rpi-lgpio 0.6 Documentation

Release 0.6

Dave Jones

May 11, 2024

CONTENTS

CHAPTER

ONE

INSTALLATION

rpi-lgpio is distributed in several formats. The following sections detail installation from a variety of formats. But first a warning:

Warning: You *cannot* install rpi-lgpio and rpi-gpio (aka RPi.GPIO, the library it emulates) at the same time, in the same Python environment. Both packages attempt to install a module named `RPi.GPIO` and obviously this will not work.

1.1 apt/deb package

If your distribution includes rpi-lgpio in its archive of apt packages, then you can simply:

```
$ sudo apt install python3-rpi-lgpio
```

If you wish to go back to rpi-gpio:

```
$ sudo apt remove python3-rpi-lgpio
$ sudo apt install python3-rpi.gpio
```

1.2 wheel package

If your distribution does not include a “native” packaging of rpi-lgpio, you can also install rpi-lgpio from PyPI[1] using pip. Please note that rpi-lgpio does still depend on lgpio[2] so you will need that installed as a dependency.

Note: It is strongly recommended that you install in a virtualenv if persuing this method, in which case you have a choice as to whether lgpio is provided by a system package (such as apt), or another wheel.

The following sections demonstrate installing from a wheel in a variety of scenarios.

[1] https://pypi.org/project/rpi-lgpio/
[2] https://pypi.org/project/lgpio/

1.2.1 in venv without system packages

Construct a “clean” virutalenv with no access to system packages, then install rpi-lgpio as a wheel within that virtualenv, trusting it to pull an appropriate lgpio dependency from PyPI as another wheel:

```
$ python3 -m venv cleanvenv
$ source cleanvenv/bin/activate
(cleanvenv) $ pip3 install rpi-lgpio
```

1.2.2 in venv with system packages

Install the lgpio dependency as a system package, construct a virtualenv with access to system packages, and install rpi-lgpio as a wheel within that virtualenv:

```
$ sudo apt install python3-lgpio
$ sudo apt remove python3-rpi.gpio
$ python3 -m venv --system-site-packages sysvenv
$ source sysvenv/bin/activate
(sysvenv) $ pip3 install rpi-lgpio
```

Note that in this case we also ensure that we remove any system-level RPi.GPIO installation that may interfere.

1.2.3 outside venv (system-wide)

If you wish to install system-wide with pip, you may need to place `sudo` in front of the `pip` (or `pip3`) commands too. Please be aware that on modern versions of pip you will need to explicitly accept the risk of trying to co-exist `apt` and `pip` packages as follows:

```
$ sudo pip3 install --break-system-packages rpi-lgpio
```

Warning: This is not an advised mode of installation, unless you are quite certain that you know what pip is going to pull in. Upgrading such an installation is also particularly risky.

CHAPTER

TWO

DIFFERENCES

Many of the assumptions underlying RPi.GPIO[3] – that it has complete access to, and control over, the registers controlling the GPIO pins – do not work when applied to the Linux gpiochip devices. To that end, while the library strives as far as possible to be "bug compatible" with RPi.GPIO, there *are* differences in behaviour that may result in incompatibility.

2.1 Bug Compatible?

What does being "bug compatible" mean? It is not enough for the library to implement the RPi.GPIO[4] API. It must also:

- Act, as far as possible, in the same way to the same calls with the same values
- Raise the same exception types, with the same messages, in the same circumstances
- Break (i.e. fail to operate correctly) in the same way, as far as possible

This last point may sound silly, but a library is *always* used in unexpected or undocumented ways by *some* applications. Thus anything that tries to take the place of that library must do more than simply operate the same as the "documented surface" would suggest.

That said, given that the underlying assumptions are fundamentally different this will not always be possible…

2.2 Pi Revision

The RPi.GPIO module attempts to determine the revision of Raspberry Pi board that it is running on when the module is imported by querying `/proc/cpuinfo`, raising `RuntimeError`[5] at import time if it finds it is not running on a Raspberry Pi. rpi-lgpio emulates this behaviour, but this can be inconvenient for certain situations including testing, and usage of rpi-lgpio on other single board computers.

To that end rpi-lgpio permits a Raspberry Pi revision code[6] to be manually specified via the environment in the `RPI_LGPIO_REVISION` value (when this is set, `/proc/cpuinfo` is not read at all). For example:

```
$ RPI_LGPIO_REVISION="c03114" python3
Python 3.10.6 (main, Aug 10 2022, 11:40:04) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from RPi import GPIO
>>> GPIO.RPI_INFO
{'P1_REVISION': 3, 'REVISION': 'c03114', 'TYPE': 'Pi 4 Model B',
'MANUFACTURER': 'Sony UK', 'PROCESSOR': 'BCM2711', 'RAM': '4GB'}
>>> exit()
$ RPI_LGPIO_REVISION="902120" python3
```

(continues on next page)

[3] https://pypi.org/project/RPi.GPIO/
[4] https://pypi.org/project/RPi.GPIO/
[5] https://docs.python.org/3/library/exceptions.html#RuntimeError
[6] https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#new-style-revision-codes

(continued from previous page)

```
Python 3.10.6 (main, Aug 10 2022, 11:40:04) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from RPi import GPIO
>>> GPIO.RPI_INFO
{'P1_REVISION': 3, 'REVISION': '902120', 'TYPE': 'Zero 2 W',
'MANUFACTURER': 'Sony UK', 'PROCESSOR': 'BCM2837', 'RAM': '512M'}
>>> exit()
```

At present, rpi-lgpio only interprets "new-style[7]" (6 hex-digit) revision codes, as found in the "Revision" field of `/proc/cpuinfo`. The old-style[8] (4 hex-digit) revision codes found on the original model B, A, A+, B+, and Compute Module 1 are not supported. If there is significant demand, this can be added but for the time being only boards made since the launch of the 2B (which introduced the new-style revision codes) are supported. Specifically, this includes the following models:

- Zero
- Zero W
- Zero 2W
- 2B
- 3B
- Compute Module 3
- 3A+
- 3B+
- Compute Module 3+
- 4B
- 400
- Compute Module 4
- 5B

A workaround for use on old-style boards is to use `RPI_LGPIO_REVISION` to fake the revision code. For example, 0004 (an old-style model B rev 2) can also be represented by 800012 in the new-style.

```
$ RPI_LGPIO_REVISION="800012" python3
Python 3.12.2 (main, Apr  2 2024, 18:40:52) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from RPi import GPIO
>>> GPIO.RPI_INFO
{'P1_REVISION': 2, 'REVISION': '800012', 'TYPE': 'Model B',
'MANUFACTURER': 'Sony UK', 'PROCESSOR': 'BCM2835', 'RAM': '256M'}
>>> exit()
```

[7] https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#new-style-revision-codes
[8] https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#old-style-revision-codes

2.3 GPIO Chip

The lgpio library needs to know the number of the `/dev/gpiochip` device it should open. By default this will be calculated from the reported *Pi Revision* (page 3) (which may be customized as detailed in that section). In practice this means the chip defaults to "4" on the Raspberry Pi Model 5B, and "0" on all other boards.

You may also specify the chip manually using the `RPI_LGPIO_CHIP` environment variable. For example:

```
$ ls /dev/gpiochip*
crw-------  1 root root    254, 0 Oct  1 15:00 /dev/gpiochip0
crw-------  1 root root    254, 1 Oct  1 15:00 /dev/gpiochip1
crw-------  1 root root    254, 2 Oct  1 15:00 /dev/gpiochip2
crw-------  1 root root    254, 3 Oct  1 15:00 /dev/gpiochip3
crw-rw----+ 1 root dialout 254, 4 Oct  1 15:00 /dev/gpiochip4
crw-------  1 root root    254, 5 Oct  1 15:00 /dev/gpiochip5
$ RPI_LGPIO_CHIP=5 python3
Python 3.11.5 (main, Aug 29 2023, 15:31:31) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from RPi import GPIO
>>> GPIO.setmode(GPIO.BCM)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python3/dist-packages/RPi/GPIO/__init__.py", line 513, in setmode
    _chip = _check(lgpio.gpiochip_open(int(chip_num)))
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/lib/python3/dist-packages/lgpio.py", line 645, in gpiochip_open
    return _u2i(handle)
           ^^^^^^^^^^^^
  File "/usr/lib/python3/dist-packages/lgpio.py", line 458, in _u2i
    raise error(error_text(v))
lgpio.error: 'can not open gpiochip'
```

This is primarily useful for other boards where the correct gpiochip device is something other than 0.

2.4 Alternate Pin Modes

The *`gpio_function()`* (page 14) function can be used to report the current mode of a pin. In RPi.GPIO this may return several "alternate" mode values including *`SPI`* (page 16), *`I2C`* (page 16), and *`HARD_PWM`* (page 16). rpi-lgpio will only ever return the basic *`IN`* (page 16) and *`OUT`* (page 15) values however, as the underlying gpiochip device cannot report alternate modes.

For example, under RPi.GPIO:

```
>>> from RPi import GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.gpio_function(2) == GPIO.I2C
True
```

Under rpi-lgpio:

```
>>> from RPi import GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.gpio_function(2) == GPIO.I2C
False
>>> GPIO.gpio_function(2) == GPIO.IN
True
```

2.5 Stack Traces

While every effort has been made to raise the same exceptions with the same messages as RPi.GPIO, rpi-lgpio does raise the exceptions from pure Python so the exceptions will generally include a larger stack trace than under RPi.GPIO. For example, under RPi.GPIO:

```
>>> from RPi import GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.setup(26, GPIO.IN)
>>> GPIO.output(26, 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: The GPIO channel has not been set up as an OUTPUT
```

Under rpi-lgpio:

```
>>> from RPi import GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.setup(26, GPIO.IN)
>>> GPIO.output(26, 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/dave/projects/rpi-lgpio/rpi-lgpio/RPi/GPIO.py", line 626, in output
    _check_output(mode, 'The GPIO channel has not been set up as an OUTPUT')
  File "/home/dave/projects/rpi-lgpio/rpi-lgpio/RPi/GPIO.py", line 242, in _check_
↪output
    raise RuntimeError(msg)
RuntimeError: The GPIO channel has not been set up as an OUTPUT
```

2.6 Simultaneous Access

Two processes using RPi.GPIO can happily control the same pin. This is simply not permitted by the Linux gpiochip device and will fail under rpi-lgpio. For example, if another process has reserved GPIO26, and our script also tries to allocate it:

```
>>> from RPi import GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.setup(26, GPIO.OUT)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/dave/projects/rpi-lgpio/rpi-lgpio/RPi/GPIO.py", line 569, in setup
    initial = _check(lgpio.gpio_read(_chip, gpio))
  File "/home/dave/envs/rpi-lgpio/lib/python3.10/site-packages/lgpio.py", line 894,
↪ in gpio_read
    return _u2i(_lgpio._gpio_read(handle&0xffff, gpio))
  File "/home/dave/envs/rpi-lgpio/lib/python3.10/site-packages/lgpio.py", line 461,
↪ in _u2i
    raise error(error_text(v))
lgpio.error: 'GPIO not allocated'
```

How can you tell if a GPIO is reserved by another process? Use the *gpioinfo(1)*[9] tool, which is part of the `gpiod` package. By default this attempts to read GPIO chip 0, which is fine on all Pi's *except* the Pi 5 where you will need to read GPIO chip 4 specifically:

```
$ gpioinfo 4
gpiochip4 - 54 lines:
    line   0:      "ID_SDA"       unused   input  active-high
```

(continues on next page)

[9] https://manpages.ubuntu.com/manpages/noble/en/man1/gpioinfo.1.html

(continued from previous page)

```
    line   1:     "ID_SCL"       unused   input  active-high
    line   2:      "GPIO2"       unused   input  active-high
    line   3:      "GPIO3"       unused   input  active-high
    line   4:      "GPIO4"       unused   input  active-high
    line   5:      "GPIO5"       unused   input  active-high
    line   6:      "GPIO6"       unused   input  active-high
    line   7:      "GPIO7"   "spi0 CS1"  output   active-low [used]
    line   8:      "GPIO8"   "spi0 CS0"  output   active-low [used]
    line   9:      "GPIO9"       unused   input  active-high
    line  10:     "GPIO10"       unused   input  active-high
    line  11:     "GPIO11"       unused   input  active-high
    line  12:     "GPIO12"       unused   input  active-high
    line  13:     "GPIO13"       unused   input  active-high
    line  14:     "GPIO14"       unused   input  active-high
    line  15:     "GPIO15"       unused   input  active-high
    line  16:     "GPIO16"       unused   input  active-high
    line  17:     "GPIO17"       unused   input  active-high
    line  18:     "GPIO18"       unused   input  active-high
    line  19:     "GPIO19"       unused   input  active-high
    line  20:     "GPIO20"       unused   input  active-high
    line  21:     "GPIO21"       unused   input  active-high
    line  22:     "GPIO22"       unused   input  active-high
    line  23:     "GPIO23"       unused   input  active-high
    line  24:     "GPIO24"       unused   input  active-high
    line  25:     "GPIO25"       unused   input  active-high
    line  26:     "GPIO26"       unused   input  active-high
    line  27:     "GPIO27"       unused   input  active-high
    line  28: "PCIE_RP1_WAKE" unused output active-high
    line  29:   "FAN_TACH"       unused   input  active-high
    line  30:   "HOST_SDA"       unused   input  active-high
    line  31:   "HOST_SCL"       unused   input  active-high
    line  32:  "ETH_RST_N"  "phy-reset"  output   active-low [used]
    line  33:          "-"       unused   input  active-high
    line  34: "CD0_IO0_MICCLK" "cam0_reg" output active-high [used]
    line  35: "CD0_IO0_MICDAT0" unused input active-high
    line  36: "RP1_PCIE_CLKREQ_N" unused input active-high
    line  37:          "-"       unused   input  active-high
    line  38:    "CD0_SDA"       unused   input  active-high
    line  39:    "CD0_SCL"       unused   input  active-high
    line  40:    "CD1_SDA"       unused   input  active-high
    line  41:    "CD1_SCL"       unused   input  active-high
    line  42: "USB_VBUS_EN" unused output active-high
    line  43:   "USB_OC_N"       unused   input  active-high
    line  44: "RP1_STAT_LED" "PWR" output active-low [used]
    line  45:    "FAN_PWM"       unused  output  active-high
    line  46: "CD1_IO0_MICCLK" "cam1_reg" output active-high [used]
    line  47:  "2712_WAKE"       unused   input  active-high
    line  48: "CD1_IO1_MICDAT1" unused input active-high
    line  49: "EN_MAX_USB_CUR" unused output active-high
    line  50:          "-"       unused   input  active-high
    line  51:          "-"       unused   input  active-high
    line  52:          "-"       unused   input  active-high
    line  53:          "-"       unused   input  active-high
```

The `[used]` suffixes indicate which GPIOs are reserved by other processes. In the output above we can see that GPIOs 7 and 8 are reserved. As it happens, these are reserved by the kernel because we have `dtparam=spi=on` in our boot configuration to enable the kernel SPI devices (`/dev/spidev0.0` and `/dev/spidev0.1`). As a result, these GPIOs *cannot* be used by rpi-lgpio because the kernel will not let anything else reserve them. They can only be used for SPI via those kernel devices, and the only way to release those GPIOs would be to change our kernel / boot configuration.

In other cases we may find that a GPIO is temporarily reserved by a process. For example, the following trivial script will reserve GPIO21.

```python
from time import sleep
from RPi import GPIO

GPIO.setmode(GPIO.BCM)
GPIO.setup(21, GPIO.OUT)
while True:
    sleep(1)
```

If we again query *gpioinfo(1)*[10] while it is running we will see the following:

```
$ gpioinfo 4 | grep GPIO21
        line  21:     "GPIO21"        "lg"  output  active-high [used bias-
↪disabled]
```

However, this reservation will disappear when the process dies.

Note: If you receive the `GPIO not allocated` error in your script, please check the output of *gpioinfo(1)*[11] to see if the GPIO you want to use is reserved by something else.

2.7 Debounce

Debouncing of signals works fundamentally differently in RPi.GPIO, and in lgpio[12] (the library underlying rpi-lgpio). Rather than attempt to add more complexity in between users and lgpio, which would also inevitably slow down edge detection (with all the attendant timing issues for certain applications) it is likely preferable to just live with this difference, but document it thoroughly.

RPi.GPIO debounces signals by tracking the last timestamp at which it saw a specified edge and suppressing reports of edges that occur within the specified number of milliseconds after that.

lgpio (and thus rpi-lgpio) debounces by waiting for a signal to be stable for the specified number of milliseconds before reporting the edge.

For some applications, there will be little/no difference other than rpi-lgpio reporting an edge a few milliseconds later than RPi.GPIO would (specifically, by the amount of debounce requsted). The following diagram shows the waveform from a "bouncy" switch being pressed once, along with the points in time where RPi.GPIO and rpi-lgpio would report the rising edge when debounce of 3ms is requested:

```
 0ms    2ms     4ms     6ms     8ms
 |      |       |       |       |
 |      ┌─┐ ┌─┐ ┌─────────────────┐
 |      │ │ │ │ │           :     │
 |      │ │ │ │ │           :     │
────────┘ └─┘ └─┘           :     └────────────────────────
        :                   :
        :                   :
     RPi.GPIO             rpi-lgpio
```

RPi.GPIO reports the edge at 2ms, then suppresses the edges at 3ms and 4ms because they are within 3ms of the last edge. By contrast, rpi-lgpio ignores the first and second rising edges (because they didn't stay stable for 3ms) and only reports the third edge at 7ms (after it's spent 3ms stable).

However, consider this same scenario if debounce of 2ms is requested:

[10] https://manpages.ubuntu.com/manpages/noble/en/man1/gpioinfo.1.html
[11] https://manpages.ubuntu.com/manpages/noble/en/man1/gpioinfo.1.html
[12] https://abyz.me.uk/lg/py_lgpio.html

```
0ms    2ms     4ms     6ms     8ms
|      |       |       |       |
|      ┌─┐ ┌─┐ ┌─────────────────┐
|      | | | | |       :         |
|      | | | | |       :         |
───────┘ └─┘ └─┘       :         └────────────────────────
       :       :       :
       :       :       :
   RPi.GPIO  RPi.GPIO  rpi-lgpio
```

In this case, RPi.GPIO reports the switch *twice* because the third edge is at least 2ms after the first edge. However, rpi-lgpio only reports the switch *once* because only one edge stayed stable for 2ms. Also note in this case, that rpi-lgpio's report time has moved back to 6ms because it's not waiting as long for stability.

Note: This implies that you may find shorter debounce periods preferable when working with rpi-lgpio, than with RPi.GPIO. They will still debounce effectively, but will reduce the delay in reporting edges.

One final scenario to consider is a waveform of equally spaced, repeating pulses (like PWM) every 2ms:

```
0ms    2ms     4ms     6ms     8ms     10ms    12ms
|      |       |       |       |       |       |
|      ┌────┐  ┌────┐  ┌────┐  ┌────┐  ┌────┐  ┌────┐  ┌──
|      |    |  |    |  |    |  |    |  |    |  |    |  |
|      |    |  |    |  |    |  |    |  |    |  |    |  |
───────┘    └──┘    └──┘    └──┘    └──┘    └──┘    └──┘
       :               :               :               :
       :               :               :               :
  RPi.GPIO        RPi.GPIO        RPi.GPIO        RPi.GPIO
```

If we request rising edge detection with a debounce of 3ms, RPi.GPIO reports half of the edges; it's suppressing every other edge as they occur within 3ms of the edge preceding them. rpi-lgpio, on the other hand, reports *no* edges at all because none of them stay stable for 3ms.

2.8 PWM on inputs

RPi.GPIO (probably erroneously) permits PWM objects to continue operating on pins that are switched to inputs:

```
>>> from RPi import GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.setup(26, GPIO.OUT)
>>> p = GPIO.PWM(26, 1000)
>>> p.start(75)
>>> GPIO.setup(26, GPIO.IN)
>>> p.stop()
>>> p.start(75)
>>> p.stop()
```

This will not work under rpi-lgpio:

```
>>> from RPi import GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.setup(26, GPIO.OUT)
>>> p = GPIO.PWM(26, 1000)
>>> p.start(75)
>>> GPIO.setup(26, GPIO.IN)
>>> p.stop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

(continues on next page)

(continued from previous page)

```
  File "/home/dave/projects/rpi-lgpio/rpi-lgpio/RPi/GPIO.py", line 190, in stop
    lgpio.tx_pwm(_chip, self._gpio, 0, 0)
  File "/home/dave/envs/rpi-lgpio/lib/python3.10/site-packages/lgpio.py", line␣
↪1074, in tx_pwm
    return _u2i(_lgpio._tx_pwm(
  File "/home/dave/envs/rpi-lgpio/lib/python3.10/site-packages/lgpio.py", line 461,
↪ in _u2i
    raise error(error_text(v))
lgpio.error: 'bad PWM micros'
```

Though note that the error occurs when the `PWM` (page 14) object is *next* acted upon, rather than at the point when the GPIO is switched to an input.

CHAPTER

THREE

API REFERENCE

The API of rpi-lgpio (naturally) follows that of rpi-gpio (aka RPi.GPIO) as closely as possible. As such the following is simply a re-iteration of that API.

3.1 Initialization

`RPi.GPIO.`**`setmode`**`(`*new_mode*`)`

Set up the numbering mode to use for the pins on the board. The options for *new_mode* are:

- *`BOARD`* (page 15) - Use Raspberry Pi board numbers
- *`BCM`* (page 15) - Use Broadcom GPIO 00..nn numbers

If a numbering mode has already been set, and *new_mode* is not the same as the result of *`getmode()`* (page 11), a `ValueError`[13] is raised.

Parameters

`new_mode` (*`int`*[14]) – The new numbering mode to apply

`RPi.GPIO.`**`getmode`**`()`

Get the numbering mode used for the pins on the board. Returns *`BOARD`* (page 15), *`BCM`* (page 15) or `None`[15].

`RPi.GPIO.`**`setup`**`(`*chanlist*, *direction*, *pull_up_down=20*, *initial=None*`)`

Set up a GPIO channel or iterable of channels with a direction and (optionally, for inputs) pull/up down control, or (optionally, for outputs) and initial state.

The GPIOs to affect are listed in *chanlist* which may be any iterable. The *direction* is either *`IN`* (page 16) or *`OUT`* (page 15).

If *direction* is *`IN`* (page 16), then *pull_up_down* may specify one of the values *`PUD_UP`* (page 15) to set the internal pull-up resistor, *`PUD_DOWN`* (page 15) to set the internal pull-down resistor, or the default *`PUD_OFF`* (page 15) which disables the internal pulls.

If *direction* is *`OUT`* (page 15), then *initial* may specify zero or one to indicate the initial state of the output.

Parameters

- **`chanlist`** (*`list`*[16] *`or tuple`*[17] *`or int`*[18]) – The list of GPIO channels to setup
- **`direction`** (*`int`*[19]) – Whether the channels should act as inputs or outputs
- **`pull_up_down`** (*`int`*[20] *`or None`*) – The internal pull resistor (if any) to enable for inputs
- **`initial`** (*`bool`*[21] *`or int`*[22] *`or None`*) – The initial state of an output

[13] https://docs.python.org/3/library/exceptions.html#ValueError
[14] https://docs.python.org/3/library/functions.html#int
[15] https://docs.python.org/3/library/constants.html#None

`RPi.GPIO.`**`cleanup`**`(`*chanlist=None*`)`

Reset the specified GPIO channels (or all channels if none are specified) to INPUT with no pull-up / pull-down and no event detection.

Parameters

`chanlist` (*`list`*[23] *`or`* *`tuple`*[24] *`or`* *`int`*[25] *`or None`*) – The channel, or channels to clean up

3.2 Pin Usage

`RPi.GPIO.`**`input`**`(`*channel*`)`

Input from a GPIO *channel*. Returns 1 or 0.

This can also be called on a GPIO output, in which case the value returned will be the last state set on the GPIO.

Parameters

`channel` (*`int`*[26]) – The board pin number or BCM number depending on *`setmode()`* (page 11)

`RPi.GPIO.`**`output`**`(`*channel*, *value*`)`

Output to a GPIO *channel* or list of channels. The *value* can be the integer *`LOW`* (page 16) or *`HIGH`* (page 16), or a list of integers.

If a list of channels is specified, with a single integer for the *value* then it is applied to all channels. Otherwise, the length of the two lists must match.

Parameters

- **`channel`** (*`list`*[27] *`or`* *`tuple`*[28] *`or`* *`int`*[29]) – The GPIO channel, or list of GPIO channels to output to
- **`value`** (*`list`*[30] *`or`* *`tuple`*[31] *`or`* *`int`*[32]) – The value, or list of values to output

[16] https://docs.python.org/3/library/stdtypes.html#list
[17] https://docs.python.org/3/library/stdtypes.html#tuple
[18] https://docs.python.org/3/library/functions.html#int
[19] https://docs.python.org/3/library/functions.html#int
[20] https://docs.python.org/3/library/functions.html#int
[21] https://docs.python.org/3/library/functions.html#bool
[22] https://docs.python.org/3/library/functions.html#int
[23] https://docs.python.org/3/library/stdtypes.html#list
[24] https://docs.python.org/3/library/stdtypes.html#tuple
[25] https://docs.python.org/3/library/functions.html#int
[26] https://docs.python.org/3/library/functions.html#int
[27] https://docs.python.org/3/library/stdtypes.html#list
[28] https://docs.python.org/3/library/stdtypes.html#tuple
[29] https://docs.python.org/3/library/functions.html#int
[30] https://docs.python.org/3/library/stdtypes.html#list
[31] https://docs.python.org/3/library/stdtypes.html#tuple
[32] https://docs.python.org/3/library/functions.html#int

3.3 Edge Detection

`RPi.GPIO.`**`wait_for_edge`**`(`*channel*, *edge*, *bouncetime=None*, *timeout=None*`)`

Wait for an *edge* on the specified *channel*. Returns *channel* or `None`[33] if *timeout* elapses before the specified edge occurs.

Note: Debounce works significantly differently in rpi-lgpio than it does in rpi-gpio; please see *Debounce* (page 8) for more information on the differences.

Parameters

- **`channel`** (*`int`*[34]) – The board pin number or BCM number depending on *`setmode()`* (page 11) to watch for changes
- **`edge`** (*`int`*[35]) – One of the constants *`RISING`* (page 16), *`FALLING`* (page 16), or *`BOTH`* (page 16)
- **`bouncetime`** (*`int`*[36] *`or None`*) – Time (in ms) used to debounce signals
- **`timeout`** (*`int`*[37] *`or None`*) – Maximum time (in ms) to wait for the edge

`RPi.GPIO.`**`add_event_detect`**`(`*channel*, *edge*, *callback=None*, *bouncetime=None*`)`

Start background *edge* detection on the specified GPIO *channel*.

If *callback* is specified, it must be a callable that will be executed when the specified *edge* is seen on the GPIO *channel*. The callable must accept a single parameter: the channel on which the edge was detected.

Note: Debounce works significantly differently in rpi-lgpio than it does in rpi-gpio; please see *Debounce* (page 8) for more information on the differences.

Parameters

- **`channel`** (*`int`*[38]) – The board pin number or BCM number depending on *`setmode()`* (page 11) to watch for changes
- **`edge`** (*`int`*[39]) – One of the constants *`RISING`* (page 16), *`FALLING`* (page 16), or *`BOTH`* (page 16)
- **`callback`** (*`callable or None`*) – The callback to run when an edge is detected; must take a single integer parameter of the channel on which the edge was detected
- **`bouncetime`** (*`int`*[40] *`or None`*) – Time (in ms) used to debounce signals

`RPi.GPIO.`**`add_event_callback`**`(`*channel*, *callback*`)`

Add a *callback* to the specified GPIO *channel* which must already have been set for background edge detection with *`add_event_detect()`* (page 13).

Parameters

- **`channel`** (*`int`*[41]) – The board pin number or BCM number depending on *`setmode()`* (page 11) to watch for changes

[33] https://docs.python.org/3/library/constants.html#None
[34] https://docs.python.org/3/library/functions.html#int
[35] https://docs.python.org/3/library/functions.html#int
[36] https://docs.python.org/3/library/functions.html#int
[37] https://docs.python.org/3/library/functions.html#int
[38] https://docs.python.org/3/library/functions.html#int
[39] https://docs.python.org/3/library/functions.html#int
[40] https://docs.python.org/3/library/functions.html#int

- **callback** – The callback to run when an edge is detected; must take a single integer parameter of the channel on which the edge was detected

`RPi.GPIO.`**`event_detected`**`(`*channel*`)`

Returns `True`[42] if an edge has occurred on the specified *channel* since the last query of the channel (if any). Querying this will also reset the internal edge detected flag for this channel.

The *channel* must previously have had edge detection enabled with `add_event_detect()` (page 13).

Parameters
channel (*int*[43]) – The board pin number or BCM number depending on `setmode()` (page 11)

3.4 Miscellaneous

`RPi.GPIO.`**`gpio_function`**`(`*channel*`)`

Return the current GPIO function (`IN` (page 16), `OUT` (page 15), `HARD_PWM` (page 16), `SERIAL` (page 16), `I2C` (page 16), `SPI` (page 16)) for the specified *channel*.

Note: This function will only return `IN` (page 16) or `OUT` (page 15) under rpi-lgpio as the underlying kernel device cannot report the alt-mode of GPIO pins.

Parameters
channel (*int*[44]) – The board pin number or BCM number depending on `setmode()` (page 11)

`RPi.GPIO.`**`setwarnings`**`(`*value*`)`

Enable or disable warning messages. These are mostly produced when calling `setup()` (page 11) or `cleanup()` (page 12) to change channel modes.

3.5 PWM

`class` `RPi.GPIO.`**`PWM`**`(`*channel*, *frequency*`)`

Initializes and controls software-based PWM (Pulse Width Modulation) on the specified *channel* at *frequency* (in Hz).

Call `start()` (page 14) and `stop()` (page 15) to generate and stop the actual output respectively. `ChangeFrequency()` (page 14) and `ChangeDutyCycle()` (page 14) can also be used to control the output.

Note: Letting the `PWM` (page 14) object go out of scope (and be garbage collected) will implicitly stop the PWM.

`ChangeDutyCycle``(`*dc*`)`

Changes the duty cycle (percentage of the time that the pin is "on") to *dc*.

`ChangeFrequency``(`*frequency*`)`

Changes the *frequency* of rising edges output by the pin.

[41] https://docs.python.org/3/library/functions.html#int
[42] https://docs.python.org/3/library/constants.html#True
[43] https://docs.python.org/3/library/functions.html#int
[44] https://docs.python.org/3/library/functions.html#int

start(*dc*)

Starts outputting a wave on the assigned pin with a duty-cycle (which must be between 0 and 100) given by *dc*.

Parameters

dc (`float`[45]) – The duty-cycle (the percentage of time the pin is "on")

stop()

Stops outputting a wave on the assigned pin, and sets the pin's state to off.

3.6 Constants

`RPi.GPIO.`**`RPI_INFO`**

A dictionary that provides information about the model of Raspberry Pi that the library is loaded onto. Includes the following keys:

P1_REVISION
The revision of the P1 header. 0 indicates no P1 header (typical on the compute module range), 1 and 2 vary on the oldest Raspberry Pi models, and 3 is the typical 40-pin header present on all modern Raspberry Pis.

REVISION
The hex board revision code[46] as a `str`[47].

TYPE
The name of the Pi model, e.g. "Pi 4 Model B"

MANUFACTURER
The name of the board manufacturer, e.g. "Sony UK"

PROCESSOR
The name of the SoC used on the board, e.g. "BCM2711"

RAM
The amount of RAM installed on the board, e.g. "4GB"

The board revision can be overridden with the `RPI_LGPIO_REVISION` environment variable; see *Pi Revision* (page 3) for further details.

`RPi.GPIO.`**`RPI_REVISION`**

The same as the `P1_REVISION` key in *`RPI_INFO`* (page 15)

`RPi.GPIO.`**`BOARD`**

Indicates to *`setmode()`* (page 11) that physical board numbering is requested

`RPi.GPIO.`**`BCM`**

Indicates to *`setmode()`* (page 11) that GPIO numbering is requested

`RPi.GPIO.`**`PUD_OFF`**

Used with *`setup()`* (page 11) to disable internal pull resistors on an input

`RPi.GPIO.`**`PUD_DOWN`**

Used with *`setup()`* (page 11) to enable the internal pull-down resistor on an input

`RPi.GPIO.`**`PUD_UP`**

Used with *`setup()`* (page 11) to enable the internal pull-up resistor on an input

[45] https://docs.python.org/3/library/functions.html#float
[46] https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#new-style-revision-codes
[47] https://docs.python.org/3/library/stdtypes.html#str

`RPi.GPIO.`**`OUT`**

Used with *`setup()`* (page 11) to set a GPIO to an output, and *`gpio_function()`* (page 14) to report a GPIO is an output

`RPi.GPIO.`**`IN`**

Used with *`setup()`* (page 11) to set a GPIO to an input, and *`gpio_function()`* (page 14) to report a GPIO is an input

`RPi.GPIO.`**`HARD_PWM`**

`RPi.GPIO.`**`SERIAL`**

`RPi.GPIO.`**`I2C`**

`RPi.GPIO.`**`SPI`**

Used with *`gpio_function()`* (page 14) to indicate "alternate" modes of certain GPIO pins.

Note: In rpi-lgpio these values will never be returned as the kernel device cannot report if pins are in alternate modes.

`RPi.GPIO.`**`LOW = 0`**

Used with *`output()`* (page 12) to turn an output GPIO off

`RPi.GPIO.`**`HIGH = 1`**

Used with *`output()`* (page 12) to turn an output GPIO on

`RPi.GPIO.`**`RISING`**

Used with *`wait_for_edge()`* (page 13) and *`add_event_detect()`* (page 13) to specify that rising edges only should be sampled

`RPi.GPIO.`**`FALLING`**

Used with *`wait_for_edge()`* (page 13) and *`add_event_detect()`* (page 13) to specify that falling edges only should be sampled

`RPi.GPIO.`**`BOTH`**

Used with *`wait_for_edge()`* (page 13) and *`add_event_detect()`* (page 13) to specify that all edges should be sampled

CHAPTER

FOUR

CHANGELOG

4.1 Release 0.6 (2024-05-11)

- Use a smarter algorithm for determining the gpiochip device to open (#10[48])
- Add a minimum compatible version to the lgpio dependency

4.2 Release 0.5 (2024-04-12)

- Fix setting pull on GPIO2 & 3 (#8[49])
- Added some bits to the Differences chapter on determining which GPIOs are reserved
- Added more information on the supported models of Raspberry Pi (#6[50])

4.3 Release 0.4 (2023-10-03)

- Add compatibility with Raspberry Pi 5 (auto-selection of correct gpiochip device)
- Add ability to override gpiochip selection; see *GPIO Chip* (page 5)
- Convert bouncetime -666 to `None`[51] (bug compatibility, which also ensures this should work with GPIO Zero's rpigpio pin driver)
- Fix `pull_up_down` default on `setup()` (page 11)
- Fix changing `pull_up_down` of already-acquired input
- Ensure `PWM.stop()` (page 15) is idempotent

4.4 Release 0.3 (2022-10-14)

- Permit override of Pi revision code; see *Pi Revision* (page 3)
- Document alternate pin modes in *Differences* (page 3)

48 https://github.com/waveform80/rpi-lgpio/issues/10
49 https://github.com/waveform80/rpi-lgpio/pull/8
50 https://github.com/waveform80/rpi-lgpio/issues/6
51 https://docs.python.org/3/library/constants.html#None

4.5 Release 0.2 (2022-10-14)

- Add support for *RPI_REVISION* (page 15) and *RPI_INFO* (page 15) globals

4.6 Release 0.1 (2022-10-14)

- Initial release

CHAPTER

FIVE

LICENSE

[52] dave@waveform.org.uk

PYTHON MODULE INDEX

INDEX