learning Guides](#) for building local firmware to flash onto your device(s):

[SAMD21 - Build Firmware Learning Guide](#)

[ESP8266 - Build Firmware Learning Guide](#)

## MicroPython compatibility

Keeping compatibility with MicroPython isn't a high priority. It should be done when it's not in conflict with any of the above goals.

We love CircuitPython and would love to see it come to more microcontroller platforms. Since 3.0 we've reworked CircuitPython to make it easier than ever to add support. While there are some major differences between ports, this page covers the similarities that make CircuitPython what it is and how that core fits into a variety of microcontrollers.

### 1.8.6 Architecture

There are three core pieces to CircuitPython:

The first is the Python VM that the awesome MicroPython devs have created. These VMs are written to be portable so there is not much needed when moving to a different microcontroller, especially if it is ARM based.

The second is the infrastructure around those VMs which provides super basic operating system functionality such as initializing hardware, running USB, prepping file systems and automatically running user code on boot. In CircuitPython we've dubbed this component the supervisor because it monitors and facilitates the VMs which run user Python code. Porting involves the supervisor because many of the tasks it does while interfacing with the hardware. Once complete, the REPL works and debugging can migrate to a Python based approach rather than C.

The third core piece is the plethora of low level APIs that CircuitPython provides as the foundation for higher level libraries including device drivers. These APIs are called from within the running VMs through the Python interfaces defined in `shared-bindings`. These bindings rely on the underlying `common_hal` C API to implement the functionality needed for the Python API. By splitting the two, we work to ensure standard functionality across which means that libraries and examples apply across ports with minimal changes.

### 1.8.7 Porting

#### Step 1: Getting building

The first step to porting to a new microcontroller is getting a build running. The primary goal of it should be to get `main.c` compiling with the assistance of the `supervisor/supervisor.mk` file. Port specific code should be isolated to the port's directory (in the top level until the `ports` directory is present). This includes the Makefile and any C library resources. Make sure these resources are compatible with the MIT License of the rest of the code!

Circuitpython has a number of modules enabled by default in `py/circuitpy_mpconfig.mk`. Most of these modules will need to be disabled in `mpconfigboard.mk` during the early stages of a port in order for it to compile. As the port progresses in module support, this list can be pruned down as a natural "TODO" list. An example minimal build list is shown below:

```
# These modules are implemented in ports/<port>/common-hal:

# Typically the first module to create
CIRCUITPY_MICROCONTROLLER = 0
# Typically the second module to create
CIRCUITPY_DIGITALIO = 0
# Other modules:
CIRCUITPY_ANALOGIO = 0
CIRCUITPY_BUSIO = 0
CIRCUITPY_COUNTIO = 0
CIRCUITPY_NEOPixel_WRITE = 0
CIRCUITPY_PULSEIO = 0
CIRCUITPY_OS = 0
```

(continues on next page)

(continued from previous page)

```
CIRCUITPY_NVM = 0
CIRCUITPY_AUDIOBUSIO = 0
CIRCUITPY_AUDIOIO = 0
CIRCUITPY_ROTARYIO = 0
CIRCUITPY_RTC = 0
CIRCUITPY_SDCARDIO = 0
CIRCUITPY_FRAMEBUFFERIO = 0
CIRCUITPY_FREQUENCYIO = 0
CIRCUITPY_I2CPERIPHERAL = 0
# Requires SPI, PulseIO (stub ok):
CIRCUITPY_DISPLAYIO = 0

# These modules are implemented in shared-module/ - they can be included in
# any port once their prerequisites in common-hal are complete.
# Requires DigitalIO:
CIRCUITPY_BITBANGIO = 0
# Requires DigitalIO
CIRCUITPY_GAMEPAD = 0
# Requires neopixel_write or SPI (dotstar)
CIRCUITPY_PIXELBUF = 0
# Requires OS
CIRCUITPY_RANDOM = 0
# Requires OS, filesystem
CIRCUITPY_STORAGE = 0
# Requires Microcontroller
CIRCUITPY_TOUCHIO = 0
# Requires USB
CIRCUITPY_USB_HID = 0
CIRCUITPY_USB_MIDI = 0
# Does nothing without I2C
CIRCUITPY_REQUIRE_I2C_PULLUPS = 0
# No requirements, but takes extra flash
CIRCUITPY_ULAB = 0
```

## Step 2: Init

Once your build is setup, the next step should be to get your clocks going as you expect from the supervisor. The supervisor calls `port_init` to allow for initialization at the beginning of main. This function also has the ability to request a safe mode state which prevents the supervisor from running user code while still allowing access to the REPL and other resources.

The core port initialization and reset methods are defined in `supervisor/port.c` and should be the first to be implemented. It's required that they be implemented in the `supervisor` directory within the port directory. That way, they are always in the expected place.

The supervisor also uses three linker variables, `_ezero`, `_estack` and `_ebss` to determine memory layout for stack overflow checking.

### Step 3: REPL

Getting the REPL going is a huge step. It involves a bunch of initialization to be done correctly and is a good sign you are well on your porting way. To get the REPL going you must implement the functions and definitions from `supervisor/serial.h` with a corresponding `supervisor/serial.c` in the port directory. This involves sending and receiving characters over some sort of serial connection. It could be UART or USB for example.

## 1.8.8 Adding `*io` support to other ports

`digitalio` provides a well-defined, cross-port hardware abstraction layer built to support different devices and their drivers. It's backed by the Common HAL, a C api suitable for supporting different hardware in a similar manner. By sharing this C api, developers can support new hardware easily and cross-port functionality to the new hardware.

These instructions also apply to `analogio`, `busio`, `pulseio` and `touchio`. Most drivers depend on `analogio`, `digitalio` and `busio` so start with those.

### File layout

Common HAL related files are found in these locations:

- `shared-bindings` Shared home for the Python <-> C bindings which includes inline RST documentation for the created interfaces. The common hal functions are defined in the `.h` files of the corresponding C files.
- `shared-modules` Shared home for C code built on the Common HAL and used by all ports. This code only uses `common_hal` methods defined in `shared-bindings`.
- `<port>/common-hal` Port-specific implementation of the Common HAL.

Each folder has the substructure of / and they should match 1:1. `__init__.c` is used for module globals that are not classes (similar to `__init__.py`).

### Adding support

### Modifying the build

The first step is to hook the `shared-bindings` into your build for the modules you wish to support. Here's an example of this step for the `atmel-samd/Makefile`:

```
SRC_BINDINGS = \
    board/__init__.c \
    microcontroller/__init__.c \
    microcontroller/Pin.c \
    analogio/__init__.c \
    analogio/AnalogIn.c \
    analogio/AnalogOut.c \
    digitalio/__init__.c \
    digitalio/DigitalInOut.c \
    pulseio/__init__.c \
    pulseio/PulseIn.c \
    pulseio/PulseOut.c \
    pulseio/PWMOut.c \
    busio/__init__.c \
    busio/I2C.c \
    busio/SPI.c \
    busio/UART.c \
```

(continues on next page)

(continued from previous page)

```
neopixel_write/__init__.c \
time/__init__.c \
usb_hid/__init__.c \
usb_hid/Device.c

SRC_BINDINGS_EXPANDED = $(addprefix shared-bindings/, $(SRC_BINDINGS)) \
                        $(addprefix common-hal/, $(SRC_BINDINGS))

# Add the resulting objects to the full list
OBJ += $(addprefix $(BUILD)/, $(SRC_BINDINGS_EXPANDED:.c=.o))
# Add the sources for QSTR generation
SRC_QSTR += $(SRC_C) $(SRC_BINDINGS_EXPANDED) $(STM_SRC_C)
```

The Makefile defines the modules to build and adds the sources to include the shared-bindings version and the common-hal version within the port specific directory. You may comment out certain subfolders to reduce the number of modules to add but don't comment out individual classes. It won't compile then.

## Hooking the modules in

Built in modules are typically defined in `mpconfigport.h`. To add support you should have something like:

```
extern const struct _mp_obj_module_t microcontroller_module;
extern const struct _mp_obj_module_t analogio_module;
extern const struct _mp_obj_module_t digitalio_module;
extern const struct _mp_obj_module_t pulseio_module;
extern const struct _mp_obj_module_t busio_module;
extern const struct _mp_obj_module_t board_module;
extern const struct _mp_obj_module_t time_module;
extern const struct _mp_obj_module_t neopixel_write_module;

#define MICROPY_PORT_BUILTIN_MODULES \
    { MP_OBJ_NEW_QSTR(MP_QSTR_microcontroller), (mp_obj_t)&microcontroller_module }, \
    { MP_OBJ_NEW_QSTR(MP_QSTR_analogio), (mp_obj_t)&analogio_module }, \
    { MP_OBJ_NEW_QSTR(MP_QSTR_digitalio), (mp_obj_t)&digitalio_module }, \
    { MP_OBJ_NEW_QSTR(MP_QSTR_pulseio), (mp_obj_t)&pulseio_module }, \
    { MP_OBJ_NEW_QSTR(MP_QSTR_busio), (mp_obj_t)&busio_module }, \
    { MP_OBJ_NEW_QSTR(MP_QSTR_board), (mp_obj_t)&board_module }, \
    { MP_OBJ_NEW_QSTR(MP_QSTR_time), (mp_obj_t)&time_module }, \
    { MP_OBJ_NEW_QSTR(MP_QSTR_neopixel_write), (mp_obj_t)&neopixel_write_module } \
```

## Implementing the Common HAL

At this point in the port, nothing will compile yet, because there's still work to be done to fix missing sources, compile issues, and link issues. I suggest start with a common-hal directory from another port that implements it such as `atmel-samd` or `esp8266`, deleting the function contents and stubbing out any return statements. Once that is done, you should be able to compile cleanly and import the modules, but nothing will work (though you are getting closer).

The last step is actually implementing each function in a port specific way. I can't help you with this. :-) If you have any questions how a Common HAL function should work then see the corresponding `.h` file in `shared-bindings`.



## Testing

Woohoo! You are almost done. After you implement everything, lots of drivers and sample code should just work. There are a number of drivers and examples written for Adafruit's Feather ecosystem. Here are places to start:

- [Adafruit repos with CircuitPython topic](#)
- [Adafruit driver bundle](#)

## 1.8.9 MicroPython libraries

### Python standard libraries and micro-libraries

The libraries below are inherited from MicroPython. They are similar to the standard Python libraries with the same name. They implement a subset of or a variant of the corresponding standard Python library.

CircuitPython's long-term goal is that code written in CircuitPython using Python standard libraries will be runnable on CPython without changes.

These libraries are not enabled on CircuitPython builds with limited flash memory, usually on non-Express builds: `binascii`, `errno`, `json`, `re`.

These libraries are not currently enabled in any CircuitPython build, but may be in the future, with the `u` prefix dropped: `uctypes`, `uhashlib`, `uzlib`.

### Builtin functions and exceptions

All builtin functions and exceptions are described here. They are also available via `builtins` module.

### Functions and types

Not all of these functions and types are turned on in all CircuitPython ports, for space reasons.

**`abs()`**

**`all()`**

**`any()`**

**`bin()`**

**`class bool`**

**`class bytearray`**

**`class bytes`**

See CPython documentation: *[bytes](#)*.

**`callable()`**

**`chr()`**

**`classmethod()`**

**`compile()`**

**`class complex`**

**delattr** (*obj*, *name*)

The argument *name* should be a string, and this function deletes the named attribute from the object given by *obj*.

**class dict**

**dir** ()

**divmod** ()

**enumerate** ()

**eval** ()

**exec** ()

**filter** ()

**class float**

**class frozenset**

*frozenset* () is not enabled on non-Express CircuitPython boards.

**getattr** ()

**globals** ()

**hasattr** ()

**hash** ()

**hex** ()

**id** ()

**input** ()

**class int**

**classmethod from\_bytes** (*bytes*, *byteorder*)

In CircuitPython, *byteorder* parameter must be positional (this is compatible with CPython).

**to\_bytes** (*size*, *byteorder*)

In CircuitPython, *byteorder* parameter must be positional (this is compatible with CPython).

**isinstance** ()

**issubclass** ()

**iter** ()

**len** ()

**class list**

**locals** ()

**map** ()

**max** ()

**class memoryview**

**min** ()

**next** ()

**class object**

`oct()`

`open()`

`ord()`

`pow()`

`print()`

`property()`

`range()`

`repr()`

`reversed()`

*`reversed()`* is not enabled on non-Express CircuitPython boards.

`round()`

`class set`

`setattr()`

`class slice`

The *slice* builtin is the type that slice objects have.

`sorted()`

`staticmethod()`

`class str`

`sum()`

`super()`

`class tuple`

`type()`

`zip()`

## Exceptions

`exception AssertionError`

`exception AttributeError`

`exception Exception`

`exception ImportError`

`exception IndexError`

`exception KeyboardInterrupt`

`exception KeyError`

`exception MemoryError`

`exception NameError`

`exception NotImplementedError`

**exception OSError**

See CPython documentation: [OSError](#). CircuitPython doesn't implement the `errno` attribute, instead use the standard way to access exception arguments: `exc.args[0]`.

**exception RuntimeError**

**exception ReloadException**

[ReloadException](#) is used internally to deal with soft restarts.

**exception StopIteration**

**exception SyntaxError**

**exception SystemExit**

See CPython documentation: [SystemExit](#).

**exception TypeError**

See CPython documentation: [TypeError](#).

**exception ValueError**

**exception ZeroDivisionError**

**uheapq – heap queue algorithm**

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [heapq](#).*

This module implements the heap queue algorithm.

A heap queue is simply a list that has its elements stored in a certain way.

## Functions

`uheapq.heappush(heap, item)`

Push the `item` onto the heap.

`uheapq.heappop(heap)`

Pop the first item from the heap, and return it. Raises `IndexError` if heap is empty.

`uheapq.heapify(x)`

Convert the list `x` into a heap. This is an in-place operation.

## array – arrays of numeric data

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [array](#).*

Supported format codes: b, B, h, H, i, I, l, L, q, Q, f, d (the latter 2 depending on the floating-point support).

### Classes

**class** `array.array`(*typecode*[, *iterable*])

Create array with elements of given type. Initial contents of the array are given by an *iterable*. If it is not provided, an empty array is created.

**append**(*val*)

Append new element *val* to the end of array, growing it.

**extend**(*iterable*)

Append new elements as contained in *iterable* to the end of array, growing it.

## binascii – binary/ASCII conversions

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [binascii](#).*

This module implements conversions between binary data and various encodings of it in ASCII form (in both directions).

### Functions

`binascii.hexlify`(*data*[, *sep*])

Convert binary data to hexadecimal representation. Returns bytes string.

---

#### Difference to CPython

If additional argument, *sep* is supplied, it is used as a separator between hexadecimal values.

---

`binascii.unhexlify`(*data*)

Convert hexadecimal data to binary representation. Returns bytes string. (i.e. inverse of `hexlify`)

`binascii.a2b_base64`(*data*)

Decode base64-encoded data, ignoring invalid characters in the input. Conforms to [RFC 2045 s.6.8](#). Returns a bytes object.

`binascii.b2a_base64`(*data*)

Encode binary data in base64 format, as in [RFC 3548](#). Returns the encoded data followed by a newline character, as a bytes object.

## `collections` – collection and container types

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: `collections`.*

This module implements advanced collection and container types to hold/accumulate various objects.

### Classes

`collections.namedtuple` (*name*, *fields*)

This is factory function to create a new namedtuple type with a specific name and set of fields. A namedtuple is a subclass of tuple which allows to access its fields not just by numeric index, but also with an attribute access syntax using symbolic field names. Fields is a sequence of strings specifying field names. For compatibility with CPython it can also be a string with space-separated field named (but this is less efficient). Example of use:

```
from collections import namedtuple

MyTuple = namedtuple("MyTuple", ("id", "name"))
t1 = MyTuple(1, "foo")
t2 = MyTuple(2, "bar")
print(t1.name)
assert t2.name == t2[1]
```

`collections.OrderedDict` (...)

dict type subclass which remembers and preserves the order of keys added. When ordered dict is iterated over, keys/items are returned in the order they were added:

```
from collections import OrderedDict

# To make benefit of ordered keys, OrderedDict should be initialized
# from sequence of (key, value) pairs.
d = OrderedDict([("z", 1), ("a", 2)])
# More items can be added as usual
d["w"] = 5
d["b"] = 3
for k, v in d.items():
    print(k, v)
```

Output:

```
z 1
a 2
w 5
b 3
```

## errno – system error codes

This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [errno](#).

This module provides access to symbolic error codes for `OSError` exception.

### Constants

#### EEXIST, EAGAIN, etc.

Error codes, based on ANSI C/POSIX standard. All error codes start with “E”. Errors are usually accessible as `exc.args[0]` where `exc` is an instance of `OSError`. Usage example:

```
try:
    os.mkdir("my_dir")
except OSError as exc:
    if exc.args[0] == errno.EEXIST:
        print("Directory already exists")
```

#### errno.errorcode

Dictionary mapping numeric error codes to strings with symbolic error code (see above):

```
>>> print(errno.errorcode[errno.EEXIST])
EEXIST
```

## gc – control the garbage collector

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [gc](#).

### Functions

#### gc.enable()

Enable automatic garbage collection.

#### gc.disable()

Disable automatic garbage collection. Heap memory can still be allocated, and garbage collection can still be initiated manually using `gc.collect()`.

#### gc.collect()

Run a garbage collection.

#### gc.mem\_alloc()

Return the number of bytes of heap RAM that are allocated.

---

### Difference to CPython

This function is a MicroPython extension.

---

`gc.mem_free()`

Return the number of bytes of available heap RAM, or -1 if this amount is not known.

---

#### Difference to CPython

This function is a MicroPython extension.

---

`gc.threshold([amount])`

Set or query the additional GC allocation threshold. Normally, a collection is triggered only when a new allocation cannot be satisfied, i.e. on an out-of-memory (OOM) condition. If this function is called, in addition to OOM, a collection will be triggered each time after *amount* bytes have been allocated (in total, since the previous time such an amount of bytes have been allocated). *amount* is usually specified as less than the full heap size, with the intention to trigger a collection earlier than when the heap becomes exhausted, and in the hope that an early collection will prevent excessive memory fragmentation. This is a heuristic measure, the effect of which will vary from application to application, as well as the optimal value of the *amount* parameter.

Calling the function without argument will return the current value of the threshold. A value of -1 means a disabled allocation threshold.

---

#### Difference to CPython

This function is a MicroPython extension. CPython has a similar function - `set_threshold()`, but due to different GC implementations, its signature and semantics are different.

---

## hashlib – hashing algorithms

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [hashlib](#).*

This module implements binary data hashing algorithms. The exact inventory of available algorithms depends on a board. Among the algorithms which may be implemented:

- SHA256 - The current generation, modern hashing algorithm (of SHA2 series). It is suitable for cryptographically-secure purposes. Included in the MicroPython core and any board is recommended to provide this, unless it has particular code size constraints.
- SHA1 - A previous generation algorithm. Not recommended for new usages, but SHA1 is a part of number of Internet standards and existing applications, so boards targeting network connectivity and interoperability will try to provide this.
- MD5 - A legacy algorithm, not considered cryptographically secure. Only selected boards, targeting interoperability with legacy applications, will offer this.



## Constructors

```
class hashlib.sha256 ([data ])
    Create an SHA256 hasher object and optionally feed data into it.

class hashlib.sha1 ([data ])
    Create an SHA1 hasher object and optionally feed data into it.

class hashlib.md5 ([data ])
    Create an MD5 hasher object and optionally feed data into it.
```

## Methods

```
hash.update (data)
    Feed more binary data into hash.

hash.digest ()
    Return hash for all data passed through hash, as a bytes object. After this method is called, more data cannot be
    fed into the hash any longer.

hash.hexdigest ()
    This method is NOT implemented. Use binascii.hexlify(hash.digest()) to achieve a similar effect.
```

## io – input/output streams

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [io](#).*

This module contains additional types of `stream` (file-like) objects and helper functions.

## Conceptual hierarchy

---

### Difference to CPython

Conceptual hierarchy of stream base classes is simplified in MicroPython, as described in this section.

---

(Abstract) base stream classes, which serve as a foundation for behavior of all the concrete classes, adhere to few dichotomies (pair-wise classifications) in CPython. In MicroPython, they are somewhat simplified and made implicit to achieve higher efficiencies and save resources.

An important dichotomy in CPython is unbuffered vs buffered streams. In MicroPython, all streams are currently unbuffered. This is because all modern OSes, and even many RTOSes and filesystem drivers already perform buffering on their side. Adding another layer of buffering is counter-productive (an issue known as “bufferbloat”) and takes precious memory. Note that there still cases where buffering may be useful, so we may introduce optional buffering support at a later time.

But in CPython, another important dichotomy is tied with “bufferedness” - it’s whether a stream may incur short read/writes or not. A short read is when a user asks e.g. 10 bytes from a stream, but gets less, similarly for writes. In CPython, unbuffered streams are automatically short operation susceptible, while buffered are guarantee against them. The no short read/writes is an important trait, as it allows to develop more concise and efficient programs - something which is highly desirable for MicroPython. So, while MicroPython doesn’t support buffered streams, it still provides for no-short-operations streams. Whether there will be short operations or not depends on each particular class’ needs,

but developers are strongly advised to favor no-short-operations behavior for the reasons stated above. For example, MicroPython sockets are guaranteed to avoid short read/writes. Actually, at this time, there is no example of a short-operations stream class in the core, and one would be a port-specific class, where such a need is governed by hardware peculiarities.

The no-short-operations behavior gets tricky in case of non-blocking streams, blocking vs non-blocking behavior being another CPython dichotomy, fully supported by MicroPython. Non-blocking streams never wait for data either to arrive or be written - they read/write whatever possible, or signal lack of data (or ability to write data). Clearly, this conflicts with “no-short-operations” policy, and indeed, a case of non-blocking buffered (and this no-short-ops) streams is convoluted in CPython - in some places, such combination is prohibited, in some it’s undefined or just not documented, in some cases it raises verbose exceptions. The matter is much simpler in MicroPython: non-blocking stream are important for efficient asynchronous operations, so this property prevails on the “no-short-ops” one. So, while blocking streams will avoid short reads/writes whenever possible (the only case to get a short read is if end of file is reached, or in case of error (but errors don’t return short data, but raise exceptions)), non-blocking streams may produce short data to avoid blocking the operation.

The final dichotomy is binary vs text streams. MicroPython of course supports these, but while in CPython text streams are inherently buffered, they aren’t in MicroPython. (Indeed, that’s one of the cases for which we may introduce buffering support.)

Note that for efficiency, MicroPython doesn’t provide abstract base classes corresponding to the hierarchy above, and it’s not possible to implement, or subclass, a stream class in pure Python.

## Functions

`io.open(name, mode='r', **kwargs)`

Open a file. Builtin `open()` function is aliased to this function. All ports (which provide access to file system) are required to support `mode` parameter, but support for other arguments vary by port.

## Classes

`class io.FileIO(...)`

This is type of a file open in binary mode, e.g. using `open(name, "rb")`. You should not instantiate this class directly.

`class io.TextIOWrapper(...)`

This is type of a file open in text mode, e.g. using `open(name, "rt")`. You should not instantiate this class directly.

`class io.StringIO([string])`

`class io.BytesIO([string])`

In-memory file-like objects for input/output. `StringIO` is used for text-mode I/O (similar to a normal file opened with “t” modifier). `BytesIO` is used for binary-mode I/O (similar to a normal file opened with “b” modifier). Initial contents of file-like objects can be specified with `string` parameter (should be normal string for `StringIO` or bytes object for `BytesIO`). All the usual file methods like `read()`, `write()`, `seek()`, `flush()`, `close()` are available on these objects, and additionally, a following method:

`getvalue()`

Get the current contents of the underlying buffer which holds data.

## json – JSON encoding and decoding

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [json](#).*

This modules allows to convert between Python objects and the JSON data format.

### Functions

`json.dump(obj, stream)`

Serialise `obj` to a JSON string, writing it to the given `stream`.

`json.dumps(obj)`

Return `obj` represented as a JSON string.

`json.load(stream)`

Parse the given `stream`, interpreting it as a JSON string and deserialising the data to a Python object. The resulting object is returned.

Parsing continues until end-of-file is encountered. A `ValueError` is raised if the data in `stream` is not correctly formed.

`json.loads(str)`

Parse the JSON `str` and return an object. Raises `ValueError` if the string is not correctly formed.

## re – simple regular expressions

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [re](#).*

This module implements regular expression operations. Regular expression syntax supported is a subset of CPython `re` module (and actually is a subset of POSIX extended regular expressions).

Supported operators are:

'.' Match any character.

'[...]' Match set of characters. Individual characters and ranges are supported, including negated sets (e.g. `[^a-c]`).

'^'

'\$'

'?'

'\*'

'+'

'??'

'\*?'

'+?'

'|'

'(...)' Grouping. Each group is capturing (a substring it captures can be accessed with `match.group()` method).

**NOT SUPPORTED:** Counted repetitions (`{m,n}`), more advanced assertions (`\b`, `\B`), named groups (`(?P<name>...)`), non-capturing groups (`(?:...)`), etc.

## Functions

`re.compile(regex_str[, flags])`

Compile regular expression, return *regex* object.

`re.match(regex_str, string)`

Compile *regex\_str* and match against *string*. Match always happens from starting position in a string.

`re.search(regex_str, string)`

Compile *regex\_str* and search it in a *string*. Unlike *match*, this will search string for first position which matches *regex* (which still may be 0 if *regex* is anchored).

`re.DEBUG`

Flag value, display debug information about compiled expression.

## Regex objects

Compiled regular expression. Instances of this class are created using *re.compile()*.

`regex.match(string)`

`regex.search(string)`

Similar to the module-level functions *match()* and *search()*. Using methods is (much) more efficient if the same *regex* is applied to multiple strings.

`regex.split(string, max_split=-1)`

Split a *string* using *regex*. If *max\_split* is given, it specifies maximum number of splits to perform. Returns list of strings (there may be up to *max\_split+1* elements if it's specified).

## Match objects

Match objects as returned by *match()* and *search()* methods.

`match.group([index])`

Return matching (sub)string. *index* is 0 for entire match, 1 and above for each capturing group. Only numeric groups are supported.

## sys – system specific functions

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [sys](#).*

## Functions

`sys.exit (retval=0)`

Terminate current program with a given exit code. Underlyingly, this function raise as `SystemExit` exception. If an argument is given, its value given as an argument to `SystemExit`.

`sys.print_exception (exc, file=sys.stdout)`

Print exception with a traceback to a file-like object *file* (or `sys.stdout` by default).

---

### Difference to CPython

This is simplified version of a function which appears in the `traceback` module in CPython. Unlike `traceback.print_exception()`, this function takes just exception value instead of exception type, exception value, and traceback object; *file* argument should be positional; further arguments are not supported.

---

## Constants

`sys.argv`

A mutable list of arguments the current program was started with.

`sys.byteorder`

The byte order of the system ("little" or "big").

`sys.implementation`

Object with information about the current Python implementation. For CircuitPython, it has following attributes:

- *name* - string "circuitpython"
- *version* - tuple (major, minor, micro), e.g. (1, 7, 0)

This object is the recommended way to distinguish CircuitPython from other Python implementations (note that it still may not exist in the very minimal ports).

---

### Difference to CPython

CPython mandates more attributes for this object, but the actual useful bare minimum is implemented in CircuitPython.

---

`sys.maxsize`

Maximum value which a native integer type can hold on the current platform, or maximum value representable by CircuitPython integer type, if it's smaller than platform max value (that is the case for CircuitPython ports without long int support).

This attribute is useful for detecting "bitness" of a platform (32-bit vs 64-bit, etc.). It's recommended to not compare this attribute to some value directly, but instead count number of bits in it:

```
bits = 0
v = sys.maxsize
while v:
    bits += 1
    v >>= 1
if bits > 32:
    # 64-bit (or more) platform
    ...
else:
```

(continues on next page)

(continued from previous page)

```
# 32-bit (or less) platform
# Note that on 32-bit platform, value of bits may be less than 32
# (e.g. 31) due to peculiarities described above, so use "> 16",
# "> 32", "> 64" style of comparisons.
```

**sys.modules**

Dictionary of loaded modules. On some ports, it may not include builtin modules.

**sys.path**

A mutable list of directories to search for imported modules.

**sys.platform**

The platform that CircuitPython is running on. For OS/RTOS ports, this is usually an identifier of the OS, e.g. "linux". For baremetal ports it is an identifier of the chip on a board, e.g. "MicroChip SAMD51". It thus can be used to distinguish one board from another. If you need to check whether your program runs on CircuitPython (vs other Python implementation), use *sys.implementation* instead.

**sys.stderr**

Standard error stream.

**sys.stdin**

Standard input stream.

**sys.stdout**

Standard output stream.

**sys.version**

Python language version that this implementation conforms to, as a string.

**sys.version\_info**

Python language version that this implementation conforms to, as a tuple of ints.

**uctypes – access binary data in a structured way**

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

This module implements “foreign data interface” for MicroPython. The idea behind it is similar to CPython’s *ctypes* modules, but the actual API is different, streamlined and optimized for small size. The basic idea of the module is to define data structure layout with about the same power as the C language allows, and then access it using familiar dot-syntax to reference sub-fields.

**See also:**

**Module *struct*** Standard Python way to access binary data structures (doesn’t scale well to large and complex structures).

## Defining structure layout

Structure layout is defined by a “descriptor” - a Python dictionary which encodes field names as keys and other properties required to access them as associated values. Currently, ctypes requires explicit specification of offsets for each field. Offset are given in bytes from a structure start.

Following are encoding examples for various field types:

- Scalar types:

```
"field_name": offset | ctypes.UINT32
```

i.e. value is scalar type identifier ORed with field offset (in bytes) from the start of the structure.

- Recursive structures:

```
"sub": (offset, {
    "b0": 0 | ctypes.UINT8,
    "b1": 1 | ctypes.UINT8,
})
```

i.e. value is a 2-tuple, first element of which is offset, and second is a structure descriptor dictionary (note: offsets in recursive descriptors are relative to the structure it defines).

- Arrays of primitive types:

```
"arr": (offset | ctypes.ARRAY, size | ctypes.UINT8),
```

i.e. value is a 2-tuple, first element of which is ARRAY flag ORed with offset, and second is scalar element type ORed number of elements in array.

- Arrays of aggregate types:

```
"arr2": (offset | ctypes.ARRAY, size, {"b": 0 | ctypes.UINT8}),
```

i.e. value is a 3-tuple, first element of which is ARRAY flag ORed with offset, second is a number of elements in array, and third is descriptor of element type.

- Pointer to a primitive type:

```
"ptr": (offset | ctypes.PTR, ctypes.UINT8),
```

i.e. value is a 2-tuple, first element of which is PTR flag ORed with offset, and second is scalar element type.

- Pointer to an aggregate type:

```
"ptr2": (offset | ctypes.PTR, {"b": 0 | ctypes.UINT8}),
```

i.e. value is a 2-tuple, first element of which is PTR flag ORed with offset, second is descriptor of type pointed to.

- Bitfields:

```
"bitf0": offset | ctypes.BFUINT16 | lsb << ctypes.BF_POS | bitsize << ctypes.BF_LEN,
```

i.e. value is type of scalar value containing given bitfield (typenamees are similar to scalar types, but prefixes with “BF”), ORed with offset for scalar value containing the bitfield, and further ORed with values for bit offset and bit length of the bitfield within scalar value, shifted by BF\_POS and BF\_LEN positions, respectively. Bitfield

position is counted from the least significant bit, and is the number of right-most bit of a field (in other words, it's a number of bits a scalar needs to be shifted right to extract the bitfield).

In the example above, first a `UINT16` value will be extracted at offset 0 (this detail may be important when accessing hardware registers, where particular access size and alignment are required), and then bitfield whose rightmost bit is *lsbit* bit of this `UINT16`, and length is *bitsize* bits, will be extracted. For example, if *lsbit* is 0 and *bitsize* is 8, then effectively it will access least-significant byte of `UINT16`.

Note that bitfield operations are independent of target byte endianness, in particular, example above will access least-significant byte of `UINT16` in both little- and big-endian structures. But it depends on the least significant bit being numbered 0. Some targets may use different numbering in their native ABI, but `uctypes` always uses the normalized numbering described above.

## Module contents

**class** `uctypes.struct(addr, descriptor, layout_type=NATIVE)`

Instantiate a “foreign data structure” object based on structure address in memory, descriptor (encoded as a dictionary), and layout type (see below).

`uctypes.LITTLE_ENDIAN`

Layout type for a little-endian packed structure. (Packed means that every field occupies exactly as many bytes as defined in the descriptor, i.e. the alignment is 1).

`uctypes.BIG_ENDIAN`

Layout type for a big-endian packed structure.

`uctypes.NATIVE`

Layout type for a native structure - with data endianness and alignment conforming to the ABI of the system on which MicroPython runs.

`uctypes.sizeof(struct)`

Return size of data structure in bytes. Argument can be either structure class or specific instantiated structure object (or its aggregate field).

`uctypes.addressof(obj)`

Return address of an object. Argument should be bytes, bytearray or other object supporting buffer protocol (and address of this buffer is what actually returned).

`uctypes.bytes_at(addr, size)`

Capture memory at the given address and size as bytes object. As bytes object is immutable, memory is actually duplicated and copied into bytes object, so if memory contents change later, created object retains original value.

`uctypes bytearray_at(addr, size)`

Capture memory at the given address and size as bytearray object. Unlike `bytes_at()` function above, memory is captured by reference, so it can be both written too, and you will access current value at the given memory address.



## Structure descriptors and instantiating structure objects

Given a structure descriptor dictionary and its layout type, you can instantiate a specific structure instance at a given memory address using `uctypes.struct()` constructor. Memory address usually comes from following sources:

- Predefined address, when accessing hardware registers on a baremetal system. Lookup these addresses in datasheet for a particular MCU/SoC.
- As a return value from a call to some FFI (Foreign Function Interface) function.
- From `uctypes.addressof()`, when you want to pass arguments to an FFI function, or alternatively, to access some data for I/O (for example, data read from a file or network socket).

## Structure objects

Structure objects allow accessing individual fields using standard dot notation: `my_struct.substruct1.field1`. If a field is of scalar type, getting it will produce a primitive value (Python integer or float) corresponding to the value contained in a field. A scalar field can also be assigned to.

If a field is an array, its individual elements can be accessed with the standard subscript operator `[]` - both read and assigned to.

If a field is a pointer, it can be dereferenced using `[0]` syntax (corresponding to `C *` operator, though `[0]` works in C too). Subscripting a pointer with other integer values but 0 are supported too, with the same semantics as in C.

Summing up, accessing structure fields generally follows C syntax, except for pointer dereference, when you need to use `[0]` operator instead of `*`.

## Limitations

Accessing non-scalar fields leads to allocation of intermediate objects to represent them. This means that special care should be taken to layout a structure which needs to be accessed when memory allocation is disabled (e.g. from an interrupt). The recommendations are:

- Avoid nested structures. For example, instead of `mcu_registers.peripheral_a.register1`, define separate layout descriptors for each peripheral, to be accessed as `peripheral_a.register1`.
- Avoid other non-scalar data, like array. For example, instead of `peripheral_a.register[0]` use `peripheral_a.register0`.

Note that these recommendations will lead to decreased readability and conciseness of layouts, so they should be used only if the need to access structure fields without allocation is anticipated (it's even possible to define 2 parallel layouts - one for normal usage, and a restricted one to use when memory allocation is prohibited).

## uselect – wait for events on a set of streams

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [select](#).*

This module provides functions to efficiently wait for events on multiple `stream` objects (select streams which are ready for operations).

## Functions

`uselect.poll()`

Create an instance of the `Poll` class.

`uselect.select(rlist, wlist, xlist[, timeout])`

Wait for activity on a set of objects.

This function is provided by some MicroPython ports for compatibility and is not efficient. Usage of `Poll` is recommended instead.

## class `Poll`

### Methods

`poll.register(obj[, eventmask])`

Register stream *obj* for polling. *eventmask* is logical OR of:

- `uselect.POLLIN` - data available for reading
- `uselect.POLLOUT` - more data can be written

Note that flags like `uselect.POLLHUP` and `uselect.POLLERR` are *not* valid as input eventmask (these are unsolicited events which will be returned from `poll()` regardless of whether they are asked for). This semantics is per POSIX.

*eventmask* defaults to `uselect.POLLIN | uselect.POLLOUT`.

`poll.unregister(obj)`

Unregister *obj* from polling.

`poll.modify(obj, eventmask)`

Modify the *eventmask* for *obj*.

`poll.poll(timeout=-1)`

Wait for at least one of the registered objects to become ready or have an exceptional condition, with optional timeout in milliseconds (if *timeout* arg is not specified or -1, there is no timeout).

Returns list of (*obj*, *event*, ...) tuples. There may be other elements in tuple, depending on a platform and version, so don't assume that its size is 2. The *event* element specifies which events happened with a stream and is a combination of `uselect.POLL*` constants described above. Note that flags `uselect.POLLHUP` and `uselect.POLLERR` can be returned at any time (even if were not asked for), and must be acted on accordingly (the corresponding stream unregistered from `poll` and likely closed), because otherwise all further invocations of `poll()` may return immediately with these flags set for this stream again.

In case of timeout, an empty list is returned.

---

### Difference to CPython

Tuples returned may contain more than 2 elements as described above.

---

`poll.ipoll(timeout=-1, flags=0)`

Like `poll.poll()`, but instead returns an iterator which yields a callee-owned tuples. This function provides efficient, allocation-free way to poll on streams.

If `flags` is 1, one-shot behavior for events is employed: streams for which events happened will have their event masks automatically reset (equivalent to `poll.modify(obj, 0)`), so new events for such a stream won't be processed until new mask is set with `poll.modify()`. This behavior is useful for asynchronous I/O schedulers.

---

### Difference to CPython

This function is a MicroPython extension.

---

## usocket – socket module

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [socket](#).*

This module provides access to the BSD socket interface.

---

### Difference to CPython

For efficiency and consistency, socket objects in MicroPython implement a `stream` (file-like) interface directly. In CPython, you need to convert a socket to a file-like object using `makefile()` method. This method is still supported by MicroPython (but is a no-op), so where compatibility with CPython matters, be sure to use it.

---

## Socket address format(s)

The native socket address format of the `usocket` module is an opaque data type returned by `getaddrinfo` function, which must be used to resolve textual address (including numeric addresses):

```
sockaddr = usocket.getaddrinfo('www.micropython.org', 80)[0][-1]
# You must use getaddrinfo() even for numeric addresses
sockaddr = usocket.getaddrinfo('127.0.0.1', 80)[0][-1]
# Now you can use that address
sock.connect(addr)
```

Using `getaddrinfo` is the most efficient (both in terms of memory and processing power) and portable way to work with addresses.

However, `socket` module (note the difference with native MicroPython `usocket` module described here) provides CPython-compatible way to specify addresses using tuples, as described below.

Summing up:

- Always use `getaddrinfo` when writing portable applications.

- Tuple addresses described below can be used as a shortcut for quick hacks and interactive use, if your port supports them.

Tuple address format for `socket` module:

- IPv4: (*ipv4\_address*, *port*), where *ipv4\_address* is a string with dot-notation numeric IPv4 address, e.g. "8.8.8.8", and *port* is an integer port number in the range 1-65535. Note the domain names are not accepted as *ipv4\_address*, they should be resolved first using `usocket.getaddrinfo()`.
- IPv6: (*ipv6\_address*, *port*, *flowinfo*, *scopeid*), where *ipv6\_address* is a string with colon-notation numeric IPv6 address, e.g. "2001:db8::1", and *port* is an integer port number in the range 1-65535. *flowinfo* must be 0. *scopeid* is the interface scope identifier for link-local addresses. Note the domain names are not accepted as *ipv6\_address*, they should be resolved first using `usocket.getaddrinfo()`.

## Functions

`usocket.socket` (*af*=`AF_INET`, *type*=`SOCK_STREAM`, *proto*=`IPPROTO_TCP`)

Create a new socket using the given address family, socket type and protocol number. Note that specifying *proto* in most cases is not required (and not recommended, as some MicroPython ports may omit `IPPROTO_*` constants). Instead, *type* argument will select needed protocol automatically:

```
# Create STREAM TCP socket
socket(AF_INET, SOCK_STREAM)
# Create DGRAM UDP socket
socket(AF_INET, SOCK_DGRAM)
```

`usocket.getaddrinfo` (*host*, *port*)

Translate the host/port argument into a sequence of 5-tuples that contain all the necessary arguments for creating a socket connected to that service. The list of 5-tuples has following structure:

```
(family, type, proto, canonname, sockaddr)
```

The following example shows how to connect to a given url:

```
s = usocket.socket()
s.connect(usocket.getaddrinfo('www.micropython.org', 80)[0][-1])
```

---

### Difference to CPython

CPython raises a `socket.gaierror` exception (`OSError` subclass) in case of error in this function. MicroPython doesn't have `socket.gaierror` and raises `OSError` directly. Note that error numbers of `getaddrinfo()` form a separate namespace and may not match error numbers from `uerrno` module. To distinguish `getaddrinfo()` errors, they are represented by negative numbers, whereas standard system errors are positive numbers (error numbers are accessible using `e.args[0]` property from an exception object). The use of negative values is a provisional detail which may change in the future.

---

`usocket.inet_ntop` (*af*, *bin\_addr*)

Convert a binary network address *bin\_addr* of the given address family *af* to a textual representation:

```
>>> usocket.inet_ntop(usocket.AF_INET, b"\x7f\x0\x0\x1")
'127.0.0.1'
```

`usocket.inet_pton` (*af*, *txt\_addr*)

Convert a textual network address *txt\_addr* of the given address family *af* to a binary representation:

```
>>> usocket.inet_pton(usocket.AF_INET, "1.2.3.4")
b'\x01\x02\x03\x04'
```

## Constants

`usocket.AF_INET`

`usocket.AF_INET6`

Address family types. Availability depends on a particular MicroPython port.

`usocket.SOCK_STREAM`

`usocket.SOCK_DGRAM`

Socket types.

`usocket.IPPROTO_UDP`

`usocket.IPPROTO_TCP`

IP protocol numbers. Availability depends on a particular MicroPython port. Note that you don't need to specify these in a call to `usocket.socket()`, because `SOCK_STREAM` socket type automatically selects `IPPROTO_TCP`, and `SOCK_DGRAM` - `IPPROTO_UDP`. Thus, the only real use of these constants is as an argument to `usocket.socket.setsockopt()`.

`usocket.SOL_*`

Socket option levels (an argument to `usocket.socket.setsockopt()`). The exact inventory depends on a MicroPython port.

`usocket.SO_*`

Socket options (an argument to `usocket.socket.setsockopt()`). The exact inventory depends on a MicroPython port.

Constants specific to WiPy:

`usocket.IPPROTO_SEC`

Special protocol value to create SSL-compatible socket.

## class socket

### Methods

`socket.close()`

Mark the socket closed and release all resources. Once that happens, all future operations on the socket object will fail. The remote end will receive EOF indication if supported by protocol.

Sockets are automatically closed when they are garbage-collected, but it is recommended to `close()` them explicitly as soon you finished working with them.

`socket.bind(address)`

Bind the socket to *address*. The socket must not already be bound.

`socket.listen([backlog])`

Enable a server to accept connections. If *backlog* is specified, it must be at least 0 (if it's lower, it will be set to 0); and specifies the number of unaccepted connections that the system will allow before refusing new connections. If not specified, a default reasonable value is chosen.

`socket.accept()`

Accept a connection. The socket must be bound to an address and listening for connections. The return value is a pair (conn, address) where conn is a new socket object usable to send and receive data on the connection, and address is the address bound to the socket on the other end of the connection.

`socket.connect(address)`

Connect to a remote socket at *address*.

`socket.send(bytes)`

Send data to the socket. The socket must be connected to a remote socket. Returns number of bytes sent, which may be smaller than the length of data (“short write”).

`socket.sendall(bytes)`

Send all data to the socket. The socket must be connected to a remote socket. Unlike `send()`, this method will try to send all of data, by sending data chunk by chunk consecutively.

The behavior of this method on non-blocking sockets is undefined. Due to this, on MicroPython, it’s recommended to use `write()` method instead, which has the same “no short writes” policy for blocking sockets, and will return number of bytes sent on non-blocking sockets.

`socket.recv(bufsize)`

Receive data from the socket. The return value is a bytes object representing the data received. The maximum amount of data to be received at once is specified by *bufsize*.

`socket.sendto(bytes, address)`

Send data to the socket. The socket should not be connected to a remote socket, since the destination socket is specified by *address*.

`socket.recvfrom(bufsize)`

Receive data from the socket. The return value is a pair (*bytes*, *address*) where *bytes* is a bytes object representing the data received and *address* is the address of the socket sending the data.

`socket.setsockopt(level, optname, value)`

Set the value of the given socket option. The needed symbolic constants are defined in the socket module (SO\_\* etc.). The *value* can be an integer or a bytes-like object representing a buffer.

`socket.settimeout(value)`

**Note:** Not every port supports this method, see below.

Set a timeout on blocking socket operations. The value argument can be a nonnegative floating point number expressing seconds, or None. If a non-zero value is given, subsequent socket operations will raise an `OSError` exception if the timeout period value has elapsed before the operation has completed. If zero is given, the socket is put in non-blocking mode. If None is given, the socket is put in blocking mode.

Not every MicroPython port supports this method. A more portable and generic solution is to use `uselect.poll` object. This allows to wait on multiple objects at the same time (and not just on sockets, but on generic stream objects which support polling). Example:

```
# Instead of:
s.settimeout(1.0) # time in seconds
s.read(10) # may timeout

# Use:
poller = uselect.poll()
poller.register(s, uselect.POLLIN)
res = poller.poll(1000) # time in milliseconds
if not res:
    # s is still not ready for input, i.e. operation timed out
```

---

### Difference to CPython

CPython raises a `socket.timeout` exception in case of timeout, which is an `OSError` subclass. MicroPython raises an `OSError` directly instead. If you use `except OSError:` to catch the exception, your code will

work both in MicroPython and CPython.

---

`socket.setblocking(flag)`

Set blocking or non-blocking mode of the socket: if *flag* is false, the socket is set to non-blocking, else to blocking mode.

This method is a shorthand for certain `settimeout()` calls:

- `sock.setblocking(True)` is equivalent to `sock.settimeout(None)`
- `sock.setblocking(False)` is equivalent to `sock.settimeout(0)`

`socket.makefile(mode='rb', buffering=0)`

Return a file object associated with the socket. The exact returned type depends on the arguments given to `makefile()`. The support is limited to binary modes only ('rb', 'wb', and 'rwb'). CPython's arguments: *encoding*, *errors* and *newline* are not supported.

---

### Difference to CPython

As MicroPython doesn't support buffered streams, values of *buffering* parameter is ignored and treated as if it was 0 (unbuffered).

---

---

### Difference to CPython

Closing the file object returned by `makefile()` WILL close the original socket as well.

---

`socket.read([size])`

Read up to *size* bytes from the socket. Return a bytes object. If *size* is not given, it reads all data available from the socket until EOF; as such the method will not return until the socket is closed. This function tries to read as much data as requested (no "short reads"). This may be not possible with non-blocking socket though, and then less data will be returned.

`socket.readinto(buf[, nbytes])`

Read bytes into the *buf*. If *nbytes* is specified then read at most that many bytes. Otherwise, read at most `len(buf)` bytes. Just as `read()`, this method follows "no short reads" policy.

Return value: number of bytes read and stored into *buf*.

`socket.readline()`

Read a line, ending in a newline character.

Return value: the line read.

`socket.write(buf)`

Write the buffer of bytes to the socket. This function will try to write all data to a socket (no "short writes"). This may be not possible with a non-blocking socket though, and returned value will be less than the length of *buf*.

Return value: number of bytes written.

**exception** `usocket.error`

MicroPython does NOT have this exception.

---

### Difference to CPython

CPython used to have a `socket.error` exception which is now deprecated, and is an alias of `OSError`. In MicroPython, use `OSError` directly.

---

## ussl – SSL/TLS module

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [ssl](#).*

This module provides access to Transport Layer Security (previously and widely known as “Secure Sockets Layer”) encryption and peer authentication facilities for network sockets, both client-side and server-side.

### Functions

`ussl.wrap_socket(sock, server_side=False, keyfile=None, certfile=None, cert_reqs=CERT_NONE, ca_certs=None)`

Takes a stream `sock` (usually `usocket.socket` instance of `SOCK_STREAM` type), and returns an instance of `ssl.SSLSocket`, which wraps the underlying stream in an SSL context. Returned object has the usual `stream` interface methods like `read()`, `write()`, etc. In MicroPython, the returned object does not expose `socket` interface and methods like `recv()`, `send()`. In particular, a server-side SSL socket should be created from a normal socket returned from `accept()` on a non-SSL listening server socket.

Depending on the underlying module implementation in a particular MicroPython port, some or all keyword arguments above may be not supported.

**Warning:** Some implementations of `ussl` module do NOT validate server certificates, which makes an SSL connection established prone to man-in-the-middle attacks.

### Exceptions

`ssl.SSLError`

This exception does NOT exist. Instead its base class, `OSError`, is used.

### Constants

`ussl.CERT_NONE`

`ussl.CERT_OPTIONAL`

`ussl.CERT_REQUIRED`

Supported values for `cert_reqs` parameter.



## uzlib – zlib decompression

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

*This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: [zlib](#).*

This module allows to decompress binary data compressed with **DEFLATE algorithm** (commonly used in zlib library and gzip archiver). Compression is not yet implemented.

### Functions

`uzlib.decompress (data, wbits=0, bufsize=0)`

Return decompressed *data* as bytes. *wbits* is DEFLATE dictionary window size used during compression (8-15, the dictionary size is power of 2 of that value). Additionally, if value is positive, *data* is assumed to be zlib stream (with zlib header). Otherwise, if it's negative, it's assumed to be raw DEFLATE stream. *bufsize* parameter is for compatibility with CPython and is ignored.

**class** `uzlib.DecomIO (stream, wbits=0)`

Create a `stream` wrapper which allows transparent decompression of compressed data in another *stream*. This allows to process compressed streams with data larger than available heap size. In addition to values described in `decompress()`, *wbits* may take values 24..31 (16 + 8..15), meaning that input stream has gzip header.

---

### Difference to CPython

This class is MicroPython extension. It's included on provisional basis and may be changed considerably or removed in later versions.

---

### Omitted functions in the `string` library

A few string operations are not enabled on small builds (usually non-Express), due to limited flash memory: `string.center()`, `string.partition()`, `string.splitlines()`, `string.reversed()`.

### CircuitPython/MicroPython-specific libraries

Functionality specific to the CircuitPython/MicroPython implementation is available in the following libraries. These libraries may change significantly or be removed in future versions of CircuitPython.

## btree – simple BTree database

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

The `btree` module implements a simple key-value database using external storage (disk files, or in general case, a random-access stream). Keys are stored sorted in the database, and besides efficient retrieval by a key value, a database also supports efficient ordered range scans (retrieval of values with the keys in a given range). On the application interface side, BTree database work as close a possible to a way standard `dict` type works, one notable difference is that both keys and values must be `bytes` objects (so, if you want to store objects of other types, you need to serialize them to `bytes` first).

The module is based on the well-known BerkeleyDB library, version 1.xx.

Example:

```
import btree

# First, we need to open a stream which holds a database
# This is usually a file, but can be in-memory database
# using uio.BytesIO, a raw flash partition, etc.
# Oftentimes, you want to create a database file if it doesn't
# exist and open if it exists. Idiom below takes care of this.
# DO NOT open database with "a+b" access mode.
try:
    f = open("mydb", "r+b")
except OSError:
    f = open("mydb", "w+b")

# Now open a database itself
db = btree.open(f)

# The keys you add will be sorted internally in the database
db[b"3"] = b"three"
db[b"1"] = b"one"
db[b"2"] = b"two"

# Assume that any changes are cached in memory unless
# explicitly flushed (or database closed). Flush database
# at the end of each "transaction".
db.flush()

# Prints b'two'
print(db[b"2"])

# Iterate over sorted keys in the database, starting from b"2"
# until the end of the database, returning only values.
# Mind that arguments passed to values() method are *key* values.
# Prints:
#   b'two'
#   b'three'
for word in db.values(b"2"):
    print(word)
```

(continues on next page)

(continued from previous page)

```

del db[b"2"]

# No longer true, prints False
print(b"2" in db)

# Prints:
# b"1"
# b"3"
for key in db:
    print(key)

db.close()

# Don't forget to close the underlying stream!
f.close()

```

## Functions

`btree.open(stream, \*, flags=0, pagesize=0, cachesize=0, minkeypage=0)`

Open a database from a random-access `stream` (like an open file). All other parameters are optional and keyword-only, and allow to tweak advanced parameters of the database operation (most users will not need them):

- *flags* - Currently unused.
- *pagesize* - Page size used for the nodes in BTree. Acceptable range is 512-65536. If 0, a port-specific default will be used, optimized for port's memory usage and/or performance.
- *cachesize* - Suggested memory cache size in bytes. For a board with enough memory using larger values may improve performance. Cache policy is as follows: entire cache is not allocated at once; instead, accessing a new page in database will allocate a memory buffer for it, until value specified by *cachesize* is reached. Then, these buffers will be managed using LRU (least recently used) policy. More buffers may still be allocated if needed (e.g., if a database contains big keys and/or values). Allocated cache buffers aren't reclaimed.
- *minkeypage* - Minimum number of keys to store per page. Default value of 0 equivalent to 2.

Returns a BTree object, which implements a dictionary protocol (set of methods), and some additional methods described below.

## Methods

`btree.close()`

Close the database. It's mandatory to close the database at the end of processing, as some unwritten data may be still in the cache. Note that this does not close underlying stream with which the database was opened, it should be closed separately (which is also mandatory to make sure that data flushed from buffer to the underlying storage).

`btree.flush()`

Flush any data in cache to the underlying stream.

`btree.__getitem__(key)`

`btree.get(key, default=None)`

`btree.__setitem__(key, val)`

```
btree.__getitem__(key)
btree.__contains__(key)
```

Standard dictionary methods.

```
btree.__iter__()
```

A BTree object can be iterated over directly (similar to a dictionary) to get access to all keys in order.

```
btree.keys([start_key[, end_key[, flags]]])
btree.values([start_key[, end_key[, flags]]])
btree.items([start_key[, end_key[, flags]]])
```

These methods are similar to standard dictionary methods, but also can take optional parameters to iterate over a key sub-range, instead of the entire database. Note that for all 3 methods, *start\_key* and *end\_key* arguments represent key values. For example, *values()* method will iterate over values corresponding to they key range given. None values for *start\_key* means “from the first key”, no *end\_key* or its value of None means “until the end of database”. By default, range is inclusive of *start\_key* and exclusive of *end\_key*, you can include *end\_key* in iteration by passing *flags* of *btree.INCL*. You can iterate in descending key direction by passing *flags* of *btree.DESC*. The flags values can be Ored together.

## Constants

**btree.INCL**

A flag for *keys()*, *values()*, *items()* methods to specify that scanning should be inclusive of the end key.

**btree.DESC**

A flag for *keys()*, *values()*, *items()* methods to specify that scanning should be in descending direction of keys.

## framebuf — Frame buffer manipulation

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

This module provides a general frame buffer which can be used to create bitmap images, which can then be sent to a display.

### class FrameBuffer

The FrameBuffer class provides a pixel buffer which can be drawn upon with pixels, lines, rectangles, text and even other FrameBuffer’s. It is useful when generating output for displays.

For example:

```
import framebuf

# FrameBuffer needs 2 bytes for every RGB565 pixel
fbuf = FrameBuffer(bytearray(10 * 100 * 2), 10, 100, framebuf.RGB565)

fbuf.fill(0)
```

(continues on next page)

(continued from previous page)

```
fbuf.text('MicroPython!', 0, 0, 0xffff)
fbuf.hline(0, 10, 96, 0xffff)
```

## Constructors

**class** `framebuf.FrameBuffer` (*buffer, width, height, format, stride=width*)

Construct a FrameBuffer object. The parameters are:

- *buffer* is an object with a buffer protocol which must be large enough to contain every pixel defined by the width, height and format of the FrameBuffer.
- *width* is the width of the FrameBuffer in pixels
- *height* is the height of the FrameBuffer in pixels
- *format* specifies the type of pixel used in the FrameBuffer; permissible values are listed under Constants below. These set the number of bits used to encode a color value and the layout of these bits in *buffer*. Where a color value *c* is passed to a method, *c* is a small integer with an encoding that is dependent on the format of the FrameBuffer.
- *stride* is the number of pixels between each horizontal line of pixels in the FrameBuffer. This defaults to *width* but may need adjustments when implementing a FrameBuffer within another larger FrameBuffer or screen. The *buffer* size must accommodate an increased step size.

One must specify valid *buffer, width, height, format* and optionally *stride*. Invalid *buffer* size or dimensions may lead to unexpected errors.

## Drawing primitive shapes

The following methods draw shapes onto the FrameBuffer.

`FrameBuffer.fill` (*c*)

Fill the entire FrameBuffer with the specified color.

`FrameBuffer.pixel` (*x, y*[, *c* ])

If *c* is not given, get the color value of the specified pixel. If *c* is given, set the specified pixel to the given color.

`FrameBuffer.hline` (*x, y, w, c*)

`FrameBuffer.vline` (*x, y, h, c*)

`FrameBuffer.line` (*x1, y1, x2, y2, c*)

Draw a line from a set of coordinates using the given color and a thickness of 1 pixel. The *line* method draws the line up to a second set of coordinates whereas the *hline* and *vline* methods draw horizontal and vertical lines respectively up to a given length.

`FrameBuffer.rect` (*x, y, w, h, c*)

`FrameBuffer.fill_rect` (*x, y, w, h, c*)

Draw a rectangle at the given location, size and color. The *rect* method draws only a 1 pixel outline whereas the *fill\_rect* method draws both the outline and interior.

## Drawing text

`Framebuffer.text(s, x, y[, c])`

Write text to the `Framebuffer` using the the coordinates as the upper-left corner of the text. The color of the text can be defined by the optional argument but is otherwise a default value of 1. All characters have dimensions of 8x8 pixels and there is currently no way to change the font.

## Other methods

`Framebuffer.scroll(xstep, ystep)`

Shift the contents of the `Framebuffer` by the given vector. This may leave a footprint of the previous colors in the `Framebuffer`.

`Framebuffer.blit(fbuf, x, y[, key])`

Draw another `Framebuffer` on top of the current one at the given coordinates. If *key* is specified then it should be a color integer and the corresponding color will be considered transparent: all pixels with that color value will not be drawn.

This method works between `Framebuffer` instances utilising different formats, but the resulting colors may be unexpected due to the mismatch in color formats.

## Constants

`framebuf.MONO_VLSB`

Monochrome (1-bit) color format This defines a mapping where the bits in a byte are vertically mapped with bit 0 being nearest the top of the screen. Consequently each byte occupies 8 vertical pixels. Subsequent bytes appear at successive horizontal locations until the rightmost edge is reached. Further bytes are rendered at locations starting at the leftmost edge, 8 pixels lower.

`framebuf.MONO_HLSB`

Monochrome (1-bit) color format This defines a mapping where the bits in a byte are horizontally mapped. Each byte occupies 8 horizontal pixels with bit 0 being the leftmost. Subsequent bytes appear at successive horizontal locations until the rightmost edge is reached. Further bytes are rendered on the next row, one pixel lower.

`framebuf.MONO_HMSB`

Monochrome (1-bit) color format This defines a mapping where the bits in a byte are horizontally mapped. Each byte occupies 8 horizontal pixels with bit 7 being the leftmost. Subsequent bytes appear at successive horizontal locations until the rightmost edge is reached. Further bytes are rendered on the next row, one pixel lower.

`framebuf.RGB565`

Red Green Blue (16-bit, 5+6+5) color format

`framebuf.GS2_HMSB`

Grayscale (2-bit) color format

`framebuf.GS4_HMSB`

Grayscale (4-bit) color format

`framebuf.GS8`

Grayscale (8-bit) color format

## micropython – access and control MicroPython internals

**Warning:** Though this MicroPython-based library may be available for use in some builds of CircuitPython, it is unsupported and its functionality may change in the future, perhaps significantly. As CircuitPython continues to develop, it may be changed to comply more closely with the corresponding standard Python library. You will likely need to change your code later if you rely on any non-standard functionality it currently provides.

### Functions

`micropython.const(expr)`

Used to declare that the expression is a constant so that the compile can optimise it. The use of this function should be as follows:

```
from micropython import const

CONST_X = const(123)
CONST_Y = const(2 * CONST_X + 1)
```

Constants declared this way are still accessible as global variables from outside the module they are declared in. On the other hand, if a constant begins with an underscore then it is hidden, it is not available as a global variable, and does not take up any memory during execution.

This `const` function is recognised directly by the MicroPython parser and is provided as part of the `micropython` module mainly so that scripts can be written which run under both CPython and MicroPython, by following the above pattern.

`micropython.opt_level(level)`

If *level* is given then this function sets the optimisation level for subsequent compilation of scripts, and returns `None`. Otherwise it returns the current optimisation level.

The optimisation level controls the following compilation features:

- Assertions: at level 0 assertion statements are enabled and compiled into the bytecode; at levels 1 and higher assertions are not compiled.
- Built-in `__debug__` variable: at level 0 this variable expands to `True`; at levels 1 and higher it expands to `False`.
- Source-code line numbers: at levels 0, 1 and 2 source-code line number are stored along with the bytecode so that exceptions can report the line number they occurred at; at levels 3 and higher line numbers are not stored.

The default optimisation level is usually level 0.

`micropython.mem_info(verbose)`

Print information about currently used memory. If the *verbose* argument is given then extra information is printed.

The information that is printed is implementation dependent, but currently includes the amount of stack and heap used. In verbose mode it prints out the entire heap indicating which blocks are used and which are free.

`micropython.qstr_info(verbose)`

Print information about currently interned strings. If the *verbose* argument is given then extra information is printed.

The information that is printed is implementation dependent, but currently includes the number of interned strings and the amount of RAM they use. In verbose mode it prints out the names of all RAM-interned strings.

```
micropython.stack_use()
```

Return an integer representing the current amount of stack that is being used. The absolute value of this is not particularly useful, rather it should be used to compute differences in stack usage at different points.

```
micropython.heap_lock()
```

```
micropython.heap_unlock()
```

Lock or unlock the heap. When locked no memory allocation can occur and a `MemoryError` will be raised if any heap allocation is attempted.

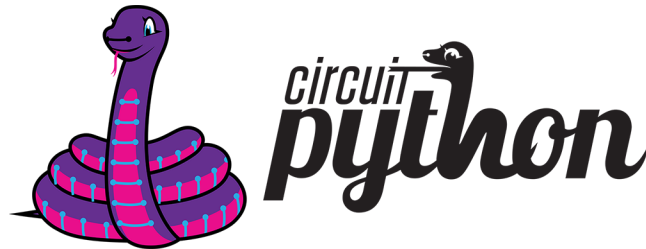
These functions can be nested, ie `heap_lock()` can be called multiple times in a row and the lock-depth will increase, and then `heap_unlock()` must be called the same number of times to make the heap available again.

```
micropython.kbd_intr(chr)
```

Set the character that will raise a `KeyboardInterrupt` exception. By default this is set to 3 during script execution, corresponding to Ctrl-C. Passing -1 to this function will disable capture of Ctrl-C, and passing 3 will restore it.

This function can be used to prevent the capturing of Ctrl-C on the incoming stream of characters that is usually used for the REPL, in case that stream is used for other purposes.

### 1.8.10 CircuitPython



[circuitpython.org](https://circuitpython.org) | [Get CircuitPython](#) | [Documentation](#) | [Contributing](#) | [Branding](#) | [Differences from Micropython](#) | [Project Structure](#)

**CircuitPython** is a *beginner friendly*, open source version of Python for tiny, inexpensive computers called microcontrollers. Microcontrollers are the brains of many electronics including a wide variety of development boards used to build hobby projects and prototypes. CircuitPython in electronics is one of the best ways to learn to code because it connects code to reality. Simply install CircuitPython on a supported board via drag and drop and then edit a `code.py` file on the CIRCUITPY drive. The code will automatically reload. No software installs are needed besides a text editor (we recommend [Mu](#) for beginners.)

CircuitPython features unified Python core APIs and a growing list of 150+ device libraries and drivers that work with it. These libraries also work on single board computers with regular Python via the [Adafruit Blinka Library](#).

CircuitPython is based on [MicroPython](#). See [below](#) for differences. CircuitPython development is sponsored by [Adafruit](#) and is available on their educational development boards. Please support both MicroPython and Adafruit.



## Get CircuitPython

Official binaries for all supported boards are available through [circuitpython.org/downloads](https://circuitpython.org/downloads). The site includes stable, unstable and continuous builds. Full release notes and assets are available through [GitHub releases](#) as well.

## Documentation

Guides and videos are available through the [Adafruit Learning System](#) under the [CircuitPython](#) category. An API reference is also available on [Read the Docs](#). A collection of awesome resources can be found at [Awesome CircuitPython](#).

Specifically useful documentation when starting out:

- [Welcome to CircuitPython](#)
- [CircuitPython Essentials](#)
- [Example Code](#)

## Code Search

GitHub doesn't currently support code search on forks. Therefore, CircuitPython doesn't have code search through GitHub because it is a fork of MicroPython. Luckily, [SourceGraph](#) has free code search for public repos like CircuitPython. So, visit [sourcegraph.com/github.com/adafruit/circuitpython](https://sourcegraph.com/github.com/adafruit/circuitpython) to search the CircuitPython codebase online.

## Contributing

See [CONTRIBUTING.md](#) for full guidelines but please be aware that by contributing to this project you are agreeing to the [Code of Conduct](#). Contributors who follow the [Code of Conduct](#) are welcome to submit pull requests and they will be promptly reviewed by project admins. Please join the [Discord](#) too.

## Branding

While we are happy to see CircuitPython forked and modified, we'd appreciate it if forked releases not use the name "CircuitPython" or the Blinka logo. "CircuitPython" means something special to us and those who learn about it. As a result, we'd like to make sure products referring to it meet a common set of requirements.

If you'd like to use the term "CircuitPython" and Blinka for your product here is what we ask:

- Your product is supported by the primary ["adafruit/circuitpython"](#) repo. This way we can update any custom code as we update the CircuitPython internals.
- Your product is listed on [circuitpython.org](https://circuitpython.org) (source [here](#)). This is to ensure that a user of your product can always download the latest version of CircuitPython from the standard place.
- Your product has a user accessible USB plug which appears as a CIRCUITPY drive when plugged in.

If you choose not to meet these requirements, then we ask you call your version of CircuitPython something else (for example, SuperDuperPython) and not use the Blinka logo. You can say it is "CircuitPython-compatible" if most CircuitPython drivers will work with it.

## Differences from MicroPython

CircuitPython:

- Supports native USB on all boards, allowing file editing without special tools.
- Floats (aka decimals) are enabled for all builds.
- Error messages are translated into 10+ languages.
- Does not support concurrency within Python (including interrupts and threading). Some concurrency is achieved with native modules for tasks that require it such as audio file playback.

## Behavior

- The order that files are run and the state that is shared between them. CircuitPython's goal is to clarify the role of each file and make each file independent from each other.
- `boot.py` (or `settings.py`) runs only once on start up before USB is initialized. This lays the ground work for configuring USB at startup rather than it being fixed. Since serial is not available, output is written to `boot_out.txt`.
- `code.py` (or `main.py`) is run after every reload until it finishes or is interrupted. After it is done running, the vm and hardware is reinitialized. **This means you cannot read state from `code.py` in the REPL anymore, as the REPL is a fresh vm.** CircuitPython's goal for this change includes reducing confusion about pins and memory being used.
- After the main code is finished the REPL can be entered by pressing any key.
- Autoreload state will be maintained across reload.
- Adds a safe mode that does not run user code after a hard crash or brown out. The hope is that this will make it easier to fix code that causes nasty crashes by making it available through mass storage after the crash. A reset (the button) is needed after it's fixed to get back into normal mode.
- RGB status LED indicating CircuitPython state, and errors through a sequence of colored flashes.
- Re-runs `code.py` or other main file after file system writes over USB mass storage. (Disable with `supervisor.disable_autoreload()`)
- Autoreload is disabled while the REPL is active.
- Main is one of these: `code.txt`, `code.py`, `main.py`, `main.txt`
- Boot is one of these: `settings.txt`, `settings.py`, `boot.py`, `boot.txt`

## API

- Unified hardware APIs. Documented on [ReadTheDocs](#).
- API docs are rST within the C files in `shared-bindings`.
- No machine API.

## Modules

- No module aliasing. (`uos` and `utime` are not available as `os` and `time` respectively.) Instead `os`, `time`, and `random` are CPython compatible.
  - New `storage` module which manages file system mounts. (Functionality from `uos` in MicroPython.)
  - Modules with a CPython counterpart, such as `time`, `os` and `random`, are strict [subsets](#) of their CPython [version](#). Therefore, code from CircuitPython is runnable on CPython but not necessarily the reverse.
  - tick count is available as `time.monotonic()`
- 

## Project Structure

Here is an overview of the top-level source code directories.

### Core

The core code of [MicroPython](#) is shared amongst ports including CircuitPython:

- `docs` High level user documentation in Sphinx reStructuredText format.
- `drivers` External device drivers written in Python.
- `examples` A few example Python scripts.
- `extmod` Shared C code used in multiple ports' modules.
- `lib` Shared core C code including externally developed libraries such as FATFS.
- `logo` The CircuitPython logo.
- `mpy-cross` A cross compiler that converts Python files to byte code prior to being run in MicroPython. Useful for reducing library size.
- `py` Core Python implementation, including compiler, runtime, and core library.
- `shared-bindings` Shared definition of Python modules, their docs and backing C APIs. Ports must implement the C API to support the corresponding module.
- `shared-module` Shared implementation of Python modules that may be based on `common-hal`.
- `tests` Test framework and test scripts.
- `tools` Various tools, including the `pyboard.py` module.

### Ports

Ports include the code unique to a microcontroller line and also variations based on the board.

Supported	Support status
atmel-samd	SAMD21 stable   SAMD51 stable
cx-d56	stable
esp32s2	beta
litex	alpha
mimxrt10xx	alpha
nrf	stable
raspberrypi	beta
stm	F4 stable   others beta
unix	alpha

- `stable` Highly unlikely to have bugs or missing functionality.
- `beta` Being actively improved but may be missing functionality and have bugs.
- `alpha` Will have bugs and missing functionality.

The remaining port directories not listed above are in the repo to maintain compatibility with the [MicroPython](#) parent project.

[Back to Top](#)

## 1.8.11 Contributing

Please note that this project is released with a [Contributor Code of Conduct](#). By participating in this project you agree to abide by its terms. Participation covers any forum used to converse about CircuitPython including unofficial and official spaces. Failure to do so will result in corrective actions such as time out or ban from the project.

### Licensing

By contributing to this repository you are certifying that you have all necessary permissions to license the code under an MIT License. You still retain the copyright but are granting many permissions under the MIT License.

If you have an employment contract with your employer please make sure that they don't automatically own your work product. Make sure to get any necessary approvals before contributing. Another term for this contribution off-hours is moonlighting.

### Ways to contribute

As CircuitPython grows, there are more and more ways to contribute. Here are some ideas:

- Build a project with CircuitPython and share how to do it online.
- Test the latest libraries and CircuitPython versions with your projects and file issues for any bugs you find.
- Contribute Python code to CircuitPython libraries that support new devices or features of an existing device.
- Contribute C code to CircuitPython which fixes an open issue or adds a new feature.

## Getting started with C

CircuitPython developer Dan Halbert (@dhalbert) has written up build instructions using native build tools [here](#).

For SAMD21 debugging workflow tips check out [this learn guide](#) from Scott (@tannewt).

## Developer contacts

Scott Shawcroft (@tannewt) is the lead developer of CircuitPython and is sponsored by [Adafruit Industries LLC](#). Scott is usually available during US West Coast working hours. Dan Halbert (@dhalbert) and Kattni Rembor (@kattni) are also sponsored by [Adafruit Industries LLC](#) and are usually available during US East Coast daytime hours including some weekends.

They are all reachable on [Discord](#), GitHub issues and the [Adafruit support forum](#).

## Code guidelines

We aim to keep our code and commit style compatible with MicroPython upstream. Please review their [code conventions](#) to do so. Familiarity with their [design philosophy](#) is also useful though not always applicable to CircuitPython.

Furthermore, CircuitPython has a [design guide](#) that covers a variety of different topics. Please read it as well.

## 1.8.12 Building CircuitPython

Welcome to CircuitPython!

This document is a quick-start guide only.

Detailed guides on how to build CircuitPython can be found in the Adafruit Learn system at <https://learn.adafruit.com/building-circuitpython/>

## Setup

Please ensure you setup your build environment appropriately, as per the guide. You will need:

- Linux: <https://learn.adafruit.com/building-circuitpython/linux>
- MacOS: <https://learn.adafruit.com/building-circuitpython/macos>
- Windows Subsystem for Linux (WSL): <https://learn.adafruit.com/building-circuitpython/windows-subsystem-for-linux>

## Submodules

This project has a bunch of git submodules. You will need to update them regularly.

```
git submodule sync
git submodule update --init
```

## mpy-cross

As part of the build process, mpy-cross is needed to compile .py files into .mpy files. To compile (or recompile) mpy-cross:

```
make -C mpy-cross
```

### 1.8.13 Building

There a number of ports of CircuitPython! To build for your board, change to the appropriate ports directory and build.

Examples:

```
cd ports/atmel-samd
make BOARD=circuitplayground_express

cd ports/nrf
make BOARD=circuitplayground_bluefruit
```

If you aren't sure what boards exist, have a peek in the boards subdirectory of your port. If you have a fast computer with many cores, consider adding `-j` to your build flags, such as `-j17` on a 6-core 12-thread machine.

### 1.8.14 Testing

If you are working on changes to the core language, you might find it useful to run the test suite. The test suite in the top level `tests` directory. It needs the unix port to run.

```
cd ports/unix
make axtls
make micropython
```

Then you can run the test suite:

```
cd ../../tests
./run-tests
```

A successful run will say something like

```
676 tests performed (19129 individual testcases)
676 tests passed
30 tests skipped: buffered_writer builtin_help builtin_range_binop class_delattr_
↪setattr cmd_parsetree extra_coverage framebuf1 framebuf16 framebuf2 framebuf4_
↪framebuf8 framebuf_subclass mpy_invalid namedtuple_asdict non_compliant resource_
↪stream schedule sys_getsizeof urandom_extra ure_groups ure_span ure_sub ure_sub_
↪unmatched vfs_basic vfs_fat_fileio1 vfs_fat_fileio2 vfs_fat_more vfs_fat_oldproto_
↪vfs_fat_ramdisk vfs_userfs
```

### 1.8.15 Debugging

The easiest way to debug CircuitPython on hardware is with a JLink device, JLinkGDBServer, and an appropriate GDB. Instructions can be found at <https://learn.adafruit.com/debugging-the-samd21-with-gdb>

If using JLink, you'll need both the JLinkGDBServer and arm-none-eabi-gdb running.

Example:

```
JLinkGDBServer -if SWD -device ATSAMD51J19
arm-none-eabi-gdb build-metro_m4_express/firmware.elf -iex "target extended-remote_
↳:2331"
```

If your port/build includes arm-none-eabi-gdb-py, consider using it instead, as it can be used for better register debugging with <https://github.com/bnahill/PyCortexMDebug>

### 1.8.16 Code Quality Checks

We apply code quality checks using pre-commit. Install pre-commit once per system with

```
python3 -mpip install pre-commit
```

Activate it once per git clone with

```
pre-commit --install
```

Pre-commit also requires some additional programs to be installed through your package manager:

- Standard Unix tools such as make, find, etc
- The gettext package, any modern version
- uncrustify version 0.71 (0.72 is also tested)

Each time you create a git commit, the pre-commit quality checks will be run. You can also run them e.g., with `pre-commit run foo.c` or `pre-commit run --all` to run on all files whether modified or not.

Some pre-commit quality checks require your active attention to resolve, others (such as the formatting checks of uncrustify) are made automatically and must simply be incorporated into your code changes by committing them.

### 1.8.17 Adafruit Community Code of Conduct

#### Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and leaders pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level or type of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

## Our Standards

We are committed to providing a friendly, safe and welcoming environment for all.

Examples of behavior that contributes to creating a positive environment include:

- Be kind and courteous to others
- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Collaborating with other community members
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and sexual attention or advances
- The use of inappropriate images, including in a community member's avatar
- The use of inappropriate language, including in a community member's nickname
- Any spamming, flaming, baiting or other attention-stealing behavior
- Excessive or unwelcome helping; answering outside the scope of the question asked
- Trolling, insulting/derogatory comments, and personal or political attacks
- Promoting or spreading disinformation, lies, or conspiracy theories against a person, group, organisation, project, or community
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate

The goal of the standards and moderation guidelines outlined here is to build and maintain a respectful community. We ask that you don't just aim to be "technically unimpeachable", but rather try to be your best self.

We value many things beyond technical expertise, including collaboration and supporting others within our community. Providing a positive experience for other community members can have a much more significant impact than simply providing the correct answer.

## Our Responsibilities

Project leaders are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project leaders have the right and responsibility to remove, edit, or reject messages, comments, commits, code, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any community member for other behaviors that they deem inappropriate, threatening, offensive, or harmful.



## Moderation

Instances of behaviors that violate the Adafruit Community Code of Conduct may be reported by any member of the community. Community members are encouraged to report these situations, including situations they witness involving other community members.

You may report in the following ways:

In any situation, you may send an email to [support@adafruit.com](mailto:support@adafruit.com).

On the Adafruit Discord, you may send an open message from any channel to all Community Moderators by tagging @community moderators. You may also send an open message from any channel, or a direct message to @kattni#1507, @tannewt#4653, @danh#1614, @cater#2442, @sommersoft#0222, @Mr. Certainly#0472 or @Andon#8175.

Email and direct message reports will be kept confidential.

In situations on Discord where the issue is particularly egregious, possibly illegal, requires immediate action, or violates the Discord terms of service, you should also report the message directly to Discord.

These are the steps for upholding our community's standards of conduct.

1. Any member of the community may report any situation that violates the Adafruit Community Code of Conduct. All reports will be reviewed and investigated.
2. If the behavior is an egregious violation, the community member who committed the violation may be banned immediately, without warning.
3. Otherwise, moderators will first respond to such behavior with a warning.
4. Moderators follow a soft "three strikes" policy - the community member may be given another chance, if they are receptive to the warning and change their behavior.
5. If the community member is unreceptive or unreasonable when warned by a moderator, or the warning goes unheeded, they may be banned for a first or second offense. Repeated offenses will result in the community member being banned.

## Scope

This Code of Conduct and the enforcement policies listed above apply to all Adafruit Community venues. This includes but is not limited to any community spaces (both public and private), the entire Adafruit Discord server, and Adafruit GitHub repositories. Examples of Adafruit Community spaces include but are not limited to meet-ups, audio chats on the Adafruit Discord, or interaction at a conference.

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. As a community member, you are representing our community, and are expected to behave accordingly.

## Attribution

This Code of Conduct is adapted from the [Contributor Covenant][homepage], version 1.4, available at <https://www.contributor-covenant.org/version/1/4/code-of-conduct.html>, and the [Rust Code of Conduct](#).

For other projects adopting the Adafruit Community Code of Conduct, please contact the maintainers of those projects for enforcement. If you wish to use this code of conduct for your own project, consider explicitly mentioning your moderation policy or making a copy with your own moderation policy so as to avoid confusion.

### 1.8.18 MicroPython & CircuitPython license information

The MIT License (MIT)

Copyright (c) 2013-2017 Damien P. George, and others

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

### 1.8.19 WebUSB Serial Support

To date, this has only been tested on one port (esp32s2), on one board (espressif\_kaluga\_1).

#### What it does

If you have ever used CircuitPython on a platform with a graphical LCD display, you have probably already seen multiple “consoles” in use (although the LCD console is “output only”).

New compile-time option `CIRCUITPY_USB_VENDOR` enables an additional “console” that can be used in parallel with the original (CDC) serial console.

Web pages that support the WebUSB standard can connect to the “vendor” interface and activate this WebUSB serial console at any time.

You can type into either console, and CircuitPython output is sent to all active consoles.

One example of a web page you can use to test drive this feature can be found at:

[https://adafruit.github.io/Adafruit\\_TinyUSB\\_Arduino/examples/webusb-serial/index.html](https://adafruit.github.io/Adafruit_TinyUSB_Arduino/examples/webusb-serial/index.html)

#### How to enable

Update your platform’s `mpconfigboard.mk` file to enable and disable specific types of USB interfaces.

`CIRCUITPY_USB_HID = xxx CIRCUITPY_USB_MIDI = xxx CIRCUITPY_USB_VENDOR = xxx`

On at least some of the hardware platforms, the maximum number of USB endpoints is fixed. For example, on the ESP32S2, you must pick only one of the above 3 interfaces to be enabled.

Original espressif\_kaluga\_1 `mpconfigboard.mk` settings:

`CIRCUITPY_USB_HID = 1 CIRCUITPY_USB_MIDI = 0 CIRCUITPY_USB_VENDOR = 0`

Settings to enable WebUSB instead:

`CIRCUITPY_USB_HID = 0 CIRCUITPY_USB_MIDI = 0 CIRCUITPY_USB_VENDOR = 1`

Notice that to enable VENDOR on ESP32-S2, we had to give up HID. There may be platforms that can have both, or even all three.

## Implementation Notes

CircuitPython uses the tinyusb library.

The tinyusb library already has support for WebUSB serial. The tinyusb examples already include a “WebUSB serial” example.

Sidenote - The use of the term “`vendor`” instead of “`WebUSB`” was done to match tinyusb.

Basically, this feature was ported into CircuitPython by pulling code snippets out of the tinyusb example, and putting them where they best belonged in the CircuitPython codebase.

There was one complication:

tinyusb uses C preprocessor macros to define things like USB descriptors.

CircuitPython uses a Python program (`tools/gen_usb_descriptor.py`) to create USB descriptors (etc.) using “helper objects” from another repo (`adafruit_usb_descriptor`). This means some of the example code had to be adapted to the new programming model, and `gen_usb_descriptor` gained new command-line options to control the generated code.

The generated files go into the “build” directory, look for `autogen_usb_descriptor.c` and `genhdr/autogen_usb_descriptor.h`.

Also worth pointing out - the re-use of the CDC connect/disconnect mechanism is not actually part of the WebUSB standard, it’s more of “common idiom”. We make use of it here because we need to know when we should be paying attention to the WebUSB serial interface, and when we should ignore it..

## Possible future work areas

The current code uses the existing Python infrastructure to create the Interface descriptor, but simply outputs the code snippets from the original tinyusb demo code to create the `WEBUSB_URL`, `BOS`, and `MS_OS_20` descriptors. I suppose additional work could be done to add these to the `adafruit_usb_descriptor` project, and then `gen_usb_descriptor.py` could be modified to make use of them.

Program `gen_usb_descriptor.py` creates objects for most interface types, regardless of whether or not they are actually enabled. This increases the size of a generated string table. I made the new vendor-interface-related code not do this (because some of the ARM platforms would no longer build), but I did not go back and do this for the other interface types (CDC, MIDI, HID, etc.) Some FLASH savings are probably possible if this is done.



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

—  
\_bleio, 35  
\_eve, 45  
\_pew, 53  
\_pixelbuf, 54  
\_stage, 55  
\_typing, 56

### a

adafruit\_bus\_device, 57  
aesio, 59  
alarm, 59  
alarm.pin, 60  
alarm.time, 60  
alarm.touch, 61  
analogio, 62  
array, 201  
audiobusio, 64  
audiocore, 66  
audioio, 68  
audiomixer, 70  
audiomp3, 72  
audiopwmio, 73

### b

binascii, 201  
bitbangio, 74  
bitmaptools, 79  
bitops, 81  
board, 82  
btree, 222  
busio, 82

### c

camera, 89  
canio, 90  
collections, 202  
countio, 94

### d

digitalio, 95  
displayio, 97

dualbank, 109

### e

errno, 203

### f

fontio, 110  
framebuf, 224  
framebufferio, 110  
frequencyio, 112

### g

gamepad, 113  
gamepadshift, 114  
gc, 203  
gnss, 115

### h

hashlib, 204

### i

i2cperipheral, 116  
io, 205  
ipaddress, 119

### j

json, 207

### m

math, 119  
memorymonitor, 121  
microcontroller, 123  
micropython, 227  
msgpack, 125  
multiterminal, 126

### n

neopixel\_write, 127  
network, 127  
nvm, 127

### o

os, 128

## p

ps2io, [129](#)  
pulseio, [131](#)  
pwmio, [134](#)

## r

random, [135](#)  
re, [207](#)  
rgbmatrix, [136](#)  
rotaryio, [137](#)  
rp2pio, [138](#)  
rtc, [141](#)

## s

samd, [142](#)  
sdcardio, [142](#)  
sdioio, [143](#)  
sharpdisplay, [145](#)  
socket, [145](#)  
socketpool, [147](#)  
ssl, [149](#)  
storage, [150](#)  
struct, [151](#)  
supervisor, [152](#)  
sys, [208](#)

## t

terminalio, [153](#)  
time, [153](#)  
touchio, [154](#)

## u

uctypes, [210](#)  
uheap, [155](#)  
uheapq, [200](#)  
ulab, [156](#)  
ulab.approx, [156](#)  
ulab.compare, [157](#)  
ulab.fft, [158](#)  
ulab.filter, [158](#)  
ulab.linalg, [159](#)  
ulab.numerical, [160](#)  
ulab.poly, [161](#)  
ulab.user, [161](#)  
ulab.vector, [161](#)  
usb\_cdc, [166](#)  
usb\_hid, [167](#)  
usb\_midi, [168](#)  
uselect, [213](#)  
usocket, [215](#)  
ussl, [220](#)  
ustack, [169](#)  
uzlib, [221](#)

## v

vectorio, [169](#)

## w

watchdog, [170](#)  
wifi, [171](#)  
wiznet, [172](#)



## Symbols

- `__ArrayLike` (in module `ulab`), 164
- `__DType` (in module `ulab`), 162
- `__DisplayBus` (in module `displayio`), 99
- `__EVE` (class in `_eve`), 45
- `__Index` (in module `ulab`), 162
- `__T` (in module `random`), 135
- `__Uname` (class in `os`), 128
- `__abs__` () (`ulab.array` method), 164
- `__add__` () (`ulab.array` method), 163
- `__bool__` () (`alarm.SleepMemory` method), 62
- `__bool__` () (`displayio.Group` method), 105
- `__bool__` () (`displayio.Palette` method), 106
- `__bool__` () (`nvm.ByteArray` method), 128
- `__bool__` () (`ps2io.Ps2` method), 131
- `__bool__` () (`pulseio.PulseIn` method), 132
- `__contains__` () (`btree.btree` method), 223
- `__delitem__` () (`displayio.Group` method), 105
- `__detitem__` () (`btree.btree` method), 223
- `__div__` () (`ulab.array` method), 164
- `__enter__` () (`adafruit_bus_device.I2CDevice` method), 57
- `__enter__` () (`adafruit_bus_device.SPIDevice` method), 58
- `__enter__` () (`analogio.AnalogIn` method), 63
- `__enter__` () (`analogio.AnalogOut` method), 63
- `__enter__` () (`audiobusio.I2SOut` method), 65
- `__enter__` () (`audiobusio.PDMIn` method), 66
- `__enter__` () (`audiocore.RawSample` method), 67
- `__enter__` () (`audiocore.WaveFile` method), 68
- `__enter__` () (`audioio.AudioOut` method), 69
- `__enter__` () (`audiomixer.Mixer` method), 71
- `__enter__` () (`audiomp3.MP3Decoder` method), 72
- `__enter__` () (`audiopwmio.PWMAudioOut` method), 74
- `__enter__` () (`bitbangio.I2C` method), 75
- `__enter__` () (`bitbangio.OneWire` method), 77
- `__enter__` () (`bitbangio.SPI` method), 78
- `__enter__` () (`busio.I2C` method), 83
- `__enter__` () (`busio.OneWire` method), 85
- `__enter__` () (`busio.SPI` method), 86
- `__enter__` () (`busio.UART` method), 88
- `__enter__` () (`canio.CAN` method), 92
- `__enter__` () (`canio.Listener` method), 92
- `__enter__` () (`countio.Counter` method), 94
- `__enter__` () (`digitalio.DigitalInOut` method), 96
- `__enter__` () (`frequencyio.FrequencyIn` method), 113
- `__enter__` () (`i2cperipheral.I2CPeripheral` method), 117
- `__enter__` () (`i2cperipheral.I2CPeripheralRequest` method), 118
- `__enter__` () (`memorymonitor.AllocationAlarm` method), 121
- `__enter__` () (`memorymonitor.AllocationSize` method), 122
- `__enter__` () (`ps2io.Ps2` method), 130
- `__enter__` () (`pulseio.PulseIn` method), 132
- `__enter__` () (`pulseio.PulseOut` method), 133
- `__enter__` () (`pwmio.PWMOut` method), 135
- `__enter__` () (`rotaryio.IncrementalEncoder` method), 137
- `__enter__` () (`rp2pio.StateMachine` method), 140
- `__enter__` () (`sdioio.SDCard` method), 145
- `__enter__` () (`socketpool.Socket` method), 147
- `__enter__` () (`ssl.SSLSocket` method), 149
- `__enter__` () (`touchio.TouchIn` method), 155
- `__eq__` () (`_bleio.Address` method), 38
- `__eq__` () (`_bleio.UUID` method), 45
- `__eq__` () (`ipaddress.IPv4Address` method), 119
- `__exit__` () (`adafruit_bus_device.I2CDevice` method), 57
- `__exit__` () (`adafruit_bus_device.SPIDevice` method), 58
- `__exit__` () (`analogio.AnalogIn` method), 63
- `__exit__` () (`analogio.AnalogOut` method), 63
- `__exit__` () (`audiobusio.I2SOut` method), 65
- `__exit__` () (`audiobusio.PDMIn` method), 66
- `__exit__` () (`audiocore.RawSample` method), 67
- `__exit__` () (`audiocore.WaveFile` method), 68
- `__exit__` () (`audioio.AudioOut` method), 69
- `__exit__` () (`audiomixer.Mixer` method), 71
- `__exit__` () (`audiomp3.MP3Decoder` method), 72
- `__exit__` () (`audiopwmio.PWMAudioOut` method), 74
- `__exit__` () (`bitbangio.I2C` method), 75

`__exit__()` (*bitbangio.OneWire method*), 77  
`__exit__()` (*bitbangio.SPI method*), 78  
`__exit__()` (*busio.I2C method*), 83  
`__exit__()` (*busio.OneWire method*), 85  
`__exit__()` (*busio.SPI method*), 86  
`__exit__()` (*busio.UART method*), 88  
`__exit__()` (*canio.CAN method*), 92  
`__exit__()` (*canio.Listener method*), 92  
`__exit__()` (*countio.Counter method*), 94  
`__exit__()` (*digitalio.DigitalInOut method*), 96  
`__exit__()` (*frequencyio.FrequencyIn method*), 113  
`__exit__()` (*i2cperipheral.I2CPeripheral method*), 117  
`__exit__()` (*i2cperipheral.I2CPeripheralRequest method*), 118  
`__exit__()` (*memorymonitor.AllocationAlarm method*), 122  
`__exit__()` (*memorymonitor.AllocationSize method*), 122  
`__exit__()` (*ps2io.Ps2 method*), 130  
`__exit__()` (*pulseio.PulseIn method*), 132  
`__exit__()` (*pulseio.PulseOut method*), 133  
`__exit__()` (*pwmio.PWMOut method*), 135  
`__exit__()` (*rotaryio.IncrementalEncoder method*), 137  
`__exit__()` (*rp2pio.StateMachine method*), 140  
`__exit__()` (*sdioio.SDCard method*), 145  
`__exit__()` (*socketpool.Socket method*), 147  
`__exit__()` (*ssl.SSLSocket method*), 149  
`__exit__()` (*touchio.TouchIn method*), 155  
`__ge__()` (*ulab.array method*), 164  
`__get__()` (*frequencyio.FrequencyIn method*), 113  
`__getitem__()` (*\_pixelbuf.PixelBuf method*), 54  
`__getitem__()` (*alarm.SleepMemory method*), 62  
`__getitem__()` (*btree.btree method*), 223  
`__getitem__()` (*displayio.Bitmap method*), 97  
`__getitem__()` (*displayio.Group method*), 105  
`__getitem__()` (*displayio.Palette method*), 107  
`__getitem__()` (*displayio.TileGrid method*), 109  
`__getitem__()` (*memorymonitor.AllocationSize method*), 122  
`__getitem__()` (*nvm.ByteArray method*), 128  
`__getitem__()` (*pulseio.PulseIn method*), 132  
`__getitem__()` (*ulab.array method*), 164  
`__gt__()` (*ulab.array method*), 164  
`__hash__()` (*\_bleio.Address method*), 38  
`__hash__()` (*ipaddress.IPv4Address method*), 119  
`__hash__()` (*socketpool.Socket method*), 148  
`__hash__()` (*ssl.SSLSocket method*), 150  
`__inv__()` (*ulab.array method*), 164  
`__iter__()` (*\_bleio.ScanResults method*), 44  
`__iter__()` (*btree.btree method*), 224  
`__iter__()` (*canio.Listener method*), 92  
`__iter__()` (*ulab.array method*), 164  
`__iter__()` (*wifi.ScannedNetworks method*), 172  
`__le__()` (*ulab.array method*), 164  
`__len__()` (*alarm.SleepMemory method*), 62  
`__len__()` (*displayio.Group method*), 105  
`__len__()` (*displayio.Palette method*), 107  
`__len__()` (*memorymonitor.AllocationSize method*), 122  
`__len__()` (*nvm.ByteArray method*), 128  
`__len__()` (*ps2io.Ps2 method*), 131  
`__len__()` (*pulseio.PulseIn method*), 132  
`__len__()` (*ulab.array method*), 164  
`__lt__()` (*ulab.array method*), 164  
`__mul__()` (*ulab.array method*), 164  
`__neg__()` (*ulab.array method*), 164  
`__next__()` (*\_bleio.ScanResults method*), 44  
`__next__()` (*canio.Listener method*), 92  
`__next__()` (*wifi.ScannedNetworks method*), 172  
`__pos__()` (*ulab.array method*), 164  
`__pow__()` (*ulab.array method*), 164  
`__radd__()` (*ulab.array method*), 164  
`__rdiv__()` (*ulab.array method*), 164  
`__rmul__()` (*ulab.array method*), 164  
`__rpow__()` (*ulab.array method*), 164  
`__rsub__()` (*ulab.array method*), 164  
`__setitem__()` (*\_pixelbuf.PixelBuf method*), 55  
`__setitem__()` (*alarm.SleepMemory method*), 62  
`__setitem__()` (*btree.btree method*), 223  
`__setitem__()` (*displayio.Bitmap method*), 98  
`__setitem__()` (*displayio.Group method*), 105  
`__setitem__()` (*displayio.Palette method*), 107  
`__setitem__()` (*displayio.TileGrid method*), 109  
`__setitem__()` (*nvm.ByteArray method*), 128  
`__setitem__()` (*ulab.array method*), 164  
`__sub__()` (*ulab.array method*), 164  
`_bleio`  
    module, 35  
`_bool` (in module *ulab*), 162  
`_eve`  
    module, 45  
`_float` (in module *ulab*), 162  
`_pew`  
    module, 53  
`_pixelbuf`  
    module, 54  
`_stage`  
    module, 55  
`_typing`  
    module, 56  
**A**  
`a2b_base64()` (in module *binascii*), 201  
`abs()`  
    built-in function, 197  
`accept()` (*socket.socket method*), 145

- `accept()` (*socketpool.Socket method*), 147
- `accept()` (*ssl.SSLSocket method*), 149
- `accept()` (*usocket.socket method*), 217
- `ack()` (*i2cperipheral.I2CPeripheralRequest method*), 118
- `acos()` (*in module math*), 119
- `acos()` (*in module ulab.vector*), 161
- `acosh()` (*in module math*), 121
- `acosh()` (*in module ulab.vector*), 161
- `adafruit_bus_device`
  - module, 57
- `Adapter` (*class in \_bleio*), 35
- `adapter` (*in module \_bleio*), 35
- `add_to_characteristic()` (*\_bleio.Descriptor class method*), 42
- `add_to_service()` (*\_bleio.Characteristic method*), 39
- `address` (*\_bleio.Adapter attribute*), 36
- `address` (*\_bleio.ScanEntry attribute*), 44
- `Address` (*class in \_bleio*), 37
- `address` (*i2cperipheral.I2CPeripheralRequest attribute*), 118
- `address_bytes` (*\_bleio.Address attribute*), 37
- `addressof()` (*in module ctypes*), 212
- `advertisement_bytes` (*\_bleio.ScanEntry attribute*), 44
- `advertising` (*\_bleio.Adapter attribute*), 36
- `AES` (*class in aesio*), 59
- `aesio`
  - module, 59
- `AF_INET` (*in module usocket*), 217
- `AF_INET` (*socket.socket attribute*), 145
- `AF_INET` (*socketpool.SocketPool attribute*), 148
- `AF_INET6` (*in module usocket*), 217
- `AF_INET6` (*socket.socket attribute*), 145
- `AF_INET6` (*socketpool.SocketPool attribute*), 148
- `alarm`
  - module, 59
- `Alarm` (*in module \_typing*), 57
- `alarm.pin`
  - module, 60
- `alarm.time`
  - module, 60
- `alarm.touch`
  - module, 61
- `all()`
  - built-in function, 197
- `AllocationAlarm` (*class in memorymonitor*), 121
- `AllocationError`, 121
- `AllocationSize` (*class in memorymonitor*), 122
- `AlphaFunc()` (*\_eve.\_EVE method*), 45
- `altitude` (*gnss.GNSS attribute*), 115
- `AnalogIn` (*class in analogio*), 62
- `analogio`
  - module, 62
- `AnalogOut` (*class in analogio*), 63
- `any()`
  - built-in function, 197
- `ap_info` (*wifi.Radio attribute*), 172
- `append()` (*array.array.array method*), 201
- `append()` (*displayio.Group method*), 105
- `arange()` (*in module ulab*), 165
- `arctan2()` (*in module ulab.vector*), 161
- `argmax()` (*in module ulab.numerical*), 160
- `argmin()` (*in module ulab.numerical*), 160
- `argsort()` (*in module ulab.numerical*), 160
- `argv` (*in module sys*), 209
- `around()` (*in module ulab.vector*), 161
- `array`
  - module, 201
- `array` (*class in ulab*), 163
- `array.array` (*class in array*), 201
- `arrayblit()` (*in module bitmaptools*), 80
- `asin()` (*in module math*), 119
- `asin()` (*in module ulab.vector*), 161
- `asinh()` (*in module math*), 121
- `asinh()` (*in module ulab.vector*), 161
- `AssertionError`, 199
- `atan()` (*in module math*), 119
- `atan()` (*in module ulab.vector*), 161
- `atan2()` (*in module math*), 119
- `atanh()` (*in module math*), 121
- `atanh()` (*in module ulab.vector*), 161
- `attribute` (*\_bleio.Connection attribute*), 41
- `Attribute` (*class in \_bleio*), 38
- `AttributeError`, 199
- `audiobusio`
  - module, 64
- `audiocore`
  - module, 66
- `audioio`
  - module, 68
- `audiomixer`
  - module, 70
- `audiomp3`
  - module, 72
- `AudioOut` (*class in audioio*), 68
- `audiopwmio`
  - module, 73
- `AudioSample` (*in module \_typing*), 56
- `authmode` (*wifi.Network attribute*), 171
- `auto_brightness` (*displayio.Display attribute*), 101
- `auto_brightness` (*framebufferio.FramebufferDisplay attribute*), 111
- `auto_refresh` (*displayio.Display attribute*), 100
- `auto_refresh` (*framebufferio.FramebufferDisplay attribute*), 111
- `AUTO_RELOAD` (*supervisor.RunReason attribute*), 152

`auto_restart` (*canio.CAN attribute*), 91  
`auto_write` (*\_pixelbuf.PixelBuf attribute*), 54

## B

`b2a_base64()` (*in module binascii*), 201  
`baudrate` (*busio.UART attribute*), 88  
`baudrate` (*canio.CAN attribute*), 91  
`Begin()` (*\_eve.\_EVE method*), 46  
`BIG_ENDIAN` (*in module ctypes*), 212  
`bin()`  
    built-in function, 197  
`binascii`  
    module, 201  
`bind()` (*socket.socket method*), 145  
`bind()` (*socketpool.Socket method*), 147  
`bind()` (*ssl.SSLSocket method*), 149  
`bind()` (*usocket.socket method*), 217  
`bisect()` (*in module ulab.approx*), 156  
`bit_transpose()` (*in module bitops*), 81  
`bitbangio`  
    module, 74  
`Bitmap` (*class in displayio*), 97  
`bitmap` (*fontio.BuiltinFont attribute*), 110  
`BitmapExtFormat()` (*\_eve.\_EVE method*), 46  
`BitmapHandle()` (*\_eve.\_EVE method*), 46  
`BitmapLayout()` (*\_eve.\_EVE method*), 46  
`BitmapLayoutH()` (*\_eve.\_EVE method*), 46  
`BitmapSize()` (*\_eve.\_EVE method*), 46  
`BitmapSizeH()` (*\_eve.\_EVE method*), 46  
`BitmapSource()` (*\_eve.\_EVE method*), 47  
`BitmapSwizzle()` (*\_eve.\_EVE method*), 47  
`bitmaptools`  
    module, 79  
`BitmapTransformA()` (*\_eve.\_EVE method*), 47  
`BitmapTransformB()` (*\_eve.\_EVE method*), 47  
`BitmapTransformC()` (*\_eve.\_EVE method*), 47  
`BitmapTransformD()` (*\_eve.\_EVE method*), 48  
`BitmapTransformE()` (*\_eve.\_EVE method*), 48  
`BitmapTransformF()` (*\_eve.\_EVE method*), 48  
`bitops`  
    module, 81  
`bits_per_sample` (*audiocore.WaveFile attribute*), 68  
`bits_per_sample` (*audiomp3.MP3Decoder attribute*), 72  
`BlendFunc()` (*\_eve.\_EVE method*), 48  
`blit()` (*displayio.Bitmap method*), 98  
`blit()` (*framebuf.FrameBuffer method*), 226  
`BluetoothError`, 35  
`board`  
    module, 82  
`bool` (*built-in class*), 197  
`bool` (*in module ulab*), 165  
`BOOTLOADER` (*microcontroller.RunMode attribute*), 125  
`bpp` (*\_pixelbuf.PixelBuf attribute*), 54

`brightness` (*\_pixelbuf.PixelBuf attribute*), 54  
`brightness` (*displayio.Display attribute*), 100  
`brightness` (*framebufferio.FramebufferDisplay attribute*), 111  
`brightness` (*rgbmatrix.RGBMatrix attribute*), 136  
`BROADCAST` (*\_bleio.Characteristic attribute*), 39  
`BROWNOUT` (*microcontroller.ResetReason attribute*), 124  
`bssid` (*wifi.Network attribute*), 171  
`btree`  
    module, 222  
built-in function  
    `abs()`, 197  
    `all()`, 197  
    `any()`, 197  
    `bin()`, 197  
    `callable()`, 197  
    `chr()`, 197  
    `classmethod()`, 197  
    `compile()`, 197  
    `delattr()`, 197  
    `dir()`, 198  
    `divmod()`, 198  
    `enumerate()`, 198  
    `eval()`, 198  
    `exec()`, 198  
    `filter()`, 198  
    `getattr()`, 198  
    `globals()`, 198  
    `hasattr()`, 198  
    `hash()`, 198  
    `help()`, 173  
    `hex()`, 198  
    `id()`, 198  
    `input()`, 198  
    `isinstance()`, 198  
    `issubclass()`, 198  
    `iter()`, 198  
    `len()`, 198  
    `locals()`, 198  
    `map()`, 198  
    `max()`, 198  
    `min()`, 198  
    `next()`, 198  
    `oct()`, 198  
    `open()`, 199  
    `ord()`, 199  
    `pow()`, 199  
    `print()`, 199  
    `property()`, 199  
    `range()`, 199  
    `repr()`, 199  
    `reversed()`, 199  
    `round()`, 199  
    `setattr()`, 199

- `sorted()`, 199
  - `staticmethod()`, 199
  - `sum()`, 199
  - `super()`, 199
  - `type()`, 199
  - `zip()`, 199
  - `BuiltinFont` (class in `fontio`), 110
  - `bus` (`displayio.Display` attribute), 101
  - `bus` (`displayio.EPaperDisplay` attribute), 103
  - `BUS_OFF` (`canio.BusState` attribute), 90
  - `busio`
    - module, 82
  - `BusState` (class in `canio`), 90
  - `busy` (`displayio.EPaperDisplay` attribute), 103
  - `bytearray` (built-in class), 197
  - `ByteArray` (class in `nvm`), 127
  - `bytearray_at()` (in module `uctypes`), 212
  - `byteorder` (`_pixelbuf.PixelBuf` attribute), 54
  - `byteorder` (in module `sys`), 209
  - `bytes` (built-in class), 197
  - `bytes_at()` (in module `uctypes`), 212
  - `bytes_per_block` (`memorymonitor.AllocationSize` attribute), 122
  - `BytesIO` (class in `io`), 206
- ## C
- `calcszize()` (in module `struct`), 151
  - `calibration` (`rtc.RTC` attribute), 141
  - `calibration` (`samd.Clock` attribute), 142
  - `Call()` (`_eve._EVE` method), 48
  - `callable()`
    - built-in function, 197
  - `camera`
    - module, 89
  - `Camera` (class in `camera`), 89
  - `CAN` (class in `canio`), 90
  - `canio`
    - module, 90
  - `capture_period` (`frequencyio.FrequencyIn` attribute), 113
  - `cc()` (`_eve._EVE` method), 45
  - `ceil()` (in module `math`), 119
  - `ceil()` (in module `ulab.vector`), 161
  - `Cell()` (`_eve._EVE` method), 48
  - `channel` (`wifi.Network` attribute), 171
  - `channel_count` (`audiocore.WaveFile` attribute), 68
  - `channel_count` (`audiomp3.MP3Decoder` attribute), 72
  - `characteristic` (`_bleio.Descriptor` attribute), 42
  - `Characteristic` (class in `_bleio`), 38
  - `CharacteristicBuffer` (class in `_bleio`), 40
  - `characteristics` (`_bleio.Service` attribute), 44
  - `chdir()` (in module `os`), 128
  - `choice()` (in module `random`), 136
  - `cholesky()` (in module `ulab.linalg`), 159
  - `chr()`
    - built-in function, 197
  - `Circle` (class in `vectorio`), 169
  - `classmethod()`
    - built-in function, 197
  - `Clear()` (`_eve._EVE` method), 49
  - `clear()` (`frequencyio.FrequencyIn` method), 113
  - `clear()` (`pulseio.PulseIn` method), 132
  - `clear_errors()` (`ps2io.Ps2` method), 130
  - `clear_rxfifo()` (`rp2pio.StateMachine` method), 141
  - `clear_secondary_terminal()` (in module `multi-terminal`), 126
  - `ClearColorA()` (`_eve._EVE` method), 49
  - `ClearColorRGB()` (`_eve._EVE` method), 49
  - `ClearStencil()` (`_eve._EVE` method), 49
  - `ClearTag()` (`_eve._EVE` method), 49
  - `clip()` (in module `ulab.compare`), 157
  - `Clock` (class in `samd`), 142
  - `close()` (`btree.btree` method), 223
  - `close()` (`socketpool.Socket` method), 147
  - `close()` (`ssl.SSLSocket` method), 149
  - `close()` (`usocket.socket` method), 217
  - `cmd()` (`_eve._EVE` method), 53
  - `cmd0()` (`_eve._EVE` method), 53
  - `code` (`msgpack.ExtType` attribute), 126
  - `collect()` (in module `gc`), 203
  - `collections`
    - module, 202
  - `ColorA()` (`_eve._EVE` method), 49
  - `ColorConverter` (class in `displayio`), 98
  - `ColorMask()` (`_eve._EVE` method), 49
  - `ColorRGB()` (`_eve._EVE` method), 50
  - `colorwheel()` (in module `_pixelbuf`), 54
  - `compile()`
    - built-in function, 197
  - `compile()` (in module `re`), 208
  - `complex` (built-in class), 197
  - `concatenate()` (in module `ulab`), 165
  - `configure()` (`bitbangio.SPI` method), 78
  - `configure()` (`busio.SPI` method), 86
  - `configure()` (`sdioio.SDCard` method), 144
  - `connect()` (`_bleio.Adapter` method), 37
  - `connect()` (`socket.socket` method), 146
  - `connect()` (`socketpool.Socket` method), 147
  - `connect()` (`ssl.SSLSocket` method), 149
  - `connect()` (`usocket.socket` method), 217
  - `connect()` (`wifi.Radio` method), 172
  - `connectable` (`_bleio.ScanEntry` attribute), 44
  - `connected` (`_bleio.Adapter` attribute), 36
  - `connected` (`_bleio.Connection` attribute), 41
  - `connected` (`usb_cdc.Serial` attribute), 166
  - `connected` (`wiznet.WIZNET5K` attribute), 173
  - `Connection` (class in `_bleio`), 41



`connection_interval` (*\_bleio.Connection* attribute), 41  
`connections` (*\_bleio.Adapter* attribute), 36  
`const()` (in module *micropython*), 227  
`convert()` (*displayio.ColorConverter* method), 98  
`convolve()` (in module *ulab.filter*), 158  
`copy()` (*ulab.array* method), 163  
`copysign()` (in module *math*), 119  
`cos()` (in module *math*), 119  
`cos()` (in module *ulab.vector*), 161  
`cosh()` (in module *math*), 120  
`cosh()` (in module *ulab.vector*), 161  
`count` (*countio.Counter* attribute), 94  
`count()` (*sdcardio.SDCard* method), 143  
`count()` (*sdioio.SDCard* method), 144  
`Counter` (class in *countio*), 94  
`countio`  
     module, 94  
`country` (*wifi.Network* attribute), 171  
`cpu` (in module *microcontroller*), 123  
`cpus` (in module *microcontroller*), 123  
`create_default_context()` (in module *ssl*), 149  
`cross()` (in module *ulab.numerical*), 160

## D

`data` (*canio.Message* attribute), 93  
`data` (*msgpack.ExtType* attribute), 126  
`datetime` (*rtc.RTC* attribute), 141  
`DEBUG` (in module *re*), 208  
`DecompIO` (class in *uzlib*), 221  
`decompress()` (in module *uzlib*), 221  
`decrypt_into()` (*aesio.AES* method), 59  
`DEEP_SLEEP_ALARM` (*microcontroller.ResetReason* attribute), 124  
`degrees()` (in module *math*), 119  
`degrees()` (in module *ulab.vector*), 161  
`deinit()` (*\_bleio.CharacteristicBuffer* method), 41  
`deinit()` (*\_bleio.PacketBuffer* method), 43  
`deinit()` (*analogio.AnalogIn* method), 63  
`deinit()` (*analogio.AnalogOut* method), 63  
`deinit()` (*audiobusio.I2SOut* method), 65  
`deinit()` (*audiobusio.PDMIn* method), 66  
`deinit()` (*audiocore.RawSample* method), 67  
`deinit()` (*audiocore.WaveFile* method), 68  
`deinit()` (*audioio.AudioOut* method), 69  
`deinit()` (*audiomixer.Mixer* method), 71  
`deinit()` (*audiomp3.MP3Decoder* method), 72  
`deinit()` (*audiopwmio.PWMAudioOut* method), 74  
`deinit()` (*bitbangio.I2C* method), 75  
`deinit()` (*bitbangio.OneWire* method), 77  
`deinit()` (*bitbangio.SPI* method), 77  
`deinit()` (*busio.I2C* method), 83  
`deinit()` (*busio.OneWire* method), 85  
`deinit()` (*busio.SPI* method), 86

`deinit()` (*busio.UART* method), 88  
`deinit()` (*camera.Camera* method), 89  
`deinit()` (*canio.CAN* method), 92  
`deinit()` (*canio.Listener* method), 92  
`deinit()` (*countio.Counter* method), 94  
`deinit()` (*digitalio.DigitalInOut* method), 96  
`deinit()` (*frequencyio.FrequencyIn* method), 113  
`deinit()` (*gamepad.GamePad* method), 114  
`deinit()` (*gamepadshift.GamePadShift* method), 115  
`deinit()` (*gnss.GNSS* method), 115  
`deinit()` (*i2cperipheral.I2CPeripheral* method), 117  
`deinit()` (*ps2io.Ps2* method), 130  
`deinit()` (*pulseio.PulseIn* method), 132  
`deinit()` (*pulseio.PulseOut* method), 133  
`deinit()` (*pwmio.PWMOut* method), 135  
`deinit()` (*rgbmatrix.RGBMatrix* method), 137  
`deinit()` (*rotaryio.IncrementalEncoder* method), 137  
`deinit()` (*rp2pio.StateMachine* method), 140  
`deinit()` (*sdcardio.SDCard* method), 143  
`deinit()` (*sdioio.SDCard* method), 145  
`deinit()` (*touchio.TouchIn* method), 155  
`deinit()` (*watchdog.WatchDogTimer* method), 170  
`delattr()`  
     built-in function, 197  
`delay_us()` (in module *microcontroller*), 123  
`DESC` (in module *btree*), 224  
`Descriptor` (class in *\_bleio*), 42  
`descriptors` (*\_bleio.Characteristic* attribute), 39  
`det()` (in module *ulab.linalg*), 159  
`Device` (class in *usb\_hid*), 167  
`devices` (in module *usb\_hid*), 167  
`dhcp` (*wiznet.WIZNET5K* attribute), 173  
`diag()` (in module *ulab*), 165  
`dict` (built-in class), 198  
`diff()` (in module *ulab.numerical*), 160  
`digest()` (*hashlib.hash* method), 205  
`DigitalInOut` (class in *digitalio*), 95  
`digitalio`  
     module, 95  
`dir()`  
     built-in function, 198  
`Direction` (class in *digitalio*), 96  
`direction` (*digitalio.DigitalInOut* attribute), 95  
`disable()` (in module *gc*), 203  
`disable_autoreload()` (in module *supervisor*), 152  
`disable_interrupts()` (in module *microcontroller*), 123  
`disconnect()` (*\_bleio.Connection* method), 41  
`discover_remote_services()`  
     (*\_bleio.Connection* method), 41  
`Display` (class in *displayio*), 99  
`Display()` (*\_eve.\_EVE* method), 50  
`displayio`

module, 97  
 dither (*displayio.ColorConverter* attribute), 98  
 divmod()  
     built-in function, 198  
 dot() (*in module ulab.linalg*), 159  
 DOWN (*digitalio.Pull* attribute), 97  
 draw\_line() (*in module bitmaptools*), 80  
 drive\_mode (*digitalio.DigitalInOut* attribute), 96  
 DriveMode (*class in digitalio*), 95  
 dtype() (*ulab.array* method), 163  
 dualbank  
     module, 109  
 dump() (*in module json*), 207  
 dumps() (*in module json*), 207  
 duty\_cycle (*pwmio.PWMOut* attribute), 135

## E

e (*in module math*), 119  
 eig() (*in module ulab.linalg*), 159  
 enable() (*in module gc*), 203  
 enable\_autoreload() (*in module supervisor*), 152  
 enable\_interrupts() (*in module microcontroller*), 123  
 enabled (*\_bleio.Adapter* attribute), 36  
 enabled (*samd.Clock* attribute), 142  
 enabled (*wifi.Radio* attribute), 171  
 encrypt\_into() (*aesio.AES* method), 59  
 ENCRYPT\_NO\_MITM (*\_bleio.Attribute* attribute), 38  
 ENCRYPT\_WITH\_MITM (*\_bleio.Attribute* attribute), 38  
 End() (*\_eve.\_EVE* method), 50  
 enumerate()  
     built-in function, 198  
 EPaperDisplay (*class in displayio*), 101  
 equal() (*in module ulab.compare*), 157  
 erase\_bonding() (*\_bleio.Adapter* method), 37  
 erase\_filesystem() (*in module storage*), 150  
 erf() (*in module math*), 121  
 erf() (*in module ulab.vector*), 161  
 erfc() (*in module math*), 121  
 erfc() (*in module ulab.vector*), 161  
 errno  
     module, 203  
 ERROR\_ACTIVE (*canio.BusState* attribute), 90  
 ERROR\_PASSIVE (*canio.BusState* attribute), 90  
 ERROR\_WARNING (*canio.BusState* attribute), 90  
 errorcode (*in module errno*), 203  
 eval()  
     built-in function, 198  
 EVEN (*busio.Parity* attribute), 89  
 Exception, 199  
 exec()  
     built-in function, 198  
 exit() (*in module sys*), 209

exit\_and\_deep\_sleep\_until\_alarms() (*in module alarm*), 61  
 exp() (*in module math*), 119  
 exp() (*in module ulab.vector*), 161  
 expm1() (*in module math*), 120  
 expm1() (*in module ulab.vector*), 162  
 extend() (*array.array.array* method), 201  
 extended (*canio.Match* attribute), 93  
 extended (*canio.Message* attribute), 93  
 extended (*canio.RemoteTransmissionRequest* attribute), 93  
 ExtType (*class in msgpack*), 126  
 eye() (*in module ulab*), 165

## F

fabs() (*in module math*), 119  
 feed() (*watchdog.WatchDogTimer* method), 170  
 fft() (*in module ulab.fft*), 158  
 file (*audiomp3.MP3Decoder* attribute), 72  
 FileIO (*class in io*), 206  
 fill() (*\_pixelbuf.PixelBuf* method), 54  
 fill() (*displayio.Bitmap* method), 98  
 fill() (*framebuf.FrameBuffer* method), 225  
 fill\_rect() (*framebuf.FrameBuffer* method), 225  
 fill\_region() (*in module bitmaptools*), 79  
 fill\_row() (*displayio.Display* method), 101  
 fill\_row() (*framebufferio.FramebufferDisplay* method), 111  
 filter()  
     built-in function, 198  
 fix (*gnss.GNSS* attribute), 115  
 FIX\_2D (*gnss.PositionFix* attribute), 116  
 FIX\_3D (*gnss.PositionFix* attribute), 116  
 flash() (*in module dualbank*), 109  
 flatten() (*ulab.array* method), 163  
 flip() (*in module ulab.numerical*), 160  
 flip\_x (*displayio.TileGrid* attribute), 109  
 flip\_y (*displayio.TileGrid* attribute), 109  
 float (*built-in class*), 198  
 float (*in module ulab*), 164  
 floor() (*in module math*), 120  
 floor() (*in module ulab.vector*), 162  
 flush() (*\_eve.\_EVE* method), 45  
 flush() (*btree.btree* method), 223  
 flush() (*usb\_cdc.Serial* method), 167  
 fmin() (*in module ulab.approx*), 156  
 fmod() (*in module math*), 120  
 FONT (*in module terminalio*), 153  
 fontio  
     module, 110  
 FourWire (*class in displayio*), 103  
 frame() (*\_stage.Layer* method), 56  
 framebuf  
     module, 224

`framebuf.GS2_HMSB` (in module `framebuf`), 226  
`framebuf.GS4_HMSB` (in module `framebuf`), 226  
`framebuf.GS8` (in module `framebuf`), 226  
`framebuf.MONO_HLSB` (in module `framebuf`), 226  
`framebuf.MONO_HMSB` (in module `framebuf`), 226  
`framebuf.MONO_VLSB` (in module `framebuf`), 226  
`framebuf.RGB565` (in module `framebuf`), 226  
`FrameBuffer` (class in `framebuf`), 225  
`framebuffer` (`framebufferio.FramebufferDisplay` attribute), 111  
`FrameBuffer` (in module `_typing`), 57  
`FramebufferDisplay` (class in `framebufferio`), 110  
`framebufferio`  
    module, 110  
`frequency` (`busio.SPI` attribute), 86  
`frequency` (`microcontroller.Processor` attribute), 124  
`frequency` (`pwmio.PWMOut` attribute), 135  
`frequency` (`rp2pio.StateMachine` attribute), 139  
`frequency` (`samd.Clock` attribute), 142  
`frequency()` (`sdioio.SDCard` property), 144  
`FrequencyIn` (class in `frequencyio`), 112  
`frequencyio`  
    module, 112  
`frexp()` (in module `math`), 120  
`from_bytes()` (int class method), 198  
`frombuffer()` (in module `ulab`), 165  
`frozenset` (built-in class), 198  
`full()` (in module `ulab`), 165

## G

`gamepad`  
    module, 113  
`GamePad` (class in `gamepad`), 113  
`gamepadshift`  
    module, 114  
`GamePadShift` (class in `gamepadshift`), 114  
`gamma()` (in module `math`), 121  
`gamma()` (in module `ulab.vector`), 162  
`gc`  
    module, 203  
`get()` (`btree.btree` method), 223  
`get_bounding_box()` (`fontio.BuiltinFont` method), 110  
`get_glyph()` (`fontio.BuiltinFont` method), 110  
`get_pressed()` (`gamepad.GamePad` method), 114  
`get_pressed()` (`gamepadshift.GamePadShift` method), 114  
`get_printoptions()` (in module `ulab`), 165  
`get_secondary_terminal()` (in module `multiterminal`), 126  
`getaddrinfo()` (in module `socket`), 146  
`getaddrinfo()` (in module `socketpool`), 148  
`getaddrinfo()` (in module `usocket`), 216  
`getattr()`

    built-in function, 198  
`getcwd()` (in module `os`), 128  
`getmount()` (in module `storage`), 150  
`getrandbits()` (in module `random`), 136  
`getvalue()` (`io.BytesIO` method), 206  
`globals()`  
    built-in function, 198  
`GLONASS` (`gnss.SatelliteSystem` attribute), 116  
`Glyph` (class in `fontio`), 110  
`gnss`  
    module, 115  
`GNSS` (class in `gnss`), 115  
`GPS` (`gnss.SatelliteSystem` attribute), 116  
`Group` (class in `displayio`), 104  
`group()` (`re.match` method), 208

## H

`hasattr()`  
    built-in function, 198  
`hash()`  
    built-in function, 198  
`hashlib`  
    module, 204  
`hashlib.md5` (class in `hashlib`), 205  
`hashlib.sh1` (class in `hashlib`), 205  
`hashlib.sha256` (class in `hashlib`), 205  
`heap_lock()` (in module `micropython`), 228  
`heap_unlock()` (in module `micropython`), 228  
`heapify()` (in module `uheapq`), 200  
`heappop()` (in module `uheapq`), 200  
`heappush()` (in module `uheapq`), 200  
`height` (`displayio.Bitmap` attribute), 97  
`height` (`displayio.Display` attribute), 101  
`height` (`displayio.EPaperDisplay` attribute), 103  
`height` (`displayio.OnDiskBitmap` attribute), 106  
`height` (`framebufferio.FramebufferDisplay` attribute), 111  
`height` (`rgbmatrix.RGBMatrix` attribute), 137  
`help()`  
    built-in function, 173  
`hex()`  
    built-in function, 198  
`hexdigest()` (`hashlib.hash` method), 205  
`hexlify()` (in module `binascii`), 201  
`hidden` (`displayio.Group` attribute), 104  
`hidden` (`displayio.TileGrid` attribute), 108  
`hline()` (`framebuf.FrameBuffer` method), 225  
`hostname` (`wifi.Radio` attribute), 171

## I

`I2C` (class in `bitbangio`), 75  
`I2C` (class in `busio`), 82  
`I2C()` (in module `board`), 82  
`I2CDevice` (class in `adafruit_bus_device`), 57



- I2CDisplay (class in *displayio*), 105
  - i2cperipheral
    - module, 116
  - I2CPeripheral (class in *i2cperipheral*), 117
  - I2CPeripheralRequest (class in *i2cperipheral*), 118
  - I2SOut (class in *audiobusio*), 64
  - id (*canio.Match* attribute), 93
  - id (*canio.Message* attribute), 93
  - id (*canio.RemoteTransmissionRequest* attribute), 93
  - id()
    - built-in function, 198
  - ifconfig() (*wiznet.WIZNET5K* method), 173
  - ifft() (in module *ulab.fft*), 158
  - ignore() (*memorymonitor.AllocationAlarm* method), 121
  - ilistdir() (*storage.VfsFat* method), 151
  - ImageFormat (class in *camera*), 89
  - implementation (in module *sys*), 209
  - ImportError, 199
  - in\_waiting (*\_bleio.CharacteristicBuffer* attribute), 40
  - in\_waiting (*busio.UART* attribute), 88
  - in\_waiting (*rp2pio.StateMachine* attribute), 140
  - in\_waiting (*usb\_cdc.Serial* attribute), 166
  - in\_waiting() (*canio.Listener* method), 92
  - INCL (in module *btree*), 224
  - incoming\_packet\_length (*\_bleio.PacketBuffer* attribute), 43
  - IncrementalEncoder (class in *rotaryio*), 137
  - index() (*displayio.Group* method), 105
  - IndexError, 199
  - INDICATE (*\_bleio.Characteristic* attribute), 39
  - inet\_ntop() (in module *usocket*), 216
  - inet\_pton() (in module *usocket*), 216
  - info() (in module *uheap*), 155
  - INPUT (*digitalio.Direction* attribute), 96
  - input()
    - built-in function, 198
  - insert() (*displayio.Group* method), 105
  - int (built-in class), 198
  - int16 (in module *ulab*), 164
  - int8 (in module *ulab*), 164
  - interp() (in module *ulab.approx*), 156
  - inv() (in module *ulab.linalg*), 159
  - INVALID (*gnss.PositionFix* attribute), 116
  - io
    - module, 205
  - ip\_address() (in module *ipaddress*), 119
  - ipaddress
    - module, 119
  - ipoll() (*uselect.poll* method), 214
  - IPPROTO\_SEC (in module *usocket*), 217
  - IPPROTO\_TCP (in module *usocket*), 217
  - IPPROTO\_TCP (*socket.socket* attribute), 145
  - IPPROTO\_UDP (in module *usocket*), 217
  - ipv4\_address (*wifi.Radio* attribute), 171
  - ipv4\_dns (*wifi.Radio* attribute), 171
  - ipv4\_gateway (*wifi.Radio* attribute), 171
  - ipv4\_subnet (*wifi.Radio* attribute), 171
  - IPv4Address (class in *ipaddress*), 119
  - is\_read (*i2cperipheral.I2CPeripheralRequest* attribute), 118
  - is\_restart (*i2cperipheral.I2CPeripheralRequest* attribute), 118
  - is\_transparent() (*displayio.Palette* method), 107
  - isfinite() (in module *math*), 120
  - isinf() (in module *math*), 120
  - isinstance()
    - built-in function, 198
  - isnan() (in module *math*), 120
  - issubclass()
    - built-in function, 198
  - items() (*btree.btree* method), 224
  - itemsizes (*ulab.array* attribute), 163
  - iter()
    - built-in function, 198
- ## J
- JPG (*camera.ImageFormat* attribute), 89
  - json
    - module, 207
  - Jump() (*\_eve.\_EVE* method), 50
- ## K
- kbd\_intr() (in module *micropython*), 228
  - KeyboardInterrupt, 199
  - KeyError, 199
  - keys() (*btree.btree* method), 224
- ## L
- label (*storage.VfsFat* attribute), 151
  - last\_received\_report (*usb\_hid.Device* attribute), 167
  - latitude (*gnss.GNSS* attribute), 115
  - Layer (class in *\_stage*), 55
  - ldexp() (in module *math*), 120
  - len()
    - built-in function, 198
  - length (*canio.RemoteTransmissionRequest* attribute), 93
  - LESC\_ENCRYPT\_WITH\_MITM (*\_bleio.Attribute* attribute), 38
  - level (*audiomixer.MixerVoice* attribute), 71
  - lgamma() (in module *math*), 121
  - lgamma() (in module *ulab.vector*), 162
  - light\_sleep\_until\_alarms() (in module *alarm*), 61
  - line() (*framebuf.FrameBuffer* method), 225

`LineWidth()` (*\_eve.EVE method*), 52  
`linspace()` (*in module ulab*), 165  
`list` (*built-in class*), 198  
`listdir()` (*in module os*), 128  
`listen()` (*canio.CAN method*), 91  
`listen()` (*socket.socket method*), 145  
`listen()` (*socketpool.Socket method*), 147  
`listen()` (*ssl.SSLSocket method*), 149  
`listen()` (*usocket.socket method*), 217  
`Listener` (*class in canio*), 92  
`LITTLE_ENDIAN` (*in module ctypes*), 212  
`load()` (*in module json*), 207  
`loads()` (*in module json*), 207  
`locals()`  
    *built-in function*, 198  
`localtime()` (*in module time*), 154  
`log()` (*in module ulab.vector*), 162  
`log10()` (*in module math*), 120  
`log10()` (*in module ulab.vector*), 162  
`log2()` (*in module math*), 120  
`log2()` (*in module ulab.vector*), 162  
`logspace()` (*in module ulab*), 165  
`longitude` (*gnss.GNSS attribute*), 115  
`loopback` (*canio.CAN attribute*), 91

## M

`mac_address` (*wifi.Radio attribute*), 171  
`machine` (*os.\_Uname attribute*), 128  
`Macro()` (*\_eve.EVE method*), 50  
`make_opaque()` (*displayio.ColorConverter method*), 98  
`make_opaque()` (*displayio.Palette method*), 107  
`make_transparent()` (*displayio.ColorConverter method*), 98  
`make_transparent()` (*displayio.Palette method*), 107  
`makefile()` (*usocket.socket method*), 219  
`map()`  
    *built-in function*, 198  
`mask` (*canio.Match attribute*), 93  
`Match` (*class in canio*), 92  
`match()` (*in module re*), 208  
`match()` (*re.regex method*), 208  
`matches()` (*\_bleio.ScanEntry method*), 44  
`math`  
    *module*, 119  
`max()`  
    *built-in function*, 198  
`max()` (*in module ulab.numerical*), 160  
`max_stack_usage()` (*in module ustack*), 169  
`maximum()` (*in module ulab.compare*), 157  
`maxlen` (*pulseio.PulseIn attribute*), 132  
`maxsize` (*in module sys*), 209  
`mean()` (*in module ulab.numerical*), 160

`median()` (*in module ulab.numerical*), 160  
`mem_alloc()` (*in module gc*), 203  
`mem_free()` (*in module gc*), 204  
`mem_info()` (*in module micropython*), 227  
`MemoryError`, 199  
`memorymonitor`  
    *module*, 121  
`memoryview` (*built-in class*), 198  
`Message` (*class in canio*), 93  
`microcontroller`  
    *module*, 123  
`micropython`  
    *module*, 227  
`min()`  
    *built-in function*, 198  
`min()` (*in module ulab.numerical*), 160  
`minimum()` (*in module ulab.compare*), 157  
`Mixer` (*class in audiomixer*), 70  
`MixerVoice` (*class in audiomixer*), 71  
`mkdir()` (*in module os*), 128  
`mkdir()` (*storage.VfsFat method*), 151  
`mkfs()` (*storage.VfsFat method*), 151  
`mktime()` (*in module time*), 154  
`mode` (*watchdog.WatchDogTimer attribute*), 170  
`modf()` (*in module math*), 120  
`modify()` (*uselect.poll method*), 214  
`module`  
    *\_bleio*, 35  
    *\_eve*, 45  
    *\_pew*, 53  
    *\_pixelbuf*, 54  
    *\_stage*, 55  
    *\_typing*, 56  
    *adafruit\_bus\_device*, 57  
    *aesio*, 59  
    *alarm*, 59  
    *alarm.pin*, 60  
    *alarm.time*, 60  
    *alarm.touch*, 61  
    *analogio*, 62  
    *array*, 201  
    *audiobusio*, 64  
    *audiocore*, 66  
    *audioio*, 68  
    *audiomixer*, 70  
    *audiomp3*, 72  
    *audiopwmio*, 73  
    *binascii*, 201  
    *bitbangio*, 74  
    *bitmaptools*, 79  
    *bitops*, 81  
    *board*, 82  
    *btrees*, 222  
    *busio*, 82

camera, 89  
 canio, 90  
 collections, 202  
 countio, 94  
 digitalio, 95  
 displayio, 97  
 dualbank, 109  
 errno, 203  
 fontio, 110  
 framebuffer, 224  
 framebufferio, 110  
 frequencyio, 112  
 gamepad, 113  
 gamepadshift, 114  
 gc, 203  
 gnss, 115  
 hashlib, 204  
 i2cperipheral, 116  
 io, 205  
 ipaddress, 119  
 json, 207  
 math, 119  
 memorymonitor, 121  
 microcontroller, 123  
 micropython, 227  
 msgpack, 125  
 multiterminal, 126  
 neopixel\_write, 127  
 network, 127  
 nvm, 127  
 os, 128  
 ps2io, 129  
 pulseio, 131  
 pwmio, 134  
 random, 135  
 re, 207  
 rgbmatrix, 136  
 rotaryio, 137  
 rp2pio, 138  
 rtc, 141  
 samd, 142  
 sdcardio, 142  
 sdioio, 143  
 sharpdisplay, 145  
 socket, 145  
 socketpool, 147  
 ssl, 149  
 storage, 150  
 struct, 151  
 supervisor, 152  
 sys, 208  
 terminalio, 153  
 time, 153  
 touchio, 154

ctypes, 210  
 uheap, 155  
 uheapq, 200  
 ulab, 156  
 ulab.approx, 156  
 ulab.compare, 157  
 ulab.fft, 158  
 ulab.filter, 158  
 ulab.linalg, 159  
 ulab.numerical, 160  
 ulab.poly, 161  
 ulab.user, 161  
 ulab.vector, 161  
 usb\_cdc, 166  
 usb\_hid, 167  
 usb\_midi, 168  
 uselect, 213  
 usocket, 215  
 ussl, 220  
 ustack, 169  
 uzlib, 221  
 vectorio, 169  
 watchdog, 170  
 wifi, 171  
 wiznet, 172  
 modules (*in module sys*), 210  
 monotonic() (*in module time*), 153  
 monotonic\_ns() (*in module time*), 154  
 monotonic\_time (*alarm.time.TimeAlarm attribute*),  
     60  
 mount() (*in module storage*), 150  
 mount() (*storage.VfsFat method*), 151  
 move() (*\_stage.Layer method*), 56  
 move() (*\_stage.Text method*), 56  
 MP3Decoder (*class in audiomp3*), 72  
 msgpack  
     module, 125  
 multiterminal  
     module, 126

## N

name (*\_bleio.Adapter attribute*), 36  
 namedtuple() (*in module collections*), 202  
 NameError, 199  
 NATIVE (*in module ctypes*), 212  
 ndinfo() (*in module ulab*), 165  
 neopixel\_write  
     module, 127  
 neopixel\_write() (*in module neopixel\_write*), 127  
 network  
     module, 127  
 Network (*class in wifi*), 171  
 newton() (*in module ulab.approx*), 157  
 next()

- built-in function, 198
- NO\_ACCESS (*\_bleio.Attribute* attribute), 38
- nodename (*os.\_Uname* attribute), 128
- Nop() (*\_eve.\_EVE* method), 50
- norm() (in module *ulab.linalg*), 160
- NORMAL (*microcontroller.RunMode* attribute), 124
- not\_equal() (in module *ulab.compare*), 157
- NOTIFY (*\_bleio.Characteristic* attribute), 39
- NotImplementedError, 199
- nvm
  - module, 127
- nvm (in module *microcontroller*), 123

## O

- object (built-in class), 198
- oct()
  - built-in function, 198
- ODD (*busio.Parity* attribute), 89
- on\_next\_reset() (in module *microcontroller*), 123
- OnDiskBitmap (class in *displayio*), 106
- ones() (in module *ulab*), 165
- OneWire (class in *bitbangio*), 76
- OneWire (class in *busio*), 84
- OneWire.OneWire (class in *busio*), 84
- OPEN (*\_bleio.Attribute* attribute), 38
- open()
  - built-in function, 199
- open() (in module *btree*), 223
- open() (in module *io*), 206
- open() (*storage.VfsFat* method), 151
- OPEN\_DRAIN (*digitalio.DriveMode* attribute), 95
- opt\_level() (in module *micropython*), 227
- ord()
  - built-in function, 199
- OrderedDict() (in module *collections*), 202
- os
  - module, 128
- OSError, 199
- out\_waiting (*usb\_cdc.Serial* attribute), 166
- outgoing\_packet\_length (*\_bleio.PacketBuffer* attribute), 43
- OUTPUT (*digitalio.Direction* attribute), 96

## P

- pack() (in module *msgpack*), 126
- pack() (in module *struct*), 151
- pack\_into() (*\_bleio.UUID* method), 45
- pack\_into() (in module *struct*), 151
- packed (*ipaddress.IPv4Address* attribute), 119
- packet\_size (*\_bleio.PacketBuffer* attribute), 43
- PacketBuffer (class in *\_bleio*), 43
- pair() (*\_bleio.Connection* method), 41
- paired (*\_bleio.Connection* attribute), 41
- Palette (class in *displayio*), 106

- PaletteSource() (*\_eve.\_EVE* method), 50
- ParallelBus (class in *displayio*), 107
- parent (*samd.Clock* attribute), 142
- Parity (class in *busio*), 89
- path (in module *sys*), 210
- pause() (*audiobusio.I2SOut* method), 65
- pause() (*audioio.AudioOut* method), 70
- pause() (*audiopwmio.PWMAudioOut* method), 74
- pause() (*frequencyio.FrequencyIn* method), 113
- pause() (*pulseio.PulseIn* method), 132
- paused (*audiobusio.I2SOut* attribute), 65
- paused (*audioio.AudioOut* attribute), 69
- paused (*audiopwmio.PWMAudioOut* attribute), 74
- paused (*pulseio.PulseIn* attribute), 132
- PDMIn (class in *audiobusio*), 65
- PewPew (class in *\_pew*), 53
- pi (in module *math*), 119
- pin (*alarm.pin.PinAlarm* attribute), 60
- pin (*alarm.touch.TouchAlarm* attribute), 61
- Pin (class in *microcontroller*), 123
- PinAlarm (class in *alarm.pin*), 60
- ping() (*wifi.Radio* method), 172
- pins\_are\_sequential() (in module *rp2pio*), 138
- pixel() (*framebuf.FrameBuffer* method), 225
- pixel\_shader (*displayio.TileGrid* attribute), 109
- pixel\_shader (*vectorio.VectorShape* attribute), 169
- PixelBuf (class in *\_pixelbuf*), 54
- platform (in module *sys*), 210
- play() (*audiobusio.I2SOut* method), 65
- play() (*audioio.AudioOut* method), 69
- play() (*audiomixer.Mixer* method), 71
- play() (*audiomixer.MixerVoice* method), 71
- play() (*audiopwmio.PWMAudioOut* method), 74
- playing (*audiobusio.I2SOut* attribute), 65
- playing (*audioio.AudioOut* attribute), 69
- playing (*audiomixer.Mixer* attribute), 70
- playing (*audiomixer.MixerVoice* attribute), 71
- playing (*audiopwmio.PWMAudioOut* attribute), 74
- points (*vectorio.Polygon* attribute), 169
- PointSize() (*\_eve.\_EVE* method), 52
- poll() (in module *uselect*), 214
- poll() (*uselect.poll* method), 214
- polyfit() (in module *ulab.poly*), 161
- Polygon (class in *vectorio*), 169
- polyval() (in module *ulab.poly*), 161
- pop() (*displayio.Group* method), 105
- popleft() (*ps2io.Ps2* method), 130
- popleft() (*pulseio.PulseIn* method), 132
- PortIn (class in *usb\_midi*), 168
- PortOut (class in *usb\_midi*), 168
- ports (in module *usb\_midi*), 168
- position (*rotaryio.IncrementalEncoder* attribute), 137
- PositionFix (class in *gnss*), 115
- pow()

built-in function, 199  
 pow() (in module math), 120  
 POWER\_ON (microcontroller.ResetReason attribute), 124  
 print()  
     built-in function, 199  
 print\_exception() (in module sys), 209  
 Processor (class in microcontroller), 123  
 properties (\_bleio.Characteristic attribute), 39  
 property()  
     built-in function, 199  
 Ps2 (class in ps2io), 129  
 ps2io  
     module, 129  
 PUBLIC (\_bleio.Address attribute), 38  
 Pull (class in digitalio), 97  
 pull (digitalio.DigitalInOut attribute), 96  
 PulseIn (class in pulseio), 131  
 pulseio  
     module, 131  
 PulseOut (class in pulseio), 133  
 PUSH\_PULL (digitalio.DriveMode attribute), 95  
 PWMAudioOut (class in audiopwmio), 73  
 pwmio  
     module, 134  
 PWMOut (class in pwmio), 134

## Q

qstr\_info() (in module micropython), 227  
 QZSS\_L1CA (gnss.SatelliteSystem attribute), 116  
 QZSS\_L1S (gnss.SatelliteSystem attribute), 116

## R

radians() (in module math), 120  
 radians() (in module ulab.vector), 162  
 Radio (class in wifi), 171  
 radio (in module wifi), 171  
 radius (vectorio.Circle attribute), 169  
 RAISE (watchdog.WatchDogMode attribute), 170  
 randint() (in module random), 136  
 random  
     module, 135  
 random() (in module random), 136  
 RANDOM\_PRIVATE\_NON\_RESOLVABLE  
     (\_bleio.Address attribute), 38  
 RANDOM\_PRIVATE\_RESOLVABLE (\_bleio.Address attribute), 38  
 RANDOM\_STATIC (\_bleio.Address attribute), 38  
 randrange() (in module random), 136  
 range()  
     built-in function, 199  
 raw\_value (touchio.TouchIn attribute), 155  
 RawSample (class in audiocore), 66  
 re  
     module, 207

READ (\_bleio.Characteristic attribute), 39  
 read() (\_bleio.CharacteristicBuffer method), 40  
 read() (busio.UART method), 88  
 read() (i2cperipheral.I2CPeripheralRequest method), 118  
 read() (usb\_cdc.Serial method), 166  
 read() (usb\_midi.PortIn method), 168  
 read() (usocket.socket method), 219  
 read\_bit() (bitbangio.OneWire method), 77  
 read\_bit() (busio.OneWire method), 85  
 ReadableBuffer (in module \_typing), 56  
 readblocks() (sdcardio.SDCard method), 143  
 readblocks() (sdioio.SDCard method), 144  
 readfrom\_into() (bitbangio.I2C method), 75  
 readfrom\_into() (busio.I2C method), 83  
 readinto() (\_bleio.CharacteristicBuffer method), 40  
 readinto() (\_bleio.PacketBuffer method), 43  
 readinto() (adafruit\_bus\_device.I2CDevice method), 57  
 readinto() (bitbangio.SPI method), 78  
 readinto() (busio.SPI method), 86  
 readinto() (busio.UART method), 88  
 readinto() (in module bitmaptools), 81  
 readinto() (rp2pio.StateMachine method), 140  
 readinto() (usb\_cdc.Serial method), 166  
 readinto() (usb\_midi.PortIn method), 168  
 readinto() (usocket.socket method), 219  
 readline() (\_bleio.CharacteristicBuffer method), 40  
 readline() (busio.UART method), 88  
 readline() (usb\_cdc.Serial method), 166  
 readline() (usocket.socket method), 219  
 readlines() (usb\_cdc.Serial method), 167  
 receive() (canio.Listener method), 92  
 receive\_error\_count (canio.CAN attribute), 91  
 record() (audiobusio.PDMIn method), 66  
 rect() (framebuf.FrameBuffer method), 225  
 Rectangle (class in vectorio), 169  
 recv() (socket.socket method), 146  
 recv() (usocket.socket method), 218  
 recv\_into() (socket.socket method), 146  
 recv\_into() (socketpool.Socket method), 147  
 recv\_into() (ssl.SSLSocket method), 149  
 recvfrom() (socket.socket method), 146  
 recvfrom() (usocket.socket method), 218  
 recvfrom\_into() (socketpool.Socket method), 147  
 reference\_voltage (analogio.AnalogIn attribute), 63  
 refresh() (displayio.Display method), 101  
 refresh() (displayio.EPaperDisplay method), 103  
 refresh() (framebufferio.FramebufferDisplay method), 111  
 refresh() (rgbmatrix.RGBMatrix method), 137  
 register() (\_eve.\_EVE method), 45  
 register() (uselect.poll method), 214



`release` (*os.\_Uname attribute*), 128  
`release_displays()` (in module *displayio*), 97  
`reload()` (in module *supervisor*), 152  
`ReloadException`, 200  
`remote` (*\_bleio.Service attribute*), 44  
`RemoteTransmissionRequest` (class in *canio*), 93  
`remount()` (in module *storage*), 150  
`remove()` (*displayio.Group method*), 105  
`remove()` (in module *os*), 128  
`rename()` (in module *os*), 128  
`render()` (in module *\_stage*), 55  
`REPL_RELOAD` (*supervisor.RunReason attribute*), 152  
`repr()`  
    built-in function, 199  
`request()` (*i2cperipheral.I2CPeripheral method*), 117  
`RESET` (*watchdog.WatchDogMode attribute*), 170  
`reset()` (*bitbangio.OneWire method*), 77  
`reset()` (*busio.OneWire method*), 85  
`reset()` (*countio.Counter method*), 94  
`reset()` (*displayio.FourWire method*), 104  
`reset()` (*displayio.I2CDisplay method*), 106  
`reset()` (*displayio.ParallelBus method*), 107  
`reset()` (in module *microcontroller*), 123  
`reset_input_buffer()`  
    (*\_bleio.CharacteristicBuffer method*), 41  
`reset_input_buffer()` (*busio.UART method*), 88  
`reset_input_buffer()` (*usb\_cdc.Serial method*), 167  
`reset_output_buffer()` (*usb\_cdc.Serial method*), 167  
`RESET_PIN` (*microcontroller.ResetReason attribute*), 124  
`reset_reason` (*microcontroller.Processor attribute*), 124  
`ResetReason` (class in *microcontroller*), 124  
`reshape()` (*ulab.array method*), 163  
`restart()` (*canio.CAN method*), 91  
`restart()` (*rp2pio.StateMachine method*), 140  
`RestoreContext()` (*\_eve.\_EVE method*), 50  
`resume()` (*audiobusio.I2SOut method*), 65  
`resume()` (*audioio.AudioOut method*), 70  
`resume()` (*audiopwmio.PWMAudioOut method*), 74  
`resume()` (*frequencyio.FrequencyIn method*), 113  
`resume()` (*pulseio.PulseIn method*), 132  
`Return()` (*\_eve.\_EVE method*), 50  
`reversed()`  
    built-in function, 199  
`RGB565` (*camera.ImageFormat attribute*), 90  
`rgbmatrix`  
    module, 136  
`RGBMatrix` (class in *rgbmatrix*), 136  
`rmdir()` (in module *os*), 128  
`rmdir()` (*storage.VfsFat method*), 151  
`rms_level` (*audiomp3.MP3Decoder attribute*), 72  
`RoleError`, 35  
`roll()` (in module *ulab.numerical*), 160  
`rotaryio`  
    module, 137  
`rotation` (*displayio.Display attribute*), 101  
`rotation` (*displayio.EPaperDisplay attribute*), 103  
`rotation` (*framebufferio.FramebufferDisplay attribute*), 111  
`rotozoom()` (in module *bitmaptools*), 79  
`round()`  
    built-in function, 199  
`route()` (in module *network*), 127  
`rp2pio`  
    module, 138  
`rsssi` (*\_bleio.ScanEntry attribute*), 44  
`rsssi` (*wifi.Network attribute*), 171  
`rtc`  
    module, 141  
`RTC` (class in *rtc*), 141  
`run()` (*rp2pio.StateMachine method*), 140  
`run_reason` (*supervisor.Runtime attribute*), 153  
`RunMode` (class in *microcontroller*), 124  
`RunReason` (class in *supervisor*), 152  
`Runtime` (class in *supervisor*), 152  
`runtime` (in module *supervisor*), 152  
`RuntimeError`, 200  
`rxstall` (*rp2pio.StateMachine attribute*), 139

## S

`SAFE_MODE` (*microcontroller.RunMode attribute*), 125  
`samd`  
    module, 142  
`sample_rate` (*audiobusio.PDMIn attribute*), 66  
`sample_rate` (*audiocore.RawSample attribute*), 67  
`sample_rate` (*audiocore.WaveFile attribute*), 68  
`sample_rate` (*audiomixer.Mixer attribute*), 71  
`sample_rate` (*audiomp3.MP3Decoder attribute*), 72  
`SatelliteSystem` (class in *gnss*), 116  
`SaveContext()` (*\_eve.\_EVE method*), 51  
`SBAS` (*gnss.SatelliteSystem attribute*), 116  
`scale` (*displayio.Group attribute*), 104  
`scan()` (*bitbangio.I2C method*), 75  
`scan()` (*busio.I2C method*), 83  
`scan_response` (*\_bleio.ScanEntry attribute*), 44  
`ScanEntry` (class in *\_bleio*), 43  
`ScannedNetworks` (class in *wifi*), 172  
`ScanResults` (class in *\_bleio*), 44  
`schedule_secondary_terminal_read()` (in module *multiterminal*), 126  
`ScissorSize()` (*\_eve.\_EVE method*), 51  
`ScissorXY()` (*\_eve.\_EVE method*), 51  
`scroll()` (*framebuf.FrameBuffer method*), 226  
`SDCard` (class in *sdcardio*), 142  
`SDCard` (class in *sdioio*), 143

- sdcardio
  - module, 142
- sdioio
  - module, 143
- search() (in module re), 208
- search() (re.regex method), 208
- secondary (\_bleio.Service attribute), 44
- SecurityError, 35
- seed() (in module random), 135
- select() (in module uselect), 214
- send() (canio.CAN method), 92
- send() (displayio.FourWire method), 104
- send() (displayio.I2CDisplay method), 106
- send() (displayio.ParallelBus method), 107
- send() (pulseio.PulseOut method), 133
- send() (socket.socket method), 146
- send() (socketpool.Socket method), 148
- send() (ssl.SSLSocket method), 149
- send() (usocket.socket method), 218
- send\_report() (usb\_hid.Device method), 168
- sendall() (usocket.socket method), 218
- sendcmd() (ps2io.Ps2 method), 130
- sendto() (socket.socket method), 146
- sendto() (socketpool.Socket method), 148
- sendto() (usocket.socket method), 218
- sep (in module os), 129
- Serial (class in usb\_cdc), 166
- serial\_bytes\_available (supervisor.Runtime attribute), 152
- serial\_connected (supervisor.Runtime attribute), 152
- serials (in module usb\_cdc), 166
- service (\_bleio.Characteristic attribute), 39
- Service (class in \_bleio), 44
- set (built-in class), 199
- set\_adapter() (in module \_bleio), 35
- set\_boundary() (displayio.Shape method), 108
- set\_cccd() (\_bleio.Characteristic method), 40
- set\_next\_stack\_limit() (in module supervisor), 152
- set\_printoptions() (in module ulab), 165
- set\_rgb\_status\_brightness() (in module supervisor), 152
- set\_secondary\_terminal() (in module multiterminal), 126
- set\_time\_source() (in module rtc), 141
- setattr()
  - built-in function, 199
- setblocking() (socket.socket method), 146
- setblocking() (socketpool.Socket method), 148
- setblocking() (ssl.SSLSocket method), 150
- setblocking() (usocket.socket method), 219
- setsockopt() (socket.socket method), 146
- setsockopt() (usocket.socket method), 218
- settimeout() (socket.socket method), 146
- settimeout() (socketpool.Socket method), 148
- settimeout() (ssl.SSLSocket method), 150
- settimeout() (usocket.socket method), 218
- Shape (class in displayio), 108
- shape (ulab.array attribute), 163
- sharpdisplay
  - module, 145
- show() (\_pixelbuf.PixelBuf method), 54
- show() (displayio.Display method), 101
- show() (displayio.EPaperDisplay method), 103
- show() (framebufferio.FramebufferDisplay method), 111
- SIGNED\_NO\_MITM (\_bleio.Attribute attribute), 38
- SIGNED\_WITH\_MITM (\_bleio.Attribute attribute), 38
- silent (canio.CAN attribute), 91
- sin() (in module math), 120
- sin() (in module ulab.vector), 162
- sinh() (in module math), 120
- sinh() (in module ulab.vector), 162
- size (\_bleio.UUID attribute), 45
- size (ulab.array attribute), 163
- sizeof() (in module ctypes), 212
- sleep() (in module time), 153
- sleep\_memory (in module alarm), 61
- SleepMemory (class in alarm), 61
- slice (built-in class), 199
- SOCK\_DGRAM (in module usocket), 217
- SOCK\_DGRAM (socket.socket attribute), 145
- SOCK\_DGRAM (socketpool.SocketPool attribute), 148
- SOCK\_RAW (socket.socket attribute), 145
- SOCK\_RAW (socketpool.SocketPool attribute), 148
- SOCK\_STREAM (in module usocket), 217
- SOCK\_STREAM (socket.socket attribute), 145
- SOCK\_STREAM (socketpool.SocketPool attribute), 148
- socket
  - module, 145
- socket (class in socket), 145
- Socket (class in socketpool), 147
- socket() (in module usocket), 216
- socket() (socketpool.SocketPool method), 148
- socketpool
  - module, 147
- SocketPool (class in socketpool), 148
- SOFTWARE (microcontroller.ResetReason attribute), 124
- sort() (displayio.Group method), 105
- sort() (in module ulab.numerical), 160
- sort() (ulab.array method), 163
- sorted()
  - built-in function, 199
- sosfilt() (in module ulab.filter), 158
- spectrogram() (in module ulab.fft), 158
- SPI (class in bitbangio), 77
- SPI (class in busio), 85

`SPI()` (in module `board`), 82  
`SPIDevice` (class in `adafruit_bus_device`), 58  
`split()` (re.regex method), 208  
`sqrt()` (in module `math`), 120  
`sqrt()` (in module `ulab.vector`), 162  
`ssid` (`wifi.Network` attribute), 171  
`ssl`  
    module, 149  
`ssl.SSLError` (in module `ssl`), 220  
`SSLContext` (class in `ssl`), 149  
`SSLSocket` (class in `ssl`), 149  
`stack_size()` (in module `ustack`), 169  
`stack_usage()` (in module `ustack`), 169  
`stack_use()` (in module `micropython`), 227  
`start_advertising()` (`_bleio.Adapter` method), 36  
`start_scan()` (`_bleio.Adapter` method), 36  
`start_scanning_networks()` (`wifi.Radio` method), 172  
`STARTUP` (`supervisor.RunReason` attribute), 152  
`stat()` (in module `os`), 128  
`stat()` (`storage.VfsFat` method), 151  
`state` (`canio.CAN` attribute), 91  
`StateMachine` (class in `rp2pio`), 138  
`staticmethod()`  
    built-in function, 199  
`statvfs()` (in module `os`), 129  
`statvfs()` (`storage.VfsFat` method), 151  
`std()` (in module `ulab.numerical`), 160  
`stderr` (in module `sys`), 210  
`stdin` (in module `sys`), 210  
`stdout` (in module `sys`), 210  
`StencilFunc()` (`_eve._EVE` method), 51  
`StencilMask()` (`_eve._EVE` method), 51  
`StencilOp()` (`_eve._EVE` method), 51  
`stop()` (`audiobusio.I2SOut` method), 65  
`stop()` (`audioio.AudioOut` method), 70  
`stop()` (`audiomixer.MixerVoice` method), 71  
`stop()` (`audiopwmio.PWMAudioOut` method), 74  
`stop()` (`rp2pio.StateMachine` method), 140  
`stop_advertising()` (`_bleio.Adapter` method), 36  
`stop_scan()` (`_bleio.Adapter` method), 37  
`stop_scanning_networks()` (`wifi.Radio` method), 172  
`stop_voice()` (`audiomixer.Mixer` method), 71  
`StopIteration`, 200  
`storage`  
    module, 150  
`str` (built-in class), 199  
`strides` (`ulab.array` attribute), 163  
`StringIO` (class in `io`), 206  
`struct`  
    module, 151  
`struct` (class in `uctypes`), 212  
`struct_time` (class in `time`), 153  
`sum()`  
    built-in function, 199  
`sum()` (in module `ulab.numerical`), 160  
`super()`  
    built-in function, 199  
`supervisor`  
    module, 152  
`SUPERVISOR_RELOAD` (`supervisor.RunReason` attribute), 152  
`switch()` (in module `dualbank`), 109  
`switch_to_input()` (`digitalio.DigitalInOut` method), 96  
`switch_to_output()` (`digitalio.DigitalInOut` method), 96  
`sync()` (in module `os`), 129  
`SyntaxError`, 200  
`sys`  
    module, 208  
`sysname` (`os._Uname` attribute), 128  
`SystemExit`, 200

## T

`Tag()` (`_eve._EVE` method), 52  
`TagMask()` (`_eve._EVE` method), 52  
`take_picture()` (`camera.Camera` method), 89  
`tan()` (in module `math`), 120  
`tan()` (in module `ulab.vector`), 162  
`tanh()` (in module `math`), 120  
`tanh()` (in module `ulab.vector`), 162  
`temperature` (`microcontroller.Processor` attribute), 124  
`Terminal` (class in `terminalio`), 153  
`terminalio`  
    module, 153  
`Text` (class in `_stage`), 56  
`text()` (`framebuf.FrameBuffer` method), 226  
`TextIOWrapper` (class in `io`), 206  
`threshold` (`touchio.TouchIn` attribute), 155  
`threshold()` (in module `gc`), 204  
`TileGrid` (class in `displayio`), 108  
`time`  
    module, 153  
`time()` (in module `time`), 154  
`time_to_refresh` (`displayio.EPaperDisplay` attribute), 103  
`TimeAlarm` (class in `alarm.time`), 60  
`timeout` (`busio.UART` attribute), 88  
`timeout` (`canio.Listener` attribute), 92  
`timeout` (`usb_cdc.Serial` attribute), 166  
`timeout` (`watchdog.WatchDogTimer` attribute), 170  
`timestamp` (`gnss.GNSS` attribute), 115  
`to_bytes()` (`int` method), 198  
`tobytes()` (`ulab.array` method), 163  
`TouchAlarm` (class in `alarm.touch`), 61



[TouchIn \(class in touchio\)](#), 154  
[touchio](#)  
     module, 154  
[trace\(\)](#) (in module [ulab.linalg](#)), 160  
[transmit\\_error\\_count](#) ([canio.CAN](#) attribute), 91  
[transpose\(\)](#) ([ulab.array](#) method), 163  
[transpose\\_xy](#) ([displayio.TileGrid](#) attribute), 109  
[trapz\(\)](#) (in module [ulab.approx](#)), 157  
[trunc\(\)](#) (in module [math](#)), 120  
[try\\_lock\(\)](#) ([bitbangio.I2C](#) method), 75  
[try\\_lock\(\)](#) ([bitbangio.SPI](#) method), 78  
[try\\_lock\(\)](#) ([busio.I2C](#) method), 83  
[try\\_lock\(\)](#) ([busio.SPI](#) method), 86  
[tuple](#) (built-in class), 199  
[type](#) ([\\_bleio.Address](#) attribute), 38  
[type\(\)](#)  
     built-in function, 199  
[TypeError](#), 200

## U

[UART \(class in busio\)](#), 87  
[UART\(\)](#) (in module [board](#)), 82  
[uctypes](#)  
     module, 210  
[uheap](#)  
     module, 155  
[uheapq](#)  
     module, 200  
[uid](#) ([microcontroller.Processor](#) attribute), 124  
[uint16](#) (in module [ulab](#)), 165  
[uint8](#) (in module [ulab](#)), 165  
[ulab](#)  
     module, 156  
[ulab.approx](#)  
     module, 156  
[ulab.compare](#)  
     module, 157  
[ulab.fft](#)  
     module, 158  
[ulab.filter](#)  
     module, 158  
[ulab.linalg](#)  
     module, 159  
[ulab.numerical](#)  
     module, 160  
[ulab.poly](#)  
     module, 161  
[ulab.user](#)  
     module, 161  
[ulab.vector](#)  
     module, 161  
[umount\(\)](#) (in module [storage](#)), 150  
[umount\(\)](#) ([storage.VfsFat](#) method), 151  
[uname\(\)](#) (in module [os](#)), 128

[unhexlify\(\)](#) (in module [binascii](#)), 201  
[uniform\(\)](#) (in module [random](#)), 136  
[UNKNOWN](#) ([microcontroller.ResetReason](#) attribute), 124  
[unlock\(\)](#) ([bitbangio.I2C](#) method), 75  
[unlock\(\)](#) ([bitbangio.SPI](#) method), 78  
[unlock\(\)](#) ([busio.I2C](#) method), 83  
[unlock\(\)](#) ([busio.SPI](#) method), 86  
[unpack\(\)](#) (in module [msgpack](#)), 126  
[unpack\(\)](#) (in module [struct](#)), 151  
[unpack\\_from\(\)](#) (in module [struct](#)), 151  
[unregister\(\)](#) ([uselect.poll](#) method), 214  
[UP](#) ([digitalio.Pull](#) attribute), 97  
[update\(\)](#) ([gnss.GNSS](#) method), 115  
[update\(\)](#) ([hashlib.hash](#) method), 205  
[urandom\(\)](#) (in module [os](#)), 129  
[usage](#) ([usb\\_hid.Device](#) attribute), 167  
[usage\\_page](#) ([usb\\_hid.Device](#) attribute), 167  
[usb\\_cdc](#)  
     module, 166  
[usb\\_connected](#) ([supervisor.Runtime](#) attribute), 152  
[usb\\_hid](#)  
     module, 167  
[usb\\_midi](#)  
     module, 168  
[uselect](#)  
     module, 213  
[usocket](#)  
     module, 215  
[usocket.error](#), 219  
[ussl](#)  
     module, 220  
[ussl.CERT\\_NONE](#) (in module [ussl](#)), 220  
[ussl.CERT\\_OPTIONAL](#) (in module [ussl](#)), 220  
[ussl.CERT\\_REQUIRED](#) (in module [ussl](#)), 220  
[ussl.wrap\\_socket\(\)](#) (in module [ussl](#)), 220  
[ustack](#)  
     module, 169  
[uuid](#) ([\\_bleio.Characteristic](#) attribute), 39  
[uuid](#) ([\\_bleio.Descriptor](#) attribute), 42  
[uuid](#) ([\\_bleio.Service](#) attribute), 44  
[UUID](#) (class in [\\_bleio](#)), 45  
[uuid128](#) ([\\_bleio.UUID](#) attribute), 45  
[uuid16](#) ([\\_bleio.UUID](#) attribute), 45  
[uzlib](#)  
     module, 221

## V

[value](#) ([\\_bleio.Characteristic](#) attribute), 39  
[value](#) ([\\_bleio.Descriptor](#) attribute), 42  
[value](#) ([alarm.pin.PinAlarm](#) attribute), 60  
[value](#) ([analogio.AnalogIn](#) attribute), 63  
[value](#) ([analogio.AnalogOut](#) attribute), 63  
[value](#) ([digitalio.DigitalInOut](#) attribute), 95  
[value](#) ([touchio.TouchIn](#) attribute), 155

`ValueError`, 200  
`values()` (*btree.btree* method), 224  
`vectorio`  
    module, 169  
`vectorize()` (in module *ulab.vector*), 162  
`VectorShape` (class in *vectorio*), 169  
`version` (in module *sys*), 210  
`version` (*ipaddress.IPv4Address* attribute), 119  
`version` (*os.\_Unix* attribute), 128  
`version_info` (in module *sys*), 210  
`Vertex2f()` (*\_eve.\_EVE* method), 52  
`Vertex2ii()` (*\_eve.\_EVE* method), 52  
`VertexFormat()` (*\_eve.\_EVE* method), 53  
`VertexTranslateX()` (*\_eve.\_EVE* method), 52  
`VertexTranslateY()` (*\_eve.\_EVE* method), 53  
`VfsFat` (class in *storage*), 151  
`vline()` (*framebuf.FrameBuffer* method), 225  
`voice` (*audiomixer.Mixer* attribute), 71  
`voltage` (*microcontroller.Processor* attribute), 124

## W

`wake_alarm` (in module *alarm*), 61  
`watchdog`  
    module, 170  
`watchdog` (in module *microcontroller*), 123  
`WATCHDOG` (*microcontroller.ResetReason* attribute), 124  
`WatchDogMode` (class in *watchdog*), 170  
`WatchDogTimer` (class in *watchdog*), 170  
`WaveFile` (class in *audiocore*), 67  
`wheel()` (in module *\_pixelbuf*), 54  
`width` (*displayio.Bitmap* attribute), 97  
`width` (*displayio.Display* attribute), 101  
`width` (*displayio.EPaperDisplay* attribute), 103  
`width` (*displayio.OnDiskBitmap* attribute), 106  
`width` (*framebufferio.FramebufferDisplay* attribute), 111  
`width` (*rgbmatrix.RGBMatrix* attribute), 137  
`width()` (*sdioio.SDCard* property), 145  
`wifi`  
    module, 171  
`wiznet`  
    module, 172  
`WIZNET5K` (class in *wiznet*), 172  
`wrap_socket()` (in module *ssl*), 149  
`WRITE` (*\_bleio.Characteristic* attribute), 39  
`write()` (*\_bleio.PacketBuffer* method), 43  
`write()` (*adafruit\_bus\_device.I2CDevice* method), 58  
`write()` (*bitbangio.SPI* method), 78  
`write()` (*busio.SPI* method), 86  
`write()` (*busio.UART* method), 88  
`write()` (*i2cperipheral.I2CPeripheralRequest* method), 118  
`write()` (*rp2pio.StateMachine* method), 140  
`write()` (*terminalio.Terminal* method), 153

`write()` (*usb\_cdc.Serial* method), 167  
`write()` (*usb\_midi.PortOut* method), 168  
`write()` (*usocket.socket* method), 219  
`write_bit()` (*bitbangio.OneWire* method), 77  
`write_bit()` (*busio.OneWire* method), 85  
`WRITE_NO_RESPONSE` (*\_bleio.Characteristic* attribute), 39  
`write_readinto()` (*bitbangio.SPI* method), 78  
`write_readinto()` (*busio.SPI* method), 87  
`write_readinto()` (*rp2pio.StateMachine* method), 140  
`write_then_readinto()`  
    (*adafruit\_bus\_device.I2CDevice* method), 58  
`write_timeout` (*usb\_cdc.Serial* attribute), 166  
`WriteableBuffer` (in module *\_typing*), 56  
`writeblocks()` (*sdcardio.SDCard* method), 143  
`writeblocks()` (*sdioio.SDCard* method), 144  
`writeto()` (*bitbangio.I2C* method), 76  
`writeto()` (*busio.I2C* method), 83  
`writeto_then_readfrom()` (*bitbangio.I2C* method), 76  
`writeto_then_readfrom()` (*busio.I2C* method), 84

## X

`x` (*displayio.Group* attribute), 104  
`x` (*displayio.TileGrid* attribute), 108  
`x` (*vectorio.VectorShape* attribute), 169

## Y

`y` (*displayio.Group* attribute), 104  
`y` (*displayio.TileGrid* attribute), 109  
`y` (*vectorio.VectorShape* attribute), 169

## Z

`ZeroDivisionError`, 200  
`zeros()` (in module *ulab*), 165  
`zip()`  
    built-in function, 199