python3-libgpiod-rpi 1.0
Functional and Technical Specification

Revision 2

Joel Savitz

June 26, 2020

1 Introduction

1.1 Purpose

This document specifies the functional requirements and and technical implementation of python3-libgpiod-rpi. Upon completion and ratification by our group, our development and quality engineering efforts will follow from this document. We will consider this document the upstream authority with respect to our implementation and as such all future changes shall be first specified here and subsequently integrated into our codebase.

1.2 Scope

This document specifies the requirements for version 1.0 of python3-libgpiod-rpi. We may release minor versions in the form of 0.x that partially satisfy this spec. This document does *not* specify features that extend the `RPi.GPIO` interface[1], in fact, we discourage extensions to the existing API in version 1.0 unless absolutely necessary.

1.3 Overview

We begin with a discussion of the problem and our proposed solution. Then, we define first the high-level functional requirements for version 1.0 and then what each API function should do and not do. Finally, we give a high-level description of the data structures and algorithms used to implement the system.

1.4 Definitions and Acronyms

A glossary of terminology and shortand used in this document can be found in table 1.

[1] With the exception of `channel_valid_or_die`(), the addition of which is basically negligible

Term	Definition
Functional Specification	A precise specification of the functional requirements that the software must conform to
The library	python3-libgpiod-rpi
Raspbian	A popular Linux distribution designed for the Raspbery Pi device by the Raspberry Pi Foundation
Raspberry Pi OS	The new name for Raspbian announced in 2020
GPIO	General Purpose Input Output, simple binary digital logic pins. For more information, search Google.
libgpiod	A generic GPIO library exposing standard Linux kernel GPIO functionality
`RPi.GPIO`	A popular python library for manipulating GPIO pins on the Raspbery Pi using obsolete kernel interfaces
python3-libgpiod	Fedora package that provides bindings for libgpiod use via `import gpiod`
gpiozero	A beginner-friendly zero-boilerplate python library to interface with GPIO devices on the Raspberry Pi
GPIO channel or pin	The number that maps to some GPIO line in terms of either of BCM or BOARD pin numbering modes
GPIO line	The object containing data needed to manipulate a physical GPIO output
GPIO event	A change in the voltage on some GPIO pin from either high to low or low to high

Table 1: Definitions and Acronyms used in this document

2 Functional Overview

This project implements a compatibility layer between `RPi.GPIO` syntax and libgpiod semantics.

Problem: `RPi.GPIO` requires non-standard kernel patches that expose the GPIO registers to userspace via a character device /dev/gpiomem [3]. As this is not supported by the mainline Linux kernel, any distribution targeting Raspberry Pi devices running the mainline kernel will not be compatible with the `RPi.GPIO` library. As a large number of tutorials, especially those targeted at beginners, demonstrate use of the RPi's GPIO pins by including `RPi.GPIO` syntax, this incompatibility limits users to distributions build on a special downstream kernel maintained by the Rapberry Pi foundation. We would like to enable beginners on any Linux distribution by allowing them to follow easily available tutorials.

Solution: Using the provided module, one will be able to write python code to use the Raspberry Pi's GPIO pins as if they were using the API implemented by `RPi.GPIO`, but instead using libgpiod's python bindings. libgpiod provides a straightforward interface for interacting with GPIO pins on supported devices via the mainline Linux kernel interface [2]. The name python3-libgpiod-rpi comes from simple concatenation of "-rpi" onto the end of the name of the Fedora package that provides the python bindings for libgpiod, python3-

libgpiod.

3 Functional Requirements for 1.0

At a high level, our list of functional requirements for 1.0 is relatively short:

- API-equivalence and feature-equivalence with `RPi.GPIO` 0.7.0
- Configurationless compatibility with gpiozero [1]

4 API Functional Specification

In this section, we define the behavior of our API by careful inspection and interpretation of the `RPi.GPIO` source code [3]. First, we define the core API functions that are accessible via the `RPi.GPIO` module. Then, we describe the Object Oriented Pulse-width Modulation interface accessible via the `RPi.GPIO.PWM` class. Finally, we describe some static data fields and debug functions.

4.1 Core `RPi.GPIO` API

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

Add a callback function to a channel previously setup for event detection by add_event_detect(). When an event is detected by the library, `callback` will be called after all previously registered callback functions are called first.

parameters:

1. channel — GPIO channel
2. callback — 0-ary callable python object

exceptions:

- RuntimeError — Channel not previously setup with add_event_detect since last Reset
- TypeError — Parameter `callback` not callable
- ValueError — Invalid channel

`RPi.GPIO`.**add_event_detect**(channel, edge, [callback], [bouncetime])

Enable detection of edge events for some GPIO channel.

parameters:

1. channel — GPIO channel
2. edge — any one of RISING, FALLING, or BOTH to specify types of events to detect
3. callback (optional) — 0-ary callable python object

   Default value: No callback function added

4. bouncetime (optional) — Cooldown time in milliseconds for callbacks

   Default value: No bouncetime enforced

exceptions:

- TypeError — Parameter `callback` not callable
- ValueError — Invalid channel, edge is not one of RISING, FALLING, or BOTH, or a negative bouncetime is specified

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

Validate `channel` using the current numbering mode.

If the channel is valid, the function returns with no effect. If the channel is invalid, a `ValueError` exception is raised.

See getmode()/setmode() in section 4.1 for more information about numbering modes.

parameters:

1. channel — GPIO channel

exceptions:

- ValueError — Invalid channel

`RPi.GPIO.`**cleanup**([channel])

Cleans up library state. Resets all GPIO lines that have been used by this program to INPUT with neither PUD_UP or PUD_DOWN set nor event detection enabled on the line. When called with no arguments, this function targets every channel, otherwise, it targets only the channels specified. Subsequent to the actions of cleanup() on a channel, one must pass it to setup() to use it again.

This function is automatically invoked with no parameters upon terminaton of a python interpreter using this module.

parameters:

1. channel (optional) — individual GPIO channel or list/tuple of GPIO channels

   Default value: All channels

exceptions:

- ValueError — Invalid channel

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

Returns True if an edge has occurred on a given GPIO at the time of call. Otherwise, returns False. This function does not block and will always return False unless add_event_detect() is called on `channel` first.

parameters:

1. channel — GPIO channel

exceptions:

- ValueError — Invalid channel

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

Get the numbering mode in use for GPIO channels. Returns one of BOARD, BCM or None

exceptions:

- RuntimeError — Module was not imported correctly

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

Get the GPIO function constant corresponding to `channel`. Returns one of IN, OUT, PWM, SERIAL, I2C, or SPI.

parameters:

1. channel — GPIO channel

exceptions:

- ValueError — Invalid channel

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

Get the current value of the GPIO line specified by `channel`. Returns one of HIGH=1 or LOW=0. Direction of `channel` must be one of INPUT or OUTPUT.

parameters:

1. channel — GPIO channel

exceptions:

- ValueError — Invalid channel
- RuntimeError — direction of `channel` not one of INPUT or OUTPUT

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

Set the value of an individual GPIO channel or a list/tuple of GPIO channels with an individual value or a list/tuple of values respectively. One may specify multiple channels and a single value for each channel to be set to the same value.

parameters:

1. channel — Individual GPIO channel or list/tuple of GPIO channels
2. value — Individual value or list/tuple of values that are one of LOW=0 or HIGH=1

exceptions:

- ValueError — Invalid channel(s) or invalid values(s)
- RuntimeError — Number of channels specified differs from number of values provided or the GPIO channel has not been setup as an output

RPi.GPIO.**remove_event_detect**(channel)

Disable and clean up edge event detection configured on a particular GPIO channel. This removes all callback functions saved for that channel.

parameters:

1. channel — GPIO channel

exceptions:

- ValueError — Invalid channel

RPi.GPIO.**setmode**(mode)

Set up numbering mode to use for channels. Mode is one of BOARD or BCM, where BOARD specifies use of Raspberry Pi board numbers and BCM specifies use of Broadcom GPIO 00..nn numbers

parameters:

1. mode — One of BOARD or BCM

exceptions:

- ValueError — Invalid mode or setmode() already called once since last call to Reset
- RuntimeError — Module was not imported correctly

RPi.GPIO.**setup**(channel, direction, [pull_up_down], [initial])

Set up an individual GPIO channel or list/tuple of GPIO channels given a direction and (optionally) a bias (e.g. PUD_UP/PUD_DOWN)

1. channel — GPIO channel
2. direction — One of IN or OUT
3. pull_up_down (optional) — Either the default value PUD_OFF or one of PUD_UP, PUD_DOWN, or PUD_DISABLE
4. initial (optional) — Initial channel value (only allowed for output)

exceptions:

- ValueError — Invalid channel(s), direction not one of IN or OUT, bias specified for OUT direction, initial pin value specified for IN direction
- RuntimeError — Module was not imported correctly

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

Enable or disable warning messages. Use True to enable and False to disable.

parameters:

1. value — A boolean

exceptions:

- RuntimeError — Module was not imported correctly

`RPi.GPIO`.**wait_for_edge**(channel, edge, [bouncetime], [timeout])

[BLOCKS] Wait for an edge event of type `edge` to take place on `channel`. Disables python's Global Interpreter Lock while blocking. A negative timeout value will cause the program to block indefinitely waiting for an edge event. A timeout value of zero will prevent the program from blocking entirely.

If an event is detected and none of the following hold true:

1. If provided a bouncetime, the difference between the current time and the time at which this event previously occurred is less than the bouncetime
2. This function, `line_event_wait` has never been called before on this channel
3. The time at which this event last occurred is later than the current time [2].

Then, the 0-ary callable objects stored in the list `_State.lines[channel].callbacks` will be serially invoked. If one subsequently invokes `event_detected(channel)`, the event detected by this function will *not* be reported.

Returns `channel` if any event is detected, otherwise returns None.

parameters:

1. channel — GPIO channel
2. edge — any one of RISING, FALLING, or BOTH to specify types of events to wait for
3. bouncetime (optional) — Cooldown time in milliseconds for callbacks

   Default value: No bouncetime enforced
4. timeout (optional) — time in milliseconds to wait before quitting

   Default value: The call will block indefinitely

exceptions:

- ValueError — Invalid channel, invalid edge, non-positive bouncetime, or non-positive timeout specified

4.2 Class `RPi.GPIO.PWM`

`RPi.GPIO.PWM` is a Pulse-width Modulation (PwM) class. Object methods follow.

`RPi.GPIO.PWM`.**ChageDutyCycle**(dutycycle)

Change the duty cycle of the PwM channel to `dutycycle`

parameters:

[2] Yeah I don't really understand this one either, for more information refer to `The RPi.GPIO 0.7.0 Source Code, source/event_gpio.c:579`[3]

1. dutycycle — percentage of pulse period to output high voltage

exceptions:

- ValueError — Invalid duty cycle

`RPi.GPIO.PWM.`**ChangeFrequency**(frequency)
Change the frequency the PwM channel to `frequency`
parameters:

1. frequency — pulse periodicity in hz

exceptions:

- ValueError — Invalid frequency value

`RPi.GPIO.PWM.`**__init__**(channel, frequency)
Initialize a new PwM object. Commonly invoked via:
`RPi.GPIO.PWM(channel, frequency)`
parameters:

1. channel — GPIO channel
2. frequency — A positive float value

exceptions:

- ValueError — Invalid frequency value
- RuntimeError — PwM already setup on `channel`

`RPi.GPIO.PWM.`**start**(dutycycle)
Start software PwM on the channel specified in `__init__`.
parameters:

1. dutycycle — A value between 0.0 and 100.0

exceptions:

- ValueError — Invalid duty cycle

`RPi.GPIO.PWM.`**stop**()
Stop software PwM on the channel specified in `__init__`.

4.3 Data

RPi.GPIO.**BCM**

Some constant representing BCM chip numbering mode, the numbering of the channels according to the Broadcom specification

RPi.GPIO.**BOARD**

Some constant representing BOARD chip numbering mode, the numbering of the channels according to the Raspberry Pi specification

RPi.GPIO.**BOTH**

Some constant representing edge event detection of both rising edge and falling edge events

RPi.GPIO.**FALLING**

Some constant representing falling edge event detection only

RPi.GPIO.**HARD_PWM**

Some constant representing that a GPIO channel is capable of supporting hardware PwM

RPi.GPIO.**HIGH**

Some nonzero constant representing a high-power state on a GPIO channel. Can be used to specify active state of a channel.

RPi.GPIO.**I2C**

Some constant representing that a GPIO channel is capable of supporting the i2c protocol

RPi.GPIO.**IN**

Some constant representing the input direction of a GPIO channel

RPi.GPIO.**LOW**

A constant zero representing a low-power state on a GPIO channel. Can be used to specify active state of a channel.

RPi.GPIO.**OUT**

Some constant representing the output direction of a GPIO channel

RPi.GPIO.**PUD_DISABLE**

Some constant representing the explicit choice to disable pull-up or pull-down resistors.

RPi.GPIO.**PUD_DOWN**

Some constant representing a pull-down resistor on a GPIO channel.

RPi.GPIO.**PUD_OFF**

Some constant representing the lack of a pull-up or pull-down resistor on a GPIO channel.

RPi.GPIO.**PUD_UP**

Some constant representing a pull-up resistor on a GPIO channel.

RPi.GPIO.**RISING**

Some constant representing rising edge event detection only

RPi.GPIO.**RPI_INFO**

A dictionary of the following key/value pair structure:

- "MANUFACTURER": Board manufacturer
- "P1_REVISION": Secondary revision number
- "PROCESSOR": CPU information
- "RAM": Total system RAM

- "REVISION": Primary revision number
- "TYPE": Board model name string

`RPi.GPIO`.**RPI_REVISION**
The major version of this Raspberry Pi device.
`RPi.GPIO`.**SERIAL**
Some constant representing that a GPIO channel is capable of supporting the serial protocol
`RPi.GPIO`.**SPI**
Some constant representing that a GPIO channel is capable of supporting the spi protocol
`RPi.GPIO`.**UNKNOWN**
Some constant representing unknown information
`RPi.GPIO`.**VERSION**
The version of this library.

4.4 Debug

What follows are a few functions that may be useful for debugging this library.
`RPi.GPIO`_DEVEL.**Reset**()
Cleanup and re-initialize the library as if it was just imported.
`RPi.GPIO`_DEVEL.**setdebuginfo**(value)
Enable or disable verbose debug messages. Use True to enable and False to disable.
parameters:

1. value — A boolean

exceptions:

- RuntimeError — Module was not imported correctly

5 High Level Technical Specification

In this section, we discuss the implementation design of python3-libgpiod-rpi. In contrast to the style of the functional specification where data and procedures were listed in alphabetical order, we will organize lists in this section in order to best reflect the underlying design.

5.1 Architecture Overview

We aim to translate `RPi.GPIO` API calls to their libgpiod equivalents. In general, the way this library modulates the user's request for an `RPi.GPIO` action is through a shared state object containing a python `gpiod.Chip` object and a list of the 54 `gpio.Line` objects. The user makes calls to our API functions which handle input validation and these functions make calls to our internal interface functions, those modifying the `Chip` object being prefixed by `chip_` and those modifying Line objects being prefixed by `line_`. These internal interface

functions modularize the library by separating API call validation from internal state changes and associated locking.

As much as possible, we try to transparently manipulate channel state as specified by the `RPi.GPIO` source code, but due to a lack of exact correspondence, we introduce line mode constants that represent the states that a particular GPIO channel can be in. We specify the meaning of these constants below.

Software Pulse-width Modulation is implemented on top of the core API. We essentially make repeated calls to `GPIO.output(channel)` corresponding appropriately to the period and duty cycle.

Event detection is done by running a poll thread using python's built-in threading class, `threading.Thread`. Since multiple active threads may try to write to the library state, we implement mutual exclusion for critical sections on a per-GPIO channel basis, with each `gpiod.Line` object having an associated lock of type `threading.lock`. These locks are intended to be used via the `begin_critical_section(channel, msg)` and `end_critical_section(channel, msg)` interface primitives. Some internal interface functions have locking and non-locking versions. This is indicated via a `_lock` suffix appended to the function identifier.

5.2 Data Design: Class _State

The top-level container for the internal state of the library. Intended to be accessed at the class level as a global shared state as if one were accessing a C struct at global scope.

_State.**mode**

One of UNKNOWN, BCM, or BOARD. Represents the current line numbering mode.

Default value: UNKNOWN

_State.**warnings**

Either True or False. Determines whether warning messages are printed to `stdout` or suppressed.

Default value: True

_State.**debuginfo**

Either True or False. Determines whether debug messages unique to this library are printed to `stdout` or suppressed.

Default value: False

_State.**chip**

The instance of class `gpiod.Chip` containing the GPIO channels provided by the `RPi.GPIO` API.

Default value: `gpiod.Chip("gpiochip0")`

_State.**event_ls**

A list of the channels on which events have recently occurred. Will be invariantly empty until event detection is setup on one or more channels.

Default value: Empty List

_State.**lines**

A list of `_Line` objects corresponding to each GPIO channel. `_Line` objects are internal to this library and contain corresponding `gpiod.Line` objects and associated data.

Default value: `[_Line(channel) for channel in range(chip_get_num_lines())]`

5.3 Data Design: Class _Line

The internal representation of a GPIO line corresponding to a particular GPIO channel.

_Line.**__init__**(channel)
Initialize a new _Line object. Invoked by `Reset`.
parameters:

1. channel — GPIO channel

_Line.**thread_start**(target_type, args)
[LOCK REQUIRED] Start a new thread on this channel to either poll for events or do pulse-width modulation. Returns True if a thread is sucessfully started and False otherwise.
parameters:

1. target_type — one of either `_line_thread_poll` or `_line_thread_pwm` to specify thread type

2. args — a tuple of arguments to pass to the entry point for the new thread

_Line.**thread_stop**()
[LOCK REQUIRED] Stop the thread running on this channel if one exists.
_Line.**cleanup**()
[LOCK REQUIRED] Reset the internal state of the object to initial state. Kills the channel's poll thread if one is running. Clears the list of callbacks. Sets line mode to `_line_mode_none`.
_Line.**mode_request**(mode, flags)
Request for `libgpiod` to execute the syscall to get permissions to access `self.channel`.
parameters:

1. mode — the desired libgpiod mode

2. flags — flags specified configuration (e.g. pull up/pull down resistors)

_Line.**channel**
The GPIO channel corresponding to this line.
Default value: An integer passed to `__init__` via `Reset()`. See `_State.lines` in section 5.2.
_Line.**line**
The `gpiod.Line` object corresponding to this channel
Default value: `gpiod.Chip.get_line(self.channel)`
_Line.**mode**
The current line mode of the GPIO channel. See section 5.7 for more on line modes.
Default value: `_line_mode_none`
_Line.**lock**
A locking primitive of type `threading.Lock`.

Used internally by the locking interface primitives `begin_critical_section(channel, msg)` and `end_critical_section(channel, msg)`.

Default value: `threading.Lock()`

_Line.**thread**

An entry of type `_LineThread` used to represent a poll thread that checks for edge events on a pin.

Default value: None

_Line.**callbacks**

A list of callable objects that are sequentially invoked upon edge event detection on `self.channel`.

Default value: Empty List

_Line.**timestamp**

A timestamp (from `time.time()`) of the last edge event detected on that channel. Used to enforce bounce timeout for callbacks on a channel.

Default value: None

5.4 Data Design: Class _LineThread

A subclass of `threading.Thread` with basic stopability.

_LineThread.**__init__**(channel, target_type, args)

Create a new instance of this class on `channel` that will call `target(*args)` when the line thread is started via `line_start_poll` or `line_start_pwm`

parameters:

1. channel — GPIO channel
2. target_type — one of either `_line_thread_poll` or `_line_thread_pwm` to specify thread type
3. args — arguments that will be passed to target at invocation

_LineThread.**kill**()

[LOCK REQUIRED] Stop a _LineThread's thread by setting the stop event and joining the thread until completion. The lock is dropped temporarily during the call to join() to allow the thread to finish its last iteration.

5.5 The `channel_` and `chip_` interfaces

This section defines the internal interface functions used to implement operations at the `gpiod.Chip` level and validation of channel numbers. External exposure of these functions is undefined behavior.

channel_**fix_and_validate**(channel)

Validate `channel` using the current numbering mode. See getmode()/setmode() in section 4.1 for more information about numbering modes.

parameters:

1. channel — GPIO channel

exceptions:

- ValueError — Invalid channel in current numbering mode

chip_**close**()
chip_**close_if_open**()

Close the file descriptor associated with the internal `gpiod.Chip` object and remove references to the object from `_State`. The former name will always attempt these operations. The latter name will perform these operations only if `_State.chip` is not None. Internally, we only call the latter except from the latter itself.

chip_**destroy**

[LOCKS ALL CHANNELS] Acquire a lock on every channel and then call `chip_close_if_open()`.

chip_**get_num_lines**()

Returns the number of GPIO lines on the chip. Used to initialize the list of `_Line` objects held by `_State`. See section 5.2 for more information.

chip_**init**()
chip_**init_if_needed**()

Initialize the `gpiod.Chip` object held by `_State.chip`. The former name performs these operations directly. The latter name performs these operations only if `_State.chip` is not None.

exceptions:

- PermissionError — Unable to open file "/dev/gpiochip0", perhaps because script was not run as root.

chip_**is_open**()

Returns the truth value of whether the chip has been initialized. True if yes, False if no.

5.6 The `line_` interface

This section defines the internal interface functions used to implement operations at the `gpiod.Line` level. External exposure of these functions is undefined behavior.

line_**add_callback**(channel, callback)

[LOCKS channel] Append a callable function to the list of callable python objects stored in `_State.lines[channel].callbacks`. Upon the next edge event detected on this channel, `callback` will be invoked after the other callback functions are invoked.

parameters:

1. channel — GPIO channel
2. callback — A 0-ary callable object

line_**do_poll**(channel, bouncetime, timeout)

[LOCKS channel] Execute the main loop of a poll thread while the thread has not been killed. Guaranteed to drop the lock for at least ten milliseconds per iteration.

parameters:

1. channel — GPIO channel

2. bouncetime — cooldown time in milliseconds for invoking the callback functions

3. timeout — time in milliseconds to wait per loop before giving up

line_**event_wait**(channel, bouncetime, timeout, track)
line_**event_wait_lock**(channel, bouncetime, track)
[first: LOCK REQUIRED, second: LOCKS channel] Wait for an edge event of type previously specified in edge event detection setup. Calls `gpiod.Line.event_wait()`. Returns the channel number if an event occurred and returns None otherwise.
parameters:

1. channel — GPIO channel

2. bouncetime — cooldown time in milliseconds for invoking the callback functions

3. timeout — time in milliseconds to wait per loop before giving up

4. track — Boolean value, the truth of which determines whether the event will subsequently be reported by `RPi.GPIO.event_detected(channel)`.

line_**get_active_state**(channel)
Returns `gpiod.Line.active_state(channel)`.
parameters:

1. channel — GPIO channel

line_**get_bias**(channel)
Returns `gpiod.Line.bias(channel)`.
parameters:

1. channel — GPIO channel

line_**get_flags**(channel)
Returns a bitwise-and of all return values of functions that get libgpiod request flags.
parameters:

1. channel — GPIO channel

line_**get_mode**(channel)
Returns `_State.lines[channel].mode`
parameters:

1. channel — GPIO channel

line_**get_unique_name**(channel)
Returns some string such that no other call to this function with a different value of channel will produce this string.
parameters:

1. channel — GPIO channel

line_**get_value**(channel)
Calls `gpiod.Line.get_value()` on `channel`.
parameters:

1. channel — GPIO channel

line_**is_active**(channel)
This functions returns a Boolean that represents the authority on whether a channel is in use by this library. A channel is defined to be active if it has line mode other than `_line_mode_none`.
parameters:

1. channel — GPIO channel

line_**is_pwm**(channel)
Returns True if a `GPIO.PWM` object has been created on `channel`, otherwise returns False.
parameters:

1. channel — GPIO channel

2. dutycycle — percentage of pulse period to output high voltage

line_**kill_poll**(channel)
line_**kill_poll_lock**(channel)
[first: LOCK REQUIRED, second: LOCKS channel] Stos poll thread running on `channel` by calling `_PollThread.kill()`. Remove the reference to that _PollThread object from the parent _Line. Do not return until the operation is complete.
parameters:

1. channel — GPIO channel

line_**poll_start**(channel, edge, callback, bouncetime)
[LOCKS channel] Kick off a new poll thread from the main thread. Creates and starts new _PollThread. Adds any specified callbacks to `_State.lines[channel].callbacks`.
parameters:

1. channel — GPIO channel

2. edge — any one of RISING, FALLING, or BOTH to specify types of events poll for

3. callback — A 0-ary callable object

4. bouncetime — cooldown time in milliseconds for invoking the callback functions

line_**pwm_set_dutycycle**(channel, dutycycle)
line_**pwm_set_dutycycl_lock**(channel, dutycycle)
[first: LOCK REQUIRED, second: LOCKS channel] Set the dutycycle value for pulse-width modulation.
parameters:

1. channel — GPIO channel

2. dutycycle — percentage of pulse period to output high voltage

line_**pwm_set_frequency**(channel, frequency)
[LOCKS channel] Set the pulse frequency value for pulse-width modulation.
parameters:

1. channel — GPIO channel

2. frequency — pulse periodicity in hz

line_**pwm_start**(channel, dutycycle)
[LOCKS channel] Start pulse-with modulation on `channel`. Returns true if successful and false otherwise.
parameters:

1. channel — GPIO channel

2. dutycycle — percentage of pulse period to output high voltage

line_**pwm_stop**(channel)
Stop a pulse-width modulation running on `channel` if it exists.
parameters:

1. channel — GPIO channel

line_**set_flags**(channel, flags)
[LOCKS channel] Set the flags on `channel` to `flags` via `gpiod.Line.set_flags()`. Used to set line bias and line active_state.
parameters:

1. channel — GPIO channel

2. flags — the new flags value to set

line_**set_mode**(channel, mode, flags)
parameters:

1. channel — GPIO channel

2. mode — A valid line mode. See section 5.7.

3. flags (optional) — the new flags value to set.
   Default value: 0

line_**set_value**(channel, value)
Calls `gpiod.Line.set_value(value)` on `channel`.
parameters:

1. channel — GPIO channel

2. value — new value for GPIO channel

line_**thread_should_die**(channel)

Return the value `_State.lines[channel].thread.killswitch.is_set()`. Used by a line thread when it considers suicide.

parameters:

1. channel — GPIO channel

5.7 Data Design: Line Modes and Line threads

The following constants are internal to the library and their external exposure is undefined. The reason for the distinction between "Values" and "Events" mode types is that the underlying `ioctl(2)` syscalls made by `libgpiod` actually request different access permissions for that GPIO channel depending on this choice.

_line_mode_none

The default line mode. This line is not accessible and should not be used in this mode.

_line_mode_in

Values input mode. This line is setup to get the value of the pin as input.

_line_mode_out

Values output mode. This line is setup to set the value of the pin as output.

_line_mode_falling

Events mode for rising edge events.

_line_mode_rising

Events mode for rising edge events.

_line_mode_both

Events mode for both rising and falling edge events. Detects all events.

_line_mode_as_is

Events as-is mode. As of yet undefined.

_line_thread_poll

A thread type to represent the lack of a thread

_line_thread_poll

A thread type that will poll for events on a channel

_line_thread_pwm

A thread type that will execute software pulse-width modulation on a channel

6 Requirements Summary for version 1.0

Functional:

- API-equivalence and feature-equivalence with `RPi.GPIO` 0.7.0 [3]
- Configurationless compatibility with gpiozero [1]

Business:

- `RPi.GPIO` API support on Fedora Linux

References

[1] gpiozero mainline repository. `https://github.com/gpiozero/gpiozero`. Accessed: 2020-05-19.

[2] libgpiod mainline repository. `https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git/`. Accessed: 2020-05-14.

[3] Rpi.gpio pypi project page. `https://pypi.org/project/RPi.GPIO/`. Accessed: 2020-05-14.