Note

You are not reading the most recent version of this documentation. v1.6.2 is the latest version available.

Generic Devices

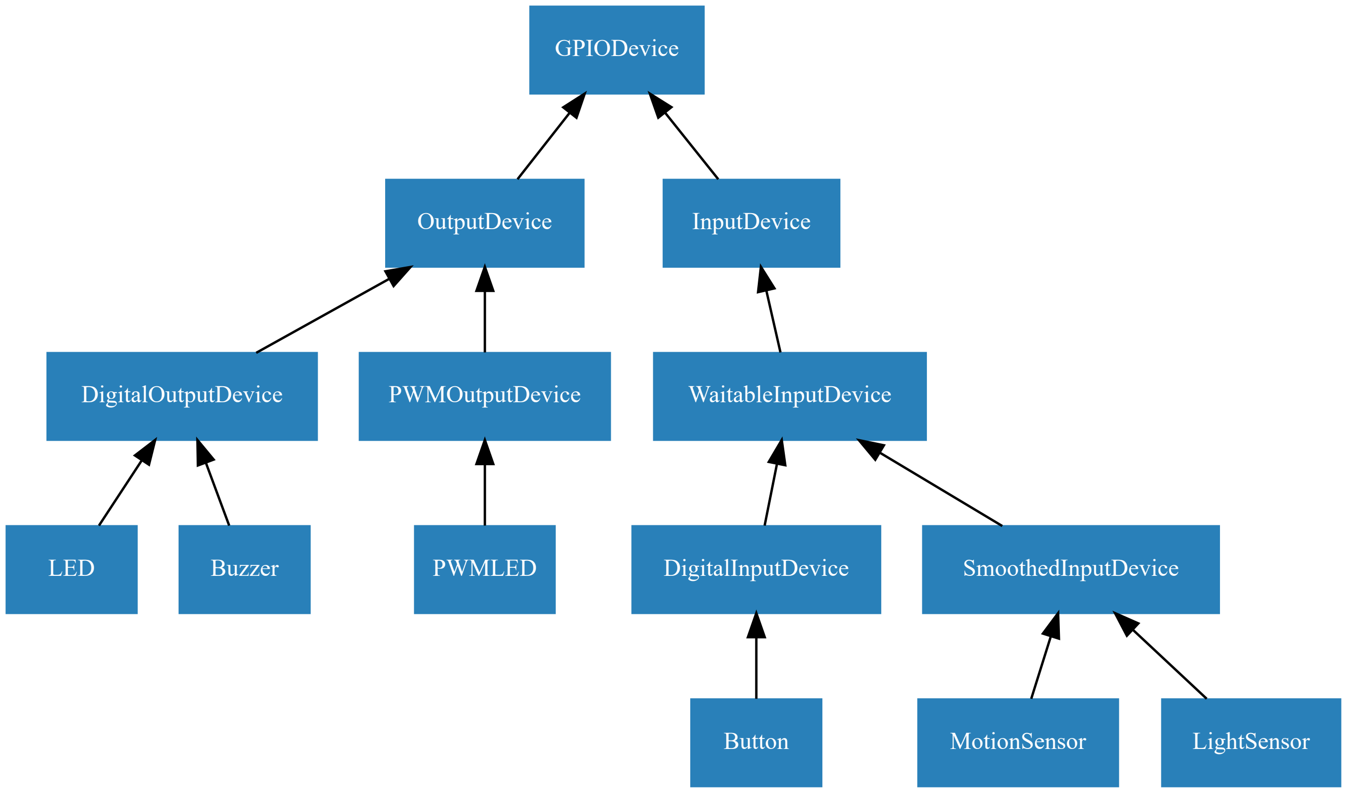
The GPIO Zero class hierarchy is quite extensive. It contains a couple of base classes:

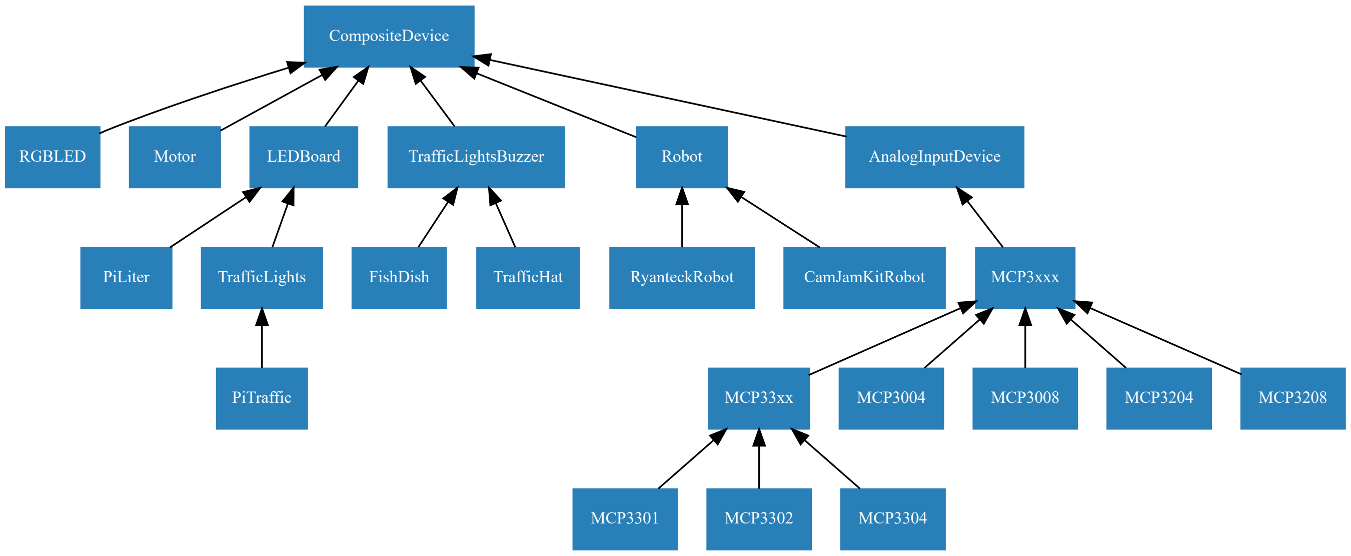
- `GPIODevice` for individual devices that attach to a single GPIO pin
- `CompositeDevice` for devices composed of multiple other devices like HATs

There are also a couple of mixin classes:

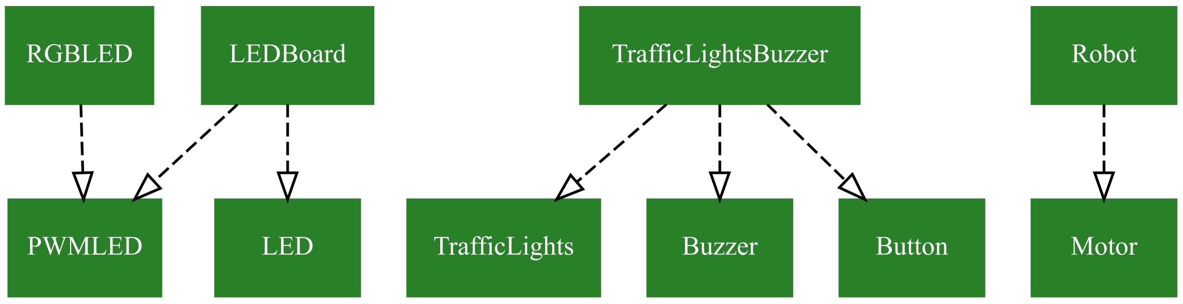
- `ValuesMixin` which defines the `values` properties; there is rarely a need to use this as the base classes mentioned above both include it (so all classes in GPIO Zero include the `values` property)
- `SourceMixin` which defines the `source` property; this is generally included in novel output device classes

The current class hierarchies are displayed below. For brevity, the mixin classes are omitted:





Finally, for composite devices, the following chart shows which devices are composed of which other devices:



Base Classes

class **gpiozero.GPIODevice**(*pin*) [source]

Represents a generic GPIO device.

This is the class at the root of the gpiozero class hierarchy. It handles ensuring that two GPIO devices do not share the same pin, and provides basic services applicable to all devices (specifically the `pin` property, `is_active` property, and the `close` method).

Parameters: **pin** (*int*) – The GPIO pin (in BCM numbering) that the device is connected to. If this is `None` a `GPIODeviceError` will be raised.

close() [source]

Shut down the device and release all associated resources.

This method is primarily intended for interactive use at the command line. It disables the device and releases its pin for use by another device.

You can attempt to do this simply by deleting an object, but unless you've cleaned up all references to the object this may not work (even if you've cleaned up all references, there's still no guarantee the garbage collector will actually delete the object at that point). By contrast, the close method provides a means of ensuring that the object is shut down.

For example, if you have a breadboard with a buzzer connected to pin 16, but then wish to attach an LED instead:

```
>>> from gpiozero import *
>>> bz = Buzzer(16)
>>> bz.on()
>>> bz.off()
>>> bz.close()
>>> led = LED(16)
>>> led.blink()
```

`GPIODevice` descendents can also be used as context managers using the `with` statement. For example:

```
>>> from gpiozero import *
>>> with Buzzer(16) as bz:
...     bz.on()
...
>>> with LED(16) as led:
...     led.on()
...
```

is_active

Returns `True` if the device is currently active and `False` otherwise.

`pin`

The `Pin` that the device is connected to. This will be `None` if the device has been closed (see the `close()` method). When dealing with GPIO pins, query `pin.number` to discover the GPIO pin (in BCM numbering) that the device is connected to.

`value`

Returns `True` if the device is currently active and `False` otherwise.

`values`

An infinite iterator of values read from *value*.

class `gpiozero.CompositeDevice` [source]

Represents a device composed of multiple GPIO devices like simple HATs, H-bridge motor controllers, robots composed of multiple motors, etc.

`close()`

Shut down the device and release all associated resources.

`closed`

Returns `True` if the device is closed (see the `close()` method). Once a device is closed you can no longer use any other methods or properties to control or query the device.

`values`

An infinite iterator of values read from *value*.

Input Devices

class `gpiozero.InputDevice`(*pin*, *pull_up=False*) [source]

Represents a generic GPIO input device.

This class extends `GPIODevice` to add facilities common to GPIO input devices. The constructor adds the optional *pull_up* parameter to specify how the pin should be pulled by the internal resistors. The `is_active` property is adjusted accordingly so that `True` still means active regardless of the `pull_up` setting.

Parameters:

- **pin** (*int*) – The GPIO pin (in Broadcom numbering) that the device is connected to. If this is `None` a `GPIODeviceError` will be raised.
- **pull_up** (*bool*) – If `True`, the device will be pulled high with an internal resistor. If `False` (the default), the pin will be pulled low.

`pull_up`

If `True`, the device uses a pull-up resistor to set the GPIO pin "high" by default. Defaults to `False`.

class `gpiozero.WaitableInputDevice`(*pin=None, pull_up=False*) [source]

Represents a generic input device with distinct waitable states.

This class extends `InputDevice` with methods for waiting on the device's status (`wait_for_active()` and `wait_for_inactive()`), and properties that hold functions to be called when the device changes state (`when_activated()` and `when_deactivated()`). These are aliased appropriately in various subclasses.

Note that this class provides no means of actually firing its events; it's effectively an abstract base class.

`wait_for_active`(*timeout=None*) [source]

Pause the script until the device is activated, or the timeout is reached.

Parameters: **timeout** (*float*) – Number of seconds to wait before proceeding. If this is `None` (the default), then wait indefinitely until the device is active.

`wait_for_inactive`(*timeout=None*) [source]

Pause the script until the device is deactivated, or the timeout is reached.

Parameters: **timeout** (*float*) – Number of seconds to wait before proceeding. If this is `None` (the default), then wait indefinitely until the device is inactive.

`when_activated`

The function to run when the device changes state from inactive to active.

This can be set to a function which accepts no (mandatory) parameters, or a Python function which accepts a single mandatory parameter (with as many optional parameters as you like). If the function accepts a single mandatory parameter, the device that activated will be passed as that parameter.

Set this property to `None` (the default) to disable the event.

`when_deactivated`

The function to run when the device changes state from active to inactive.

This can be set to a function which accepts no (mandatory) parameters, or a Python function which accepts a single mandatory parameter (with as many optional parameters as you like). If the function accepts a single mandatory parameter, the device that deactivated will be passed as that parameter.

Set this property to `None` (the default) to disable the event.

class `gpiozero.DigitalInputDevice`*(pin, pull_up=False, bounce_time=None)* [source]

Represents a generic input device with typical on/off behaviour.

This class extends `WaitableInputDevice` with machinery to fire the active and inactive events for devices that operate in a typical digital manner: straight forward on / off states with (reasonably) clean transitions between the two.

Parameters: **bouncetime** (*float*) – Specifies the length of time (in seconds) that the component will ignore changes in state after an initial change. This defaults to `None` which indicates that no bounce compensation will be performed.

class `gpiozero.SmoothedInputDevice`*(pin=None, pull_up=False, threshold=0.5, queue_len=5, sample_wait=0.0, partial=False)* [source]

Represents a generic input device which takes its value from the mean of a queue of historical values.

This class extends `WaitableInputDevice` with a queue which is filled by a background thread which continually polls the state of the underlying device. The mean of the values in the queue is compared to a threshold which is used to determine the state of the `is_active` property.

This class is intended for use with devices which either exhibit analog behaviour (such as the charging time of a capacitor with an LDR), or those which exhibit "twitchy" behaviour (such as certain motion sensors).

Parameters:

- **threshold** (*float*) – The value above which the device will be considered “on”.
- **queue_len** (*int*) – The length of the internal queue which is filled by the background thread.
- **sample_wait** (*float*) – The length of time to wait between retrieving the state of the underlying device. Defaults to 0.0 indicating that values are retrieved as fast as possible.
- **partial** (*bool*) – If `False` (the default), attempts to read the state of the device (from the `is_active` property) will block until the queue has filled. If `True`, a value will be returned immediately, but be aware that this value is likely to fluctuate excessively.

`close()` [source]

Shut down the device and release all associated resources.

This method is primarily intended for interactive use at the command line. It disables the device and releases its pin for use by another device.

You can attempt to do this simply by deleting an object, but unless you’ve cleaned up all references to the object this may not work (even if you’ve cleaned up all references, there’s still no guarantee the garbage collector will actually delete the object at that point). By contrast, the close method provides a means of ensuring that the object is shut down.

For example, if you have a breadboard with a buzzer connected to pin 16, but then wish to attach an LED instead:

```
>>> from gpiozero import *
>>> bz = Buzzer(16)
>>> bz.on()
>>> bz.off()
>>> bz.close()
>>> led = LED(16)
>>> led.blink()
```

`GPIODevice` descendents can also be used as context managers using the `with` statement. For example:

```
>>> from gpiozero import *
>>> with Buzzer(16) as bz:
...     bz.on()
...
>>> with LED(16) as led:
...     led.on()
...
```

`is_active`

Returns `True` if the device is currently active and `False` otherwise.

`partial`

If `False` (the default), attempts to read the `value` or `is_active` properties will block until the queue has filled.

`queue_len`

The length of the internal queue of values which is averaged to determine the overall state of the device. This defaults to 5.

`threshold`

If `value` exceeds this amount, then `is_active` will return `True`.

`value`

Returns the mean of the values in the internal queue. This is compared to `threshold` to determine whether `is_active` is `True`.

class **`gpiozero.AnalogInputDevice`**(*device=0, bits=None*) [source]

Represents an analog input device connected to SPI (serial interface).

Typical analog input devices are analog to digital converters (ADCs). Several classes are provided for specific ADC chips, including `MCP3004`, `MCP3008`, `MCP3204`, and `MCP3208`.

The following code demonstrates reading the first channel of an MCP3008 chip attached to the Pi's SPI pins:

```
from gpiozero import MCP3008

pot = MCP3008(0)
print(pot.value)
```

The `value` attribute is normalized such that its value is always between 0.0 and 1.0 (or in special cases, such as differential sampling, -1 to +1). Hence, you can use an analog input to control the brightness of a `PWMLED` like so:

```
from gpiozero import MCP3008, PWMLED

pot = MCP3008(0)
led = PWMLED(17)
led.source = pot.values
```

`close()` [source]

Shut down the device and release all associated resources.

`bits`

The bit-resolution of the device/channel.

`bus`

The SPI bus that the device is connected to. As the Pi only has a single (user accessible) SPI bus, this always returns 0.

`device`

The select pin that the device is connected to. The Pi has two select pins so this will be 0 or 1.

`raw_value`

The raw value as read from the device.

`value`

The current value read from the device, scaled to a value between 0 and 1.

Output Devices

class `gpiozero.OutputDevice`(*pin*, *active_high=True*, *initial_value=False*) [source]

Represents a generic GPIO output device.

This class extends `GPIODevice` to add facilities common to GPIO output devices: an `on()` method to switch the device on, and a corresponding `off()` method.

Parameters:

- **pin** (*int*) – The GPIO pin (in BCM numbering) that the device is connected to. If this is `None` a `GPIODeviceError` will be raised.
- **active_high** (*bool*) – If `True` (the default), the `on()` method will set the GPIO to HIGH. If `False`, the `on()` method will set the GPIO to LOW (the `off()` method always does the opposite).
- **initial_value** (*bool*) – If `False` (the default), the device will be off initially. If `None`, the device will be left in whatever state the pin is found in when configured for output (warning: this can be on). If `True`, the device will be switched on initially.

`off()` [source]

Turns the device off.

`on()` [source]

Turns the device on.

---

class `gpiozero.PWMOutputDevice`(*pin*, *active_high=True*, *initial_value=0*, *frequency=100*) [source]

Generic output device configured for pulse-width modulation (PWM).

Parameters:

- **pin** (*int*) – The GPIO pin which the device is attached to. See *Notes* for valid pin numbers.
- **active_high** (*bool*) – If `True` (the default), the `on()` method will set the GPIO to HIGH. If `False`, the `on()` method will set the GPIO to LOW (the `off()` method always does the opposite).
- **initial_value** (*bool*) – If `0` (the default), the device's duty cycle will be 0 initially. Other values between 0 and 1 can be specified as an initial duty cycle. Note that `None` cannot be specified (unlike the parent class) as there is no way to tell PWM not to alter the state of the pin.
- **frequency** (*int*) – The frequency (in Hz) of pulses emitted to drive the device. Defaults to 100Hz.

`blink`(*on_time=1*, *off_time=1*, *fade_in_time=0*, *fade_out_time=0*, *n=None*, *background=True*) [source]

Make the device turn on and off repeatedly.

Parameters:

- **on_time** (*float*) – Number of seconds on. Defaults to 1 second.
- **off_time** (*float*) – Number of seconds off. Defaults to 1 second.
- **fade_in_time** (*float*) – Number of seconds to spend fading in. Defaults to 0.
- **fade_out_time** (*float*) – Number of seconds to spend fading out. Defaults to 0.
- **n** (*int*) – Number of times to blink; `None` (the default) means forever.
- **background** (*bool*) – If `True` (the default), start a background thread to continue blinking and return immediately. If `False`, only return when the blink is finished (warning: the default value of *n* will result in this method never returning).

`close()` [source]

Shut down the device and release all associated resources.

This method is primarily intended for interactive use at the command line. It disables the device and releases its pin for use by another device.

You can attempt to do this simply by deleting an object, but unless you've cleaned up all references to the object this may not work (even if you've cleaned up all references, there's still no guarantee the garbage collector will actually delete the object at that point). By contrast, the close method provides a means of ensuring that the object is shut down.

For example, if you have a breadboard with a buzzer connected to pin 16, but then wish to attach an LED instead:

```
>>> from gpiozero import *
>>> bz = Buzzer(16)
>>> bz.on()
>>> bz.off()
>>> bz.close()
>>> led = LED(16)
>>> led.blink()
```

`GPIODevice` descendents can also be used as context managers using the `with` statement. For example:

```
>>> from gpiozero import *
>>> with Buzzer(16) as bz:
...     bz.on()
...
>>> with LED(16) as led:
...     led.on()
...
```

off() [source]

Turns the device off.

on() [source]

Turns the device on.

toggle() [source]

Toggle the state of the device. If the device is currently off (`value` is 0.0), this changes it to "fully" on (`value` is 1.0). If the device has a duty cycle (`value`) of 0.1, this will toggle it to 0.9, and so on.

frequency

The frequency of the pulses used with the PWM device, in Hz. The default is 100Hz.

is_active

Returns `True` if the device is currently active (`value` is non-zero) and `False` otherwise.

value

The duty cycle of the PWM device. 0.0 is off, 1.0 is fully on. Values in between may be specified for varying levels of power in the device.

class **gpiozero.DigitalOutputDevice**(*pin, active_high=True, initial_value=False*) [source]

Represents a generic output device with typical on/off behaviour.

This class extends `OutputDevice` with a `toggle()` method to switch the device between its on and off states, and a `blink()` method which uses an optional background thread to handle toggling the device state without further interaction.

blink(*on_time=1, off_time=1, n=None, background=True*) [source]

Make the device turn on and off repeatedly.

Parameters:

- **on_time** (*float*) – Number of seconds on. Defaults to 1 second.
- **off_time** (*float*) – Number of seconds off. Defaults to 1 second.
- **n** (*int*) – Number of times to blink; `None` (the default) means forever.
- **background** (*bool*) – If `True` (the default), start a background thread to continue blinking and return immediately. If `False`, only return when the blink is finished (warning: the default value of *n* will result in this method never returning).

`close()` [source]

Shut down the device and release all associated resources.

This method is primarily intended for interactive use at the command line. It disables the device and releases its pin for use by another device.

You can attempt to do this simply by deleting an object, but unless you've cleaned up all references to the object this may not work (even if you've cleaned up all references, there's still no guarantee the garbage collector will actually delete the object at that point). By contrast, the close method provides a means of ensuring that the object is shut down.

For example, if you have a breadboard with a buzzer connected to pin 16, but then wish to attach an LED instead:

```
>>> from gpiozero import *
>>> bz = Buzzer(16)
>>> bz.on()
>>> bz.off()
>>> bz.close()
>>> led = LED(16)
>>> led.blink()
```

`GPIODevice` descendents can also be used as context managers using the `with` statement. For example:

```
>>> from gpiozero import *
>>> with Buzzer(16) as bz:
...     bz.on()
...
>>> with LED(16) as led:
...     led.on()
...
```

`off()` [source]

Turns the device off.

`on()` [source]

Turns the device on.

`toggle()` [source]

Reverse the state of the device. If it's on, turn it off; if it's off, turn it on.

Mixin Classes

class `gpiozero.ValuesMixin`(...) [source]

`values`

An infinite iterator of values read from *value*.

class `gpiozero.SourceMixin`(...) [source]

`source`

The iterable to use as a source of values for *value*.