

# libgpod

## 1.6.3

Generated by Doxygen 1.12.0



<b>1 libgpiod public API</b>	<b>1</b>
<b>2 Deprecated List</b>	<b>3</b>
<b>3 Topic Index</b>	<b>5</b>
3.1 Topics . . . . .	5
<b>4 Class Index</b>	<b>7</b>
4.1 Class List . . . . .	7
<b>5 File Index</b>	<b>9</b>
5.1 File List . . . . .	9
<b>6 Topic Documentation</b>	<b>11</b>
6.1 Common helper macros . . . . .	11
6.1.1 Detailed Description . . . . .	11
6.1.2 Macro Definition Documentation . . . . .	11
6.1.2.1 GPIOD_BIT . . . . .	11
6.2 GPIO chip operations . . . . .	12
6.2.1 Detailed Description . . . . .	12
6.2.2 Function Documentation . . . . .	12
6.2.2.1 gpiod_chip_close() . . . . .	12
6.2.2.2 gpiod_chip_find_line() . . . . .	13
6.2.2.3 gpiod_chip_find_lines() . . . . .	13
6.2.2.4 gpiod_chip_get_all_lines() . . . . .	14
6.2.2.5 gpiod_chip_get_line() . . . . .	14
6.2.2.6 gpiod_chip_get_lines() . . . . .	15
6.2.2.7 gpiod_chip_label() . . . . .	15
6.2.2.8 gpiod_chip_name() . . . . .	15
6.2.2.9 gpiod_chip_num_lines() . . . . .	16
6.2.2.10 gpiod_chip_open() . . . . .	16
6.2.2.11 gpiod_chip_open_by_label() . . . . .	16
6.2.2.12 gpiod_chip_open_by_name() . . . . .	17
6.2.2.13 gpiod_chip_open_by_number() . . . . .	17
6.2.2.14 gpiod_chip_open_lookup() . . . . .	17
6.3 GPIO line operations . . . . .	18
6.3.1 Detailed Description . . . . .	18
6.3.2 Line events handling . . . . .	18
6.3.2.1 Detailed Description . . . . .	19
6.3.2.2 Enumeration Type Documentation . . . . .	19
6.3.2.3 Function Documentation . . . . .	19
6.3.3 Line info . . . . .	22
6.3.3.1 Detailed Description . . . . .	23
6.3.3.2 Enumeration Type Documentation . . . . .	23

6.3.3.3 Function Documentation . . . . .	24
6.3.4 Line requests . . . . .	28
6.3.4.1 Detailed Description . . . . .	30
6.3.4.2 Enumeration Type Documentation . . . . .	30
6.3.4.3 Function Documentation . . . . .	31
6.3.5 Misc line functions . . . . .	40
6.3.5.1 Detailed Description . . . . .	41
6.3.5.2 Function Documentation . . . . .	41
6.3.6 Operating on multiple lines . . . . .	42
6.3.6.1 Detailed Description . . . . .	43
6.3.6.2 Macro Definition Documentation . . . . .	43
6.3.6.3 Function Documentation . . . . .	44
6.3.7 Reading & setting line values . . . . .	45
6.3.7.1 Detailed Description . . . . .	46
6.3.7.2 Function Documentation . . . . .	46
6.3.8 Setting line configuration . . . . .	47
6.3.8.1 Detailed Description . . . . .	48
6.3.8.2 Function Documentation . . . . .	48
6.4 High-level API . . . . .	51
6.4.1 Detailed Description . . . . .	53
6.4.2 Typedef Documentation . . . . .	53
6.4.2.1 gpiod_ctxless_event_handle_cb . . . . .	53
6.4.2.2 gpiod_ctxless_event_poll_cb . . . . .	53
6.4.3 Enumeration Type Documentation . . . . .	53
6.4.3.1 anonymous enum . . . . .	53
6.4.3.2 anonymous enum . . . . .	54
6.4.3.3 anonymous enum . . . . .	54
6.4.3.4 anonymous enum . . . . .	54
6.4.3.5 anonymous enum . . . . .	55
6.4.4 Function Documentation . . . . .	55
6.4.4.1 gpiod_ctxless_event_loop() . . . . .	55
6.4.4.2 gpiod_ctxless_event_loop_multiple() . . . . .	56
6.4.4.3 gpiod_ctxless_event_monitor() . . . . .	57
6.4.4.4 gpiod_ctxless_event_monitor_ext() . . . . .	57
6.4.4.5 gpiod_ctxless_event_monitor_multiple() . . . . .	58
6.4.4.6 gpiod_ctxless_event_monitor_multiple_ext() . . . . .	59
6.4.4.7 gpiod_ctxless_find_line() . . . . .	60
6.4.4.8 gpiod_ctxless_get_value() . . . . .	60
6.4.4.9 gpiod_ctxless_get_value_ext() . . . . .	61
6.4.4.10 gpiod_ctxless_get_value_multiple() . . . . .	61
6.4.4.11 gpiod_ctxless_get_value_multiple_ext() . . . . .	62
6.4.4.12 gpiod_ctxless_set_value() . . . . .	62

6.4.4.13 <code>gpiod_ctxless_set_value_ext()</code> . . . . .	63
6.4.4.14 <code>gpiod_ctxless_set_value_multiple()</code> . . . . .	64
6.4.4.15 <code>gpiod_ctxless_set_value_multiple_ext()</code> . . . . .	64
6.5 Iterators for GPIO chips and lines . . . . .	65
6.5.1 Detailed Description . . . . .	65
6.5.2 Macro Definition Documentation . . . . .	66
6.5.2.1 <code>gpiod_foreach_chip</code> . . . . .	66
6.5.2.2 <code>gpiod_foreach_chip_noclose</code> . . . . .	66
6.5.2.3 <code>gpiod_foreach_line</code> . . . . .	66
6.5.3 Function Documentation . . . . .	67
6.5.3.1 <code>gpiod_chip_iter_free()</code> . . . . .	67
6.5.3.2 <code>gpiod_chip_iter_free_noclose()</code> . . . . .	67
6.5.3.3 <code>gpiod_chip_iter_new()</code> . . . . .	67
6.5.3.4 <code>gpiod_chip_iter_next()</code> . . . . .	67
6.5.3.5 <code>gpiod_chip_iter_next_noclose()</code> . . . . .	68
6.5.3.6 <code>gpiod_line_iter_free()</code> . . . . .	68
6.5.3.7 <code>gpiod_line_iter_new()</code> . . . . .	68
6.5.3.8 <code>gpiod_line_iter_next()</code> . . . . .	69
6.6 Stuff that didn't fit anywhere else . . . . .	69
6.6.1 Detailed Description . . . . .	69
6.6.2 Function Documentation . . . . .	69
6.6.2.1 <code>gpiod_version_string()</code> . . . . .	69
<b>7 Class Documentation</b> . . . . .	<b>71</b>
7.1 <code>gpiod_ctxless_event_poll_fd</code> Struct Reference . . . . .	71
7.1.1 Detailed Description . . . . .	71
7.1.2 Member Data Documentation . . . . .	71
7.1.2.1 <code>event</code> . . . . .	71
7.1.2.2 <code>fd</code> . . . . .	71
7.2 <code>gpiod_line_bulk</code> Struct Reference . . . . .	72
7.2.1 Detailed Description . . . . .	72
7.2.2 Member Data Documentation . . . . .	72
7.2.2.1 <code>lines</code> . . . . .	72
7.2.2.2 <code>num_lines</code> . . . . .	72
7.3 <code>gpiod_line_event</code> Struct Reference . . . . .	72
7.3.1 Detailed Description . . . . .	73
7.3.2 Member Data Documentation . . . . .	73
7.3.2.1 <code>event_type</code> . . . . .	73
7.3.2.2 <code>ts</code> . . . . .	73
7.4 <code>gpiod_line_request_config</code> Struct Reference . . . . .	73
7.4.1 Detailed Description . . . . .	73
7.4.2 Member Data Documentation . . . . .	73

7.4.2.1 consumer . . . . .	73
7.4.2.2 flags . . . . .	74
7.4.2.3 request_type . . . . .	74
<b>8 File Documentation</b>	<b>75</b>
8.1 gpiod.h File Reference . . . . .	75
8.2 gpiod.h . . . . .	82
<b>Index</b>	<b>89</b>

# Chapter 1

## libgpiod public API

This is the complete documentation of the public API made available to users of libgpiod.

The public header is logically split into two high-level parts: the simple API and the low-level API. The former allows users to easily interact with the GPIOs in the system without dealing with the low-level data structures and resource control. The latter gives the user much more fine-grained control over the GPIO interface.

The low-level API is further logically split into several parts such as: GPIO chip & line operators, iterators, GPIO events handling etc.

General note on error handling: all routines exported by libgpiod set `errno` to one of the error values defined in `errno.h` upon failure. The way of notifying the caller that an error occurred varies between functions, but in general a function that returns an `int`, returns `-1` on error, while a function returning a pointer bails out on error condition by returning a `NULL` pointer.





## Chapter 2

# Deprecated List

Member [gpiod\\_ctxless\\_event\\_loop](#) (const char \*device, unsigned int offset, bool active\_low, const char \*consumer, const struct timespec \*timeout, gpiod\_ctxless\_event\_poll\_cb poll\_cb, gpiod\_ctxless\_event\_handle\_cb event\_cb, void \*data) GPIO\_API GPIO\_DEPRECATED

This function suffers from an issue where HW may not allow setting up both rising and falling edge interrupts at the same time.

Member [gpiod\\_ctxless\\_event\\_loop\\_multiple](#) (const char \*device, const unsigned int \*offsets, unsigned int num\_lines, bool active\_low, const char \*consumer, const struct timespec \*timeout, gpiod\_ctxless\_event\_poll\_cb poll\_cb, gpiod\_ctxless\_event\_handle\_cb event\_cb, void \*data) GPIO\_API GPIO\_DEPRECATED

This function suffers from an issue where HW may not allow setting up both rising and falling edge interrupts at the same time.

Member [gpiod\\_line\\_needs\\_update](#) (struct gpiod\_line \*line) GPIO\_API GPIO\_DEPRECATED

This mechanism no longer exists in the library and this function does nothing.



## Chapter 3

# Topic Index

### 3.1 Topics

Here is a list of all topics with brief descriptions:

Common helper macros . . . . .	11
GPIO chip operations . . . . .	12
GPIO line operations . . . . .	18
Line events handling . . . . .	18
Line info . . . . .	22
Line requests . . . . .	28
Misc line functions . . . . .	40
Operating on multiple lines . . . . .	42
Reading & setting line values . . . . .	45
Setting line configuration . . . . .	47
High-level API . . . . .	51
Iterators for GPIO chips and lines . . . . .	65
Stuff that didn't fit anywhere else . . . . .	69



## Chapter 4

# Class Index

### 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">gpiod_ctxless_event_poll_fd</a>	
Helper structure for the ctxless event loop poll callback . . . . .	71
<a href="#">gpiod_line_bulk</a>	
Helper structure for storing a set of GPIO line objects . . . . .	72
<a href="#">gpiod_line_event</a>	
Structure holding event info . . . . .	72
<a href="#">gpiod_line_request_config</a>	
Structure holding configuration of a line request . . . . .	73



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">gpiod.h</a> . . . . .	75
-----------------------------------	----





## Chapter 6

# Topic Documentation

### 6.1 Common helper macros

#### Macros

- `#define GPIOD_API __attribute__((visibility("default")))`  
*Makes symbol visible.*
- `#define GPIOD_UNUSED __attribute__((unused))`  
*Marks a function argument or variable as potentially unused.*
- `#define GPIOD_BIT(nr)`  
*Shift 1 by given offset.*
- `#define GPIOD_DEPRECATED __attribute__((deprecated))`  
*Marks a public function as deprecated.*

#### 6.1.1 Detailed Description

Commonly used utility macros.

#### 6.1.2 Macro Definition Documentation

##### 6.1.2.1 GPIOD\_BIT

```
#define GPIOD_BIT(  
    nr  
)  
  
#include <gpiod.h>
```

#### Value:

`(1UL « (nr))`

Shift 1 by given offset.

#### Parameters

<i>nr</i>	Bit position.
-----------	---------------

#### Returns

1 shifted by nr.

## 6.2 GPIO chip operations

### Functions

- struct `gpiochip` \* `gpiochip_open` (const char \*path) [GPIO\\_API](#)  
*Open a gpiochip by path.*
- struct `gpiochip` \* `gpiochip_open_by_name` (const char \*name) [GPIO\\_API](#)  
*Open a gpiochip by name.*
- struct `gpiochip` \* `gpiochip_open_by_number` (unsigned int num) [GPIO\\_API](#)  
*Open a gpiochip by number.*
- struct `gpiochip` \* `gpiochip_open_by_label` (const char \*label) [GPIO\\_API](#)  
*Open a gpiochip by label.*
- struct `gpiochip` \* `gpiochip_open_lookup` (const char \*descr) [GPIO\\_API](#)  
*Open a gpiochip based on the best guess what the path is.*
- void `gpiochip_close` (struct `gpiochip` \*chip) [GPIO\\_API](#)  
*Close a GPIO chip handle and release all allocated resources.*
- const char \* `gpiochip_name` (struct `gpiochip` \*chip) [GPIO\\_API](#)  
*Get the GPIO chip name as represented in the kernel.*
- const char \* `gpiochip_label` (struct `gpiochip` \*chip) [GPIO\\_API](#)  
*Get the GPIO chip label as represented in the kernel.*
- unsigned int `gpiochip_num_lines` (struct `gpiochip` \*chip) [GPIO\\_API](#)  
*Get the number of GPIO lines exposed by this chip.*
- struct `gpio_line` \* `gpiochip_get_line` (struct `gpiochip` \*chip, unsigned int offset) [GPIO\\_API](#)  
*Get the handle to the GPIO line at given offset.*
- int `gpiochip_get_lines` (struct `gpiochip` \*chip, unsigned int \*offsets, unsigned int num\_offsets, struct `gpio_line_bulk` \*bulk) [GPIO\\_API](#)  
*Retrieve a set of lines and store them in a line bulk object.*
- int `gpiochip_get_all_lines` (struct `gpiochip` \*chip, struct `gpio_line_bulk` \*bulk) [GPIO\\_API](#)  
*Retrieve all lines exposed by a chip and store them in a bulk object.*
- struct `gpio_line` \* `gpiochip_find_line` (struct `gpiochip` \*chip, const char \*name) [GPIO\\_API](#)  
*Find a GPIO line by name among lines associated with given GPIO chip.*
- int `gpiochip_find_lines` (struct `gpiochip` \*chip, const char \*\*names, struct `gpio_line_bulk` \*bulk) [GPIO\\_API](#)  
*Find a set of GPIO lines by names among lines exposed by this chip.*

### 6.2.1 Detailed Description

Functions and data structures dealing with GPIO chips.

### 6.2.2 Function Documentation

#### 6.2.2.1 `gpiochip_close()`

```
void gpiochip_close (
    struct gpiochip * chip)
```

```
#include <gpio.h>
```

Close a GPIO chip handle and release all allocated resources.

**Parameters**

<i>chip</i>	The GPIO chip object.
-------------	-----------------------

**6.2.2.2 gpiod\_chip\_find\_line()**

```
struct gpiod_line * gpiod_chip_find_line (  
    struct gpiod_chip * chip,  
    const char * name)
```

```
#include <gpiod.h>
```

Find a GPIO line by name among lines associated with given GPIO chip.

**Parameters**

<i>chip</i>	The GPIO chip object.
<i>name</i>	The name of the GPIO line.

**Returns**

Pointer to the GPIO line handle or NULL if the line could not be found or an error occurred.

**Note**

In case a line with given name is not associated with given chip, the function sets `errno` to `ENOENT`.

**Attention**

GPIO line names are not unique in the linux kernel, neither globally nor within a single chip. This function finds the first line with given name.

**6.2.2.3 gpiod\_chip\_find\_lines()**

```
int gpiod_chip_find_lines (  
    struct gpiod_chip * chip,  
    const char ** names,  
    struct gpiod_line_bulk * bulk)
```

```
#include <gpiod.h>
```

Find a set of GPIO lines by names among lines exposed by this chip.

**Parameters**

<i>chip</i>	The GPIO chip object.
<i>names</i>	Array of pointers to C-strings containing the names of the lines to lookup. Must end with a NULL-pointer.
<i>bulk</i>	Line bulk object in which the located lines will be stored.

**Returns**

0 if all lines were located, -1 on error.

**Note**

If at least one line from the list could not be found among the lines exposed by this chip, the function sets `errno` to `ENOENT`.

**Attention**

GPIO line names are not unique in the linux kernel, neither globally nor within a single chip. This function finds the first line with given name.

**6.2.2.4 gpiod\_chip\_get\_all\_lines()**

```
int gpiod_chip_get_all_lines (  
    struct gpiod_chip * chip,  
    struct gpiod_line_bulk * bulk)
```

```
#include <gpiod.h>
```

Retrieve all lines exposed by a chip and store them in a bulk object.

**Parameters**

<i>chip</i>	The GPIO chip object.
<i>bulk</i>	Line bulk object in which to store the line handles.

**Returns**

0 on success, -1 on error.

**6.2.2.5 gpiod\_chip\_get\_line()**

```
struct gpiod_line * gpiod_chip_get_line (  
    struct gpiod_chip * chip,  
    unsigned int offset)
```

```
#include <gpiod.h>
```

Get the handle to the GPIO line at given offset.

**Parameters**

<i>chip</i>	The GPIO chip object.
<i>offset</i>	The offset of the GPIO line.

**Returns**

Pointer to the GPIO line handle or NULL if an error occurred.

### 6.2.2.6 `gpiod_chip_get_lines()`

```
int gpiod_chip_get_lines (
    struct gpiod_chip * chip,
    unsigned int * offsets,
    unsigned int num_offsets,
    struct gpiod_line_bulk * bulk)
```

```
#include <gpiod.h>
```

Retrieve a set of lines and store them in a line bulk object.

#### Parameters

<i>chip</i>	The GPIO chip object.
<i>offsets</i>	Array of offsets of lines to retrieve.
<i>num_offsets</i>	Number of lines to retrieve.
<i>bulk</i>	Line bulk object in which to store the line handles.

#### Returns

0 on success, -1 on error.

### 6.2.2.7 `gpiod_chip_label()`

```
const char * gpiod_chip_label (
    struct gpiod_chip * chip)
```

```
#include <gpiod.h>
```

Get the GPIO chip label as represented in the kernel.

#### Parameters

<i>chip</i>	The GPIO chip object.
-------------	-----------------------

#### Returns

Pointer to a human-readable string containing the chip label.

### 6.2.2.8 `gpiod_chip_name()`

```
const char * gpiod_chip_name (
    struct gpiod_chip * chip)
```

```
#include <gpiod.h>
```

Get the GPIO chip name as represented in the kernel.

**Parameters**

<i>chip</i>	The GPIO chip object.
-------------	-----------------------

**Returns**

Pointer to a human-readable string containing the chip name.

**6.2.2.9 gpiod\_chip\_num\_lines()**

```
unsigned int gpiod_chip_num_lines (  
    struct gpiod_chip * chip)
```

```
#include <gpiod.h>
```

Get the number of GPIO lines exposed by this chip.

**Parameters**

<i>chip</i>	The GPIO chip object.
-------------	-----------------------

**Returns**

Number of GPIO lines.

**6.2.2.10 gpiod\_chip\_open()**

```
struct gpiod_chip * gpiod_chip_open (  
    const char * path)
```

```
#include <gpiod.h>
```

Open a gpiochip by path.

**Parameters**

<i>path</i>	Path to the gpiochip device file.
-------------	-----------------------------------

**Returns**

GPIO chip handle or NULL if an error occurred.

**6.2.2.11 gpiod\_chip\_open\_by\_label()**

```
struct gpiod_chip * gpiod_chip_open_by_label (  
    const char * label)
```

```
#include <gpiod.h>
```

Open a gpiochip by label.

**Parameters**

<i>label</i>	Label of the gpiochip to open.
--------------	--------------------------------

**Returns**

GPIO chip handle or NULL if the chip with given label was not found or an error occurred.

**Note**

If the chip cannot be found but no other error occurred, `errno` is set to `ENOENT`.

**6.2.2.12 `gpiod_chip_open_by_name()`**

```
struct gpiod_chip * gpiod_chip_open_by_name (  
    const char * name)
```

```
#include <gpiod.h>
```

Open a gpiochip by name.

**Parameters**

<i>name</i>	Name of the gpiochip to open.
-------------	-------------------------------

**Returns**

GPIO chip handle or NULL if an error occurred.

This routine appends name to `/dev/` to create the path.

**6.2.2.13 `gpiod_chip_open_by_number()`**

```
struct gpiod_chip * gpiod_chip_open_by_number (  
    unsigned int num)
```

```
#include <gpiod.h>
```

Open a gpiochip by number.

**Parameters**

<i>num</i>	Number of the gpiochip.
------------	-------------------------

**Returns**

GPIO chip handle or NULL if an error occurred.

This routine appends num to `/dev/gpiochip` to create the path.

**6.2.2.14 `gpiod_chip_open_lookup()`**

```
struct gpiod_chip * gpiod_chip_open_lookup (  
    const char * descr)
```

```
#include <gpiod.h>
```

Open a gpiochip based on the best guess what the path is.

#### Parameters

<i>descr</i>	String describing the gpiochip.
--------------	---------------------------------

#### Returns

GPIO chip handle or NULL if an error occurred.

This routine tries to figure out whether the user passed it the path to the GPIO chip, its name, label or number as a string. Then it tries to open it using one of the `gpiod_chip_open**` variants.

## 6.3 GPIO line operations

#### Topics

- [Line events handling](#)
- [Line info](#)
- [Line requests](#)
- [Misc line functions](#)
- [Operating on multiple lines](#)
- [Reading & setting line values](#)
- [Setting line configuration](#)

### 6.3.1 Detailed Description

Functions and data structures dealing with GPIO lines.

### 6.3.2 Line events handling

#### Classes

- struct [gpiod\\_line\\_event](#)  
*Structure holding event info.*

#### Enumerations

- enum { [GPIOD\\_LINE\\_EVENT\\_RISING\\_EDGE](#) = 1 , [GPIOD\\_LINE\\_EVENT\\_FALLING\\_EDGE](#) }  
*Event types.*



## Functions

- int `gpiod_line_event_wait` (struct `gpiod_line` \*line, const struct `timespec` \*timeout) [GPIOD\\_API](#)  
*Wait for an event on a single line.*
- int `gpiod_line_event_wait_bulk` (struct `gpiod_line_bulk` \*bulk, const struct `timespec` \*timeout, struct `gpiod_line_bulk` \*event\_bulk) [GPIOD\\_API](#)  
*Wait for events on a set of lines.*
- int `gpiod_line_event_read` (struct `gpiod_line` \*line, struct `gpiod_line_event` \*event) [GPIOD\\_API](#)  
*Read next pending event from the GPIO line.*
- int `gpiod_line_event_read_multiple` (struct `gpiod_line` \*line, struct `gpiod_line_event` \*events, unsigned int num\_events) [GPIOD\\_API](#)  
*Read up to a certain number of events from the GPIO line.*
- int `gpiod_line_event_get_fd` (struct `gpiod_line` \*line) [GPIOD\\_API](#)  
*Get the event file descriptor.*
- int `gpiod_line_event_read_fd` (int fd, struct `gpiod_line_event` \*event) [GPIOD\\_API](#)  
*Read the last GPIO event directly from a file descriptor.*
- int `gpiod_line_event_read_fd_multiple` (int fd, struct `gpiod_line_event` \*events, unsigned int num\_events) [GPIOD\\_API](#)  
*Read up to a certain number of events directly from a file descriptor.*

### 6.3.2.1 Detailed Description

Structures and functions allowing to poll lines for events and read them, both for individual lines as well as in bulk. Also contains functions for retrieving the associated file descriptors and operate on them for easy integration with standard unix interfaces.

### 6.3.2.2 Enumeration Type Documentation

#### 6.3.2.2.1 anonymous enum

anonymous enum

```
#include <gpiod.h>
```

Event types.

Enumerator

<code>GPIOD_LINE_EVENT_RISING_EDGE</code>	Rising edge event.
<code>GPIOD_LINE_EVENT_FALLING_EDGE</code>	Falling edge event.

### 6.3.2.3 Function Documentation

#### 6.3.2.3.1 `gpiod_line_event_get_fd()`

```
int gpiod_line_event_get_fd (
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Get the event file descriptor.

#### Parameters

<i>line</i>	GPIO line object.
-------------	-------------------

#### Returns

Number of the event file descriptor or -1 if the user tries to retrieve the descriptor from a line that wasn't configured for event monitoring.

Users may want to poll the event file descriptor on their own. This routine allows to access it.

#### 6.3.2.3.2 `gpiod_line_event_read()`

```
int gpiod_line_event_read (  
    struct gpiod_line * line,  
    struct gpiod_line_event * event)
```

```
#include <gpiod.h>
```

Read next pending event from the GPIO line.

#### Parameters

<i>line</i>	GPIO line object.
<i>event</i>	Buffer to which the event data will be copied.

#### Returns

0 if the event was read correctly, -1 on error.

#### Note

This function will block if no event was queued for this line.

#### 6.3.2.3.3 `gpiod_line_event_read_fd()`

```
int gpiod_line_event_read_fd (  
    int fd,  
    struct gpiod_line_event * event)
```

```
#include <gpiod.h>
```

Read the last GPIO event directly from a file descriptor.

#### Parameters

<i>fd</i>	File descriptor.
<i>event</i>	Buffer in which the event data will be stored.

#### Returns

0 if the event was read correctly, -1 on error.

Users who directly poll the file descriptor for incoming events can also directly read the event data from it using this routine. This function translates the kernel representation of the event to the libgpiod format.

#### 6.3.2.3.4 `gpiod_line_event_read_fd_multiple()`

```
int gpiod_line_event_read_fd_multiple (
    int fd,
    struct gpiod_line_event * events,
    unsigned int num_events)
```

```
#include <gpiod.h>
```

Read up to a certain number of events directly from a file descriptor.

##### Parameters

<i>fd</i>	File descriptor.
<i>events</i>	Buffer to which the event data will be copied. Must hold at least the amount of events specified in <i>num_events</i> .
<i>num_events</i>	Specifies how many events can be stored in the buffer.

##### Returns

On success returns the number of events stored in the buffer, on failure -1 is returned.

#### 6.3.2.3.5 `gpiod_line_event_read_multiple()`

```
int gpiod_line_event_read_multiple (
    struct gpiod_line * line,
    struct gpiod_line_event * events,
    unsigned int num_events)
```

```
#include <gpiod.h>
```

Read up to a certain number of events from the GPIO line.

##### Parameters

<i>line</i>	GPIO line object.
<i>events</i>	Buffer to which the event data will be copied. Must hold at least the amount of events specified in <i>num_events</i> .
<i>num_events</i>	Specifies how many events can be stored in the buffer.

##### Returns

On success returns the number of events stored in the buffer, on failure -1 is returned.

#### 6.3.2.3.6 `gpiod_line_event_wait()`

```
int gpiod_line_event_wait (
    struct gpiod_line * line,
    const struct timespec * timeout)
```

```
#include <gpiod.h>
```

Wait for an event on a single line.

**Parameters**

<i>line</i>	GPIO line object.
<i>timeout</i>	Wait time limit.

**Returns**

0 if wait timed out, -1 if an error occurred, 1 if an event occurred.

**6.3.2.3.7 gpiod\_line\_event\_wait\_bulk()**

```
int gpiod_line_event_wait_bulk (
    struct gpiod_line_bulk * bulk,
    const struct timespec * timeout,
    struct gpiod_line_bulk * event_bulk)
```

```
#include <gpiod.h>
```

Wait for events on a set of lines.

**Parameters**

<i>bulk</i>	Set of GPIO lines to monitor.
<i>timeout</i>	Wait time limit.
<i>event_bulk</i>	Bulk object in which to store the line handles on which events occurred. Can be NULL.

**Returns**

0 if wait timed out, -1 if an error occurred, 1 if at least one event occurred.

**6.3.3 Line info****Enumerations**

- enum { `GPIOD_LINE_DIRECTION_INPUT` = 1 , `GPIOD_LINE_DIRECTION_OUTPUT` }  
*Possible direction settings.*
- enum { `GPIOD_LINE_ACTIVE_STATE_HIGH` = 1 , `GPIOD_LINE_ACTIVE_STATE_LOW` }  
*Possible active state settings.*
- enum { `GPIOD_LINE_BIAS_AS_IS` = 1 , `GPIOD_LINE_BIAS_DISABLE` , `GPIOD_LINE_BIAS_PULL_UP` , `GPIOD_LINE_BIAS_PULL_DOWN` }  
*Possible internal bias settings.*

## Functions

- unsigned int `gpiod_line_offset` (struct `gpiod_line` \*line) `GPIO_API`  
*Read the GPIO line offset.*
- const char \* `gpiod_line_name` (struct `gpiod_line` \*line) `GPIO_API`  
*Read the GPIO line name.*
- const char \* `gpiod_line_consumer` (struct `gpiod_line` \*line) `GPIO_API`  
*Read the GPIO line consumer name.*
- int `gpiod_line_direction` (struct `gpiod_line` \*line) `GPIO_API`  
*Read the GPIO line direction setting.*
- int `gpiod_line_active_state` (struct `gpiod_line` \*line) `GPIO_API`  
*Read the GPIO line active state setting.*
- int `gpiod_line_bias` (struct `gpiod_line` \*line) `GPIO_API`  
*Read the GPIO line bias setting.*
- bool `gpiod_line_is_used` (struct `gpiod_line` \*line) `GPIO_API`  
*Check if the line is currently in use.*
- bool `gpiod_line_is_open_drain` (struct `gpiod_line` \*line) `GPIO_API`  
*Check if the line is an open-drain GPIO.*
- bool `gpiod_line_is_open_source` (struct `gpiod_line` \*line) `GPIO_API`  
*Check if the line is an open-source GPIO.*
- int `gpiod_line_update` (struct `gpiod_line` \*line) `GPIO_API`  
*Re-read the line info.*
- bool `gpiod_line_needs_update` (struct `gpiod_line` \*line) `GPIO_API` `GPIO_DEPRECATED`  
*Check if the line info needs to be updated.*

### 6.3.3.1 Detailed Description

Definitions and functions for retrieving kernel information about both requested and free lines.

### 6.3.3.2 Enumeration Type Documentation

#### 6.3.3.2.1 anonymous enum

```
anonymous enum
```

```
#include <gpiod.h>
```

Possible direction settings.

Enumerator

<code>GPIO_LINE_DIRECTION_INPUT</code>	Direction is input - we're reading the state of a GPIO line.
<code>GPIO_LINE_DIRECTION_OUTPUT</code>	Direction is output - we're driving the GPIO line.

#### 6.3.3.2.2 anonymous enum

```
anonymous enum
```

```
#include <gpiod.h>
```

Possible active state settings.

**Enumerator**

GPIOD_LINE_ACTIVE_STATE_HIGH	The active state of a GPIO is active-high.
GPIOD_LINE_ACTIVE_STATE_LOW	The active state of a GPIO is active-low.

**6.3.3.2.3 anonymous enum**

anonymous enum

```
#include <gpiod.h>
```

Possible internal bias settings.

**Enumerator**

GPIOD_LINE_BIAS_AS_IS	The internal bias state is unknown.
GPIOD_LINE_BIAS_DISABLE	The internal bias is disabled.
GPIOD_LINE_BIAS_PULL_UP	The internal pull-up bias is enabled.
GPIOD_LINE_BIAS_PULL_DOWN	The internal pull-down bias is enabled.

**6.3.3.3 Function Documentation****6.3.3.3.1 gpiod\_line\_active\_state()**

```
int gpiod_line_active_state (
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Read the GPIO line active state setting.

**Parameters**

<i>line</i>	GPIO line object.
-------------	-------------------

**Returns**

Returns GPIOD\_LINE\_ACTIVE\_STATE\_HIGH or GPIOD\_LINE\_ACTIVE\_STATE\_LOW.

**6.3.3.3.2 gpiod\_line\_bias()**

```
int gpiod_line_bias (
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Read the GPIO line bias setting.

## Parameters

<i>line</i>	GPIO line object.
-------------	-------------------

## Returns

Returns `GPIO_LINE_BIAS_PULL_UP`, `GPIO_LINE_BIAS_PULL_DOWN`, `GPIO_LINE_BIAS_DISABLE` or `GPIO_LINE_BIAS_AS_IS`.

**6.3.3.3 `gpiod_line_consumer()`**

```
const char * gpiod_line_consumer (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Read the GPIO line consumer name.

## Parameters

<i>line</i>	GPIO line object.
-------------	-------------------

## Returns

Name of the GPIO consumer name as it is represented in the kernel. This routine returns a pointer to a null-terminated string or NULL if the line is not used.

**6.3.3.4 `gpiod_line_direction()`**

```
int gpiod_line_direction (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Read the GPIO line direction setting.

## Parameters

<i>line</i>	GPIO line object.
-------------	-------------------

## Returns

Returns `GPIO_LINE_DIRECTION_INPUT` or `GPIO_LINE_DIRECTION_OUTPUT`.

**6.3.3.5 `gpiod_line_is_open_drain()`**

```
bool gpiod_line_is_open_drain (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Check if the line is an open-drain GPIO.

**Parameters**

<i>line</i>	GPIO line object.
-------------	-------------------

**Returns**

True if the line is an open-drain GPIO, false otherwise.

**6.3.3.3.6 gpiod\_line\_is\_open\_source()**

```
bool gpiod_line_is_open_source (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Check if the line is an open-source GPIO.

**Parameters**

<i>line</i>	GPIO line object.
-------------	-------------------

**Returns**

True if the line is an open-source GPIO, false otherwise.

**6.3.3.3.7 gpiod\_line\_is\_used()**

```
bool gpiod_line_is_used (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Check if the line is currently in use.

**Parameters**

<i>line</i>	GPIO line object.
-------------	-------------------

**Returns**

True if the line is in use, false otherwise.

The user space can't know exactly why a line is busy. It may have been requested by another process or hogged by the kernel. It only matters that the line is used and we can't request it.

**6.3.3.3.8 gpiod\_line\_name()**

```
const char * gpiod_line_name (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Read the GPIO line name.



**Parameters**

<i>line</i>	GPIO line object.
-------------	-------------------

**Returns**

Name of the GPIO line as it is represented in the kernel. This routine returns a pointer to a null-terminated string or NULL if the line is unnamed.

**6.3.3.3.9 `gpiod_line_needs_update()`**

```
bool gpiod_line_needs_update (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Check if the line info needs to be updated.

**Parameters**

<i>line</i>	GPIO line object.
-------------	-------------------

**Returns**

Always returns false.

**Deprecated** This mechanism no longer exists in the library and this function does nothing.

**6.3.3.3.10 `gpiod_line_offset()`**

```
unsigned int gpiod_line_offset (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Read the GPIO line offset.

**Parameters**

<i>line</i>	GPIO line object.
-------------	-------------------

**Returns**

Line offset.

**6.3.3.3.11 `gpiod_line_update()`**

```
int gpiod_line_update (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Re-read the line info.

## Parameters

<i>line</i>	GPIO line object.
-------------	-------------------

## Returns

0 if the operation succeeds. In case of an error this routine returns -1 and sets the last error number.

The line info is initially retrieved from the kernel by `gpiod_chip_get_line()` and is later re-read after every successful request. Users can use this function to manually re-read the line info when needed.

We currently have no mechanism provided by the kernel for keeping the line info synchronized and for the sake of speed and simplicity of this low-level library we don't want to re-read the line info automatically everytime a property is retrieved. Any daemon using this library must track the state of lines on its own and call this routine if needed.

The state of requested lines is kept synchronized (or rather cannot be changed by external agents while the ownership of the line is taken) so there's no need to call this function in that case.

### 6.3.4 Line requests

## Classes

- struct `gpiod_line_request_config`  
*Structure holding configuration of a line request.*

## Enumerations

- enum {  
`GPIO_LINE_REQUEST_DIRECTION_AS_IS` = 1 , `GPIO_LINE_REQUEST_DIRECTION_INPUT` ,  
`GPIO_LINE_REQUEST_DIRECTION_OUTPUT` , `GPIO_LINE_REQUEST_EVENT_FALLING_EDGE`  
, `GPIO_LINE_REQUEST_EVENT_RISING_EDGE` , `GPIO_LINE_REQUEST_EVENT_BOTH_EDGES` }  
*Available types of requests.*
- enum {  
`GPIO_LINE_REQUEST_FLAG_OPEN_DRAIN` = `GPIO_BIT(0)` , `GPIO_LINE_REQUEST_FLAG_OPEN_SOURCE`  
= `GPIO_BIT(1)` , `GPIO_LINE_REQUEST_FLAG_ACTIVE_LOW` = `GPIO_BIT(2)` , `GPIO_LINE_REQUEST_FLAG_BIAS`  
= `GPIO_BIT(3)` ,  
`GPIO_LINE_REQUEST_FLAG_BIAS_PULL_DOWN` = `GPIO_BIT(4)` , `GPIO_LINE_REQUEST_FLAG_BIAS_PULL_UP`  
= `GPIO_BIT(5)` }  
*Miscellaneous GPIO request flags.*

## Functions

- `int gpiod_line_request` (struct `gpiod_line` \*line, const struct `gpiod_line_request_config` \*config, int default\_val) [GPIO\\_API](#)  
*Reserve a single line.*
- `int gpiod_line_request_input` (struct `gpiod_line` \*line, const char \*consumer) [GPIO\\_API](#)  
*Reserve a single line, set the direction to input.*
- `int gpiod_line_request_output` (struct `gpiod_line` \*line, const char \*consumer, int default\_val) [GPIO\\_API](#)  
*Reserve a single line, set the direction to output.*
- `int gpiod_line_request_rising_edge_events` (struct `gpiod_line` \*line, const char \*consumer) [GPIO\\_API](#)  
*Request rising edge event notifications on a single line.*
- `int gpiod_line_request_falling_edge_events` (struct `gpiod_line` \*line, const char \*consumer) [GPIO\\_API](#)  
*Request falling edge event notifications on a single line.*
- `int gpiod_line_request_both_edges_events` (struct `gpiod_line` \*line, const char \*consumer) [GPIO\\_API](#)  
*Request all event type notifications on a single line.*
- `int gpiod_line_request_input_flags` (struct `gpiod_line` \*line, const char \*consumer, int flags) [GPIO\\_API](#)  
*Reserve a single line, set the direction to input.*
- `int gpiod_line_request_output_flags` (struct `gpiod_line` \*line, const char \*consumer, int flags, int default\_val) [GPIO\\_API](#)  
*Reserve a single line, set the direction to output.*
- `int gpiod_line_request_rising_edge_events_flags` (struct `gpiod_line` \*line, const char \*consumer, int flags) [GPIO\\_API](#)  
*Request rising edge event notifications on a single line.*
- `int gpiod_line_request_falling_edge_events_flags` (struct `gpiod_line` \*line, const char \*consumer, int flags) [GPIO\\_API](#)  
*Request falling edge event notifications on a single line.*
- `int gpiod_line_request_both_edges_events_flags` (struct `gpiod_line` \*line, const char \*consumer, int flags) [GPIO\\_API](#)  
*Request all event type notifications on a single line.*
- `int gpiod_line_request_bulk` (struct `gpiod_line_bulk` \*bulk, const struct `gpiod_line_request_config` \*config, const int \*default\_vals) [GPIO\\_API](#)  
*Reserve a set of GPIO lines.*
- `int gpiod_line_request_bulk_input` (struct `gpiod_line_bulk` \*bulk, const char \*consumer) [GPIO\\_API](#)  
*Reserve a set of GPIO lines, set the direction to input.*
- `int gpiod_line_request_bulk_output` (struct `gpiod_line_bulk` \*bulk, const char \*consumer, const int \*default\_vals) [GPIO\\_API](#)  
*Reserve a set of GPIO lines, set the direction to output.*
- `int gpiod_line_request_bulk_rising_edge_events` (struct `gpiod_line_bulk` \*bulk, const char \*consumer) [GPIO\\_API](#)  
*Request rising edge event notifications on a set of lines.*
- `int gpiod_line_request_bulk_falling_edge_events` (struct `gpiod_line_bulk` \*bulk, const char \*consumer) [GPIO\\_API](#)  
*Request falling edge event notifications on a set of lines.*
- `int gpiod_line_request_bulk_both_edges_events` (struct `gpiod_line_bulk` \*bulk, const char \*consumer) [GPIO\\_API](#)  
*Request all event type notifications on a set of lines.*
- `int gpiod_line_request_bulk_input_flags` (struct `gpiod_line_bulk` \*bulk, const char \*consumer, int flags) [GPIO\\_API](#)  
*Reserve a set of GPIO lines, set the direction to input.*
- `int gpiod_line_request_bulk_output_flags` (struct `gpiod_line_bulk` \*bulk, const char \*consumer, int flags, const int \*default\_vals) [GPIO\\_API](#)  
*Reserve a set of GPIO lines, set the direction to output.*

- int [gpiod\\_line\\_request\\_bulk\\_rising\\_edge\\_events\\_flags](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer, int flags) [GPIO\\_API](#)  
*Request rising edge event notifications on a set of lines.*
- int [gpiod\\_line\\_request\\_bulk\\_falling\\_edge\\_events\\_flags](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer, int flags) [GPIO\\_API](#)  
*Request falling edge event notifications on a set of lines.*
- int [gpiod\\_line\\_request\\_bulk\\_both\\_edges\\_events\\_flags](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer, int flags) [GPIO\\_API](#)  
*Request all event type notifications on a set of lines.*
- void [gpiod\\_line\\_release](#) (struct [gpiod\\_line](#) \*line) [GPIO\\_API](#)  
*Release a previously reserved line.*
- void [gpiod\\_line\\_release\\_bulk](#) (struct [gpiod\\_line\\_bulk](#) \*bulk) [GPIO\\_API](#)  
*Release a set of previously reserved lines.*
- bool [gpiod\\_line\\_is\\_requested](#) (struct [gpiod\\_line](#) \*line) [GPIO\\_API](#)  
*Check if the calling user has ownership of this line.*
- bool [gpiod\\_line\\_is\\_free](#) (struct [gpiod\\_line](#) \*line) [GPIO\\_API](#)  
*Check if the calling user has neither requested ownership of this line nor configured any event notifications.*

#### 6.3.4.1 Detailed Description

Interface for requesting GPIO lines from userspace for both values and events.

#### 6.3.4.2 Enumeration Type Documentation

##### 6.3.4.2.1 anonymous enum

anonymous enum

```
#include <gpiod.h>
```

Available types of requests.

Enumerator

<a href="#">GPIOD_LINE_REQUEST_DIRECTION_AS_IS</a>	Request the line(s), but don't change current direction.
<a href="#">GPIOD_LINE_REQUEST_DIRECTION_INPUT</a>	Request the line(s) for reading the GPIO line state.
<a href="#">GPIOD_LINE_REQUEST_DIRECTION_OUTPUT</a>	Request the line(s) for setting the GPIO line state.
<a href="#">GPIOD_LINE_REQUEST_EVENT_FALLING_EDGE</a>	Only watch falling edge events.
<a href="#">GPIOD_LINE_REQUEST_EVENT_RISING_EDGE</a>	Only watch rising edge events.
<a href="#">GPIOD_LINE_REQUEST_EVENT_BOTH_EDGES</a>	Monitor both types of events.

##### 6.3.4.2.2 anonymous enum

anonymous enum

```
#include <gpiod.h>
```

Miscellaneous GPIO request flags.

## Enumerator

GPIO_LINE_REQUEST_FLAG_OPEN_DRAIN	The line is an open-drain port.
GPIO_LINE_REQUEST_FLAG_OPEN_SOURCE	The line is an open-source port.
GPIO_LINE_REQUEST_FLAG_ACTIVE_LOW	The active state of the line is low (high is the default).
GPIO_LINE_REQUEST_FLAG_BIAS_DISABLE	The line has neither either pull-up nor pull-down resistor.
GPIO_LINE_REQUEST_FLAG_BIAS_PULL_DOWN	The line has pull-down resistor enabled.
GPIO_LINE_REQUEST_FLAG_BIAS_PULL_UP	The line has pull-up resistor enabled.

## 6.3.4.3 Function Documentation

6.3.4.3.1 `gpiod_line_is_free()`

```
bool gpiod_line_is_free (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Check if the calling user has neither requested ownership of this line nor configured any event notifications.

## Parameters

<i>line</i>	GPIO line object.
-------------	-------------------

## Returns

True if given line is free, false otherwise.

6.3.4.3.2 `gpiod_line_is_requested()`

```
bool gpiod_line_is_requested (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Check if the calling user has ownership of this line.

## Parameters

<i>line</i>	GPIO line object.
-------------	-------------------

## Returns

True if given line was requested, false otherwise.

6.3.4.3.3 `gpiod_line_release()`

```
void gpiod_line_release (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Release a previously reserved line.

**Parameters**

<i>line</i>	GPIO line object.
-------------	-------------------

**6.3.4.3.4 gpiod\_line\_release\_bulk()**

```
void gpiod_line_release_bulk (
    struct gpiod_line_bulk * bulk)
```

```
#include <gpiod.h>
```

Release a set of previously reserved lines.

**Parameters**

<i>bulk</i>	Set of GPIO lines to release.
-------------	-------------------------------

If the lines were not previously requested together, the behavior is undefined.

**6.3.4.3.5 gpiod\_line\_request()**

```
int gpiod_line_request (
    struct gpiod_line * line,
    const struct gpiod_line_request_config * config,
    int default_val)
```

```
#include <gpiod.h>
```

Reserve a single line.

**Parameters**

<i>line</i>	GPIO line object.
<i>config</i>	Request options.
<i>default_val</i>	Initial line value - only relevant if we're setting the direction to output.

**Returns**

0 if the line was properly reserved. In case of an error this routine returns -1 and sets the last error number.

If this routine succeeds, the caller takes ownership of the GPIO line until it's released.

**6.3.4.3.6 gpiod\_line\_request\_both\_edges\_events()**

```
int gpiod_line_request_both_edges_events (
    struct gpiod_line * line,
    const char * consumer)
```

```
#include <gpiod.h>
```

Request all event type notifications on a single line.

## Parameters

<i>line</i>	GPIO line object.
<i>consumer</i>	Name of the consumer.

## Returns

0 if the operation succeeds, -1 on failure.

**6.3.4.3.7 `gpiod_line_request_both_edges_events_flags()`**

```
int gpiod_line_request_both_edges_events_flags (  
    struct gpiod_line * line,  
    const char * consumer,  
    int flags)
```

```
#include <gpiod.h>
```

Request all event type notifications on a single line.

## Parameters

<i>line</i>	GPIO line object.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	Additional request flags.

## Returns

0 if the operation succeeds, -1 on failure.

**6.3.4.3.8 `gpiod_line_request_bulk()`**

```
int gpiod_line_request_bulk (  
    struct gpiod_line_bulk * bulk,  
    const struct gpiod_line_request_config * config,  
    const int * default_vals)
```

```
#include <gpiod.h>
```

Reserve a set of GPIO lines.

## Parameters

<i>bulk</i>	Set of GPIO lines to reserve.
<i>config</i>	Request options.
<i>default_vals</i>	Initial line values - only relevant if we're setting the direction to output.

## Returns

0 if the all lines were properly requested. In case of an error this routine returns -1 and sets the last error number.

If this routine succeeds, the caller takes ownership of the GPIO lines until they're released. All the requested lines must be provided by the same gpiochip.

#### 6.3.4.3.9 `gpiod_line_request_bulk_both_edges_events()`

```
int gpiod_line_request_bulk_both_edges_events (  
    struct gpiod_line_bulk * bulk,  
    const char * consumer)
```

```
#include <gpiod.h>
```

Request all event type notifications on a set of lines.

##### Parameters

<i>bulk</i>	Set of GPIO lines to request.
<i>consumer</i>	Name of the consumer.

##### Returns

0 if the operation succeeds, -1 on failure.

#### 6.3.4.3.10 `gpiod_line_request_bulk_both_edges_events_flags()`

```
int gpiod_line_request_bulk_both_edges_events_flags (  
    struct gpiod_line_bulk * bulk,  
    const char * consumer,  
    int flags)
```

```
#include <gpiod.h>
```

Request all event type notifications on a set of lines.

##### Parameters

<i>bulk</i>	Set of GPIO lines to request.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	Additional request flags.

##### Returns

0 if the operation succeeds, -1 on failure.

#### 6.3.4.3.11 `gpiod_line_request_bulk_falling_edge_events()`

```
int gpiod_line_request_bulk_falling_edge_events (  
    struct gpiod_line_bulk * bulk,  
    const char * consumer)
```

```
#include <gpiod.h>
```

Request falling edge event notifications on a set of lines.



## Parameters

<i>bulk</i>	Set of GPIO lines to request.
<i>consumer</i>	Name of the consumer.

## Returns

0 if the operation succeeds, -1 on failure.

**6.3.4.3.12 `gpiod_line_request_bulk_falling_edge_events_flags()`**

```
int gpiod_line_request_bulk_falling_edge_events_flags (  
    struct gpiod_line_bulk * bulk,  
    const char * consumer,  
    int flags)
```

```
#include <gpiod.h>
```

Request falling edge event notifications on a set of lines.

## Parameters

<i>bulk</i>	Set of GPIO lines to request.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	Additional request flags.

## Returns

0 if the operation succeeds, -1 on failure.

**6.3.4.3.13 `gpiod_line_request_bulk_input()`**

```
int gpiod_line_request_bulk_input (  
    struct gpiod_line_bulk * bulk,  
    const char * consumer)
```

```
#include <gpiod.h>
```

Reserve a set of GPIO lines, set the direction to input.

## Parameters

<i>bulk</i>	Set of GPIO lines to reserve.
<i>consumer</i>	Name of the consumer.

## Returns

0 if the lines were properly reserved, -1 on failure.

**6.3.4.3.14 `gpiod_line_request_bulk_input_flags()`**

```
int gpiod_line_request_bulk_input_flags (  
    struct gpiod_line_bulk * bulk,  
    const char * consumer,  
    int flags)
```

```
#include <gpiod.h>
```

Reserve a set of GPIO lines, set the direction to input.

**Parameters**

<i>bulk</i>	Set of GPIO lines to reserve.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	Additional request flags.

**Returns**

0 if the lines were properly reserved, -1 on failure.

**6.3.4.3.15 gpiod\_line\_request\_bulk\_output()**

```
int gpiod_line_request_bulk_output (  
    struct gpiod_line_bulk * bulk,  
    const char * consumer,  
    const int * default_vals)
```

```
#include <gpiod.h>
```

Reserve a set of GPIO lines, set the direction to output.

**Parameters**

<i>bulk</i>	Set of GPIO lines to reserve.
<i>consumer</i>	Name of the consumer.
<i>default_vals</i>	Initial line values.

**Returns**

0 if the lines were properly reserved, -1 on failure.

**6.3.4.3.16 gpiod\_line\_request\_bulk\_output\_flags()**

```
int gpiod_line_request_bulk_output_flags (  
    struct gpiod_line_bulk * bulk,  
    const char * consumer,  
    int flags,  
    const int * default_vals)
```

```
#include <gpiod.h>
```

Reserve a set of GPIO lines, set the direction to output.

**Parameters**

<i>bulk</i>	Set of GPIO lines to reserve.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	Additional request flags.
<i>default_vals</i>	Initial line values.

**Returns**

0 if the lines were properly reserved, -1 on failure.

#### 6.3.4.3.17 `gpiod_line_request_bulk_rising_edge_events()`

```
int gpiod_line_request_bulk_rising_edge_events (  
    struct gpiod_line_bulk * bulk,  
    const char * consumer)
```

```
#include <gpiod.h>
```

Request rising edge event notifications on a set of lines.

##### Parameters

<i>bulk</i>	Set of GPIO lines to request.
<i>consumer</i>	Name of the consumer.

##### Returns

0 if the operation succeeds, -1 on failure.

#### 6.3.4.3.18 `gpiod_line_request_bulk_rising_edge_events_flags()`

```
int gpiod_line_request_bulk_rising_edge_events_flags (  
    struct gpiod_line_bulk * bulk,  
    const char * consumer,  
    int flags)
```

```
#include <gpiod.h>
```

Request rising edge event notifications on a set of lines.

##### Parameters

<i>bulk</i>	Set of GPIO lines to request.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	Additional request flags.

##### Returns

0 if the operation succeeds, -1 on failure.

#### 6.3.4.3.19 `gpiod_line_request_falling_edge_events()`

```
int gpiod_line_request_falling_edge_events (  
    struct gpiod_line * line,  
    const char * consumer)
```

```
#include <gpiod.h>
```

Request falling edge event notifications on a single line.

**Parameters**

<i>line</i>	GPIO line object.
<i>consumer</i>	Name of the consumer.

**Returns**

0 if the operation succeeds, -1 on failure.

**6.3.4.3.20 `gpiod_line_request_falling_edge_events_flags()`**

```
int gpiod_line_request_falling_edge_events_flags (  
    struct gpiod_line * line,  
    const char * consumer,  
    int flags)
```

```
#include <gpiod.h>
```

Request falling edge event notifications on a single line.

**Parameters**

<i>line</i>	GPIO line object.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	Additional request flags.

**Returns**

0 if the operation succeeds, -1 on failure.

**6.3.4.3.21 `gpiod_line_request_input()`**

```
int gpiod_line_request_input (  
    struct gpiod_line * line,  
    const char * consumer)
```

```
#include <gpiod.h>
```

Reserve a single line, set the direction to input.

**Parameters**

<i>line</i>	GPIO line object.
<i>consumer</i>	Name of the consumer.

**Returns**

0 if the line was properly reserved, -1 on failure.

**6.3.4.3.22 `gpiod_line_request_input_flags()`**

```
int gpiod_line_request_input_flags (  
    struct gpiod_line * line,  
    const char * consumer,  
    int flags)
```

```
#include <gpiod.h>
```

Reserve a single line, set the direction to input.

## Parameters

<i>line</i>	GPIO line object.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	Additional request flags.

## Returns

0 if the line was properly reserved, -1 on failure.

**6.3.4.3.23 gpiod\_line\_request\_output()**

```
int gpiod_line_request_output (  
    struct gpiod_line * line,  
    const char * consumer,  
    int default_val)
```

```
#include <gpiod.h>
```

Reserve a single line, set the direction to output.

## Parameters

<i>line</i>	GPIO line object.
<i>consumer</i>	Name of the consumer.
<i>default_val</i>	Initial line value.

## Returns

0 if the line was properly reserved, -1 on failure.

**6.3.4.3.24 gpiod\_line\_request\_output\_flags()**

```
int gpiod_line_request_output_flags (  
    struct gpiod_line * line,  
    const char * consumer,  
    int flags,  
    int default_val)
```

```
#include <gpiod.h>
```

Reserve a single line, set the direction to output.

## Parameters

<i>line</i>	GPIO line object.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	Additional request flags.
<i>default_val</i>	Initial line value.

## Returns

0 if the line was properly reserved, -1 on failure.

#### 6.3.4.3.25 `gpiod_line_request_rising_edge_events()`

```
int gpiod_line_request_rising_edge_events (
    struct gpiod_line * line,
    const char * consumer)
```

```
#include <gpiod.h>
```

Request rising edge event notifications on a single line.

##### Parameters

<i>line</i>	GPIO line object.
<i>consumer</i>	Name of the consumer.

##### Returns

0 if the operation succeeds, -1 on failure.

#### 6.3.4.3.26 `gpiod_line_request_rising_edge_events_flags()`

```
int gpiod_line_request_rising_edge_events_flags (
    struct gpiod_line * line,
    const char * consumer,
    int flags)
```

```
#include <gpiod.h>
```

Request rising edge event notifications on a single line.

##### Parameters

<i>line</i>	GPIO line object.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	Additional request flags.

##### Returns

0 if the operation succeeds, -1 on failure.

### 6.3.5 Misc line functions

#### Functions

- struct gpiod\_line \* [gpiod\\_line\\_get](#) (const char \*device, unsigned int offset) [GPIO API](#)  
*Get a GPIO line handle by GPIO chip description and offset.*
- struct gpiod\_line \* [gpiod\\_line\\_find](#) (const char \*name) [GPIO API](#)  
*Find a GPIO line by its name.*
- void [gpiod\\_line\\_close\\_chip](#) (struct gpiod\_line \*line) [GPIO API](#)  
*Close a GPIO chip owning this line and release all resources.*
- struct gpiod\_chip \* [gpiod\\_line\\_get\\_chip](#) (struct gpiod\_line \*line) [GPIO API](#)  
*Get the handle to the GPIO chip controlling this line.*

### 6.3.5.1 Detailed Description

Functions that didn't fit anywhere else.

### 6.3.5.2 Function Documentation

#### 6.3.5.2.1 `gpiod_line_close_chip()`

```
void gpiod_line_close_chip (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Close a GPIO chip owning this line and release all resources.

##### Parameters

<i>line</i>	GPIO line object
-------------	------------------

After this function returns, the line must no longer be used.

#### 6.3.5.2.2 `gpiod_line_find()`

```
struct gpiod_line * gpiod_line_find (  
    const char * name)
```

```
#include <gpiod.h>
```

Find a GPIO line by its name.

##### Parameters

<i>name</i>	Name of the GPIO line.
-------------	------------------------

##### Returns

Returns the GPIO line handle if the line exists in the system or NULL if it couldn't be located or an error occurred.

##### Attention

GPIO lines are not unique in the linux kernel, neither globally nor within a single chip. This function finds the first line with given name.

If this routine succeeds, the user must manually close the GPIO chip owning this line to avoid memory leaks. If the line could not be found, this functions sets `errno` to `ENOENT`.

#### 6.3.5.2.3 `gpiod_line_get()`

```
struct gpiod_line * gpiod_line_get (  
    const char * device,  
    unsigned int offset)
```

```
#include <gpiod.h>
```

Get a GPIO line handle by GPIO chip description and offset.

**Parameters**

<i>device</i>	String describing the gpiochip.
<i>offset</i>	The offset of the GPIO line.

**Returns**

GPIO line handle or NULL if an error occurred.

This routine provides a shorter alternative to calling [gpiod\\_chip\\_open\\_lookup](#) and [gpiod\\_chip\\_get\\_line](#).

If this function succeeds, the caller is responsible for closing the associated GPIO chip.

**6.3.5.2.4 gpiod\_line\_get\_chip()**

```
struct gpiod_chip * gpiod_line_get_chip (
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Get the handle to the GPIO chip controlling this line.

**Parameters**

<i>line</i>	The GPIO line object.
-------------	-----------------------

**Returns**

Pointer to the GPIO chip handle controlling this line.

**6.3.6 Operating on multiple lines****Classes**

- struct [gpiod\\_line\\_bulk](#)  
*Helper structure for storing a set of GPIO line objects.*

**Macros**

- #define **GPIOD\_LINE\_BULK\_MAX\_LINES** 64  
*Maximum number of GPIO lines that can be requested at once.*
- #define [GPIOD\\_LINE\\_BULK\\_INITIALIZER](#) { { NULL }, 0 }  
*Static initializer for GPIO bulk objects.*
- #define [gpiod\\_line\\_bulk\\_foreach\\_line](#)(bulk, line, lineptr)  
*Iterate over all line handles held by a line bulk object.*
- #define [gpiod\\_line\\_bulk\\_foreach\\_line\\_off](#)(bulk, line, offset)  
*Iterate over all line handles held by a line bulk object (integer counter variant).*



## Functions

- static void `gpiod_line_bulk_init` (struct `gpiod_line_bulk` \*bulk)  
*Initialize a GPIO bulk object.*
- static void `gpiod_line_bulk_add` (struct `gpiod_line_bulk` \*bulk, struct `gpiod_line` \*line)  
*Add a single line to a GPIO bulk object.*
- static struct `gpiod_line` \* `gpiod_line_bulk_get_line` (struct `gpiod_line_bulk` \*bulk, unsigned int offset)  
*Retrieve the line handle from a line bulk object at given offset.*
- static unsigned int `gpiod_line_bulk_num_lines` (struct `gpiod_line_bulk` \*bulk)  
*Retrieve the number of GPIO lines held by this line bulk object.*

### 6.3.6.1 Detailed Description

Convenience data structures and helper functions for storing and operating on multiple lines at once.

### 6.3.6.2 Macro Definition Documentation

#### 6.3.6.2.1 `gpiod_line_bulk_foreach_line`

```
#define gpiod_line_bulk_foreach_line(  
    bulk,  
    line,  
    lineptr)
```

```
#include <gpiod.h>
```

#### Value:

```
for ((lineptr) = (bulk)->lines, (line) = *(lineptr);  
    (lineptr) <= (bulk)->lines + ((bulk)->num_lines - 1);  
    (lineptr)++, (line) = *(lineptr))
```

Iterate over all line handles held by a line bulk object.

#### Parameters

<i>bulk</i>	Line bulk object.
<i>line</i>	GPIO line handle. On each iteration, the subsequent line handle is assigned to this pointer.
<i>lineptr</i>	Pointer to a GPIO line handle used to store the loop state.

#### 6.3.6.2.2 `gpiod_line_bulk_foreach_line_off`

```
#define gpiod_line_bulk_foreach_line_off(  
    bulk,  
    line,  
    offset)
```

```
#include <gpiod.h>
```

#### Value:

```
for ((offset) = 0, (line) = (bulk)->lines[0];  
    (offset) < (bulk)->num_lines;  
    (offset)++, (line) = (bulk)->lines[(offset)])
```

Iterate over all line handles held by a line bulk object (integer counter variant).

## Parameters

<i>bulk</i>	Line bulk object.
<i>line</i>	GPIO line handle. On each iteration, the subsequent line handle is assigned to this pointer.
<i>offset</i>	An integer variable used to store the loop state.

This is a variant of [gpiod\\_line\\_bulk\\_foreach\\_line](#) which uses an integer variable (either signed or unsigned) to store the loop state. This offset variable is guaranteed to correspond to the offset of the current line in the `bulk->lines` array.

### 6.3.6.2.3 GPIOD\_LINE\_BULK\_INITIALIZER

```
#define GPIOD_LINE_BULK_INITIALIZER { { NULL }, 0 }
```

```
#include <gpiod.h>
```

Static initializer for GPIO bulk objects.

This macro simply sets the internally held number of lines to 0.

### 6.3.6.3 Function Documentation

#### 6.3.6.3.1 gpiod\_line\_bulk\_add()

```
static void gpiod_line_bulk_add (
    struct gpiod_line_bulk * bulk,
    struct gpiod_line * line) [inline], [static]
```

```
#include <gpiod.h>
```

Add a single line to a GPIO bulk object.

## Parameters

<i>bulk</i>	Line bulk object.
<i>line</i>	Line to add.

#### 6.3.6.3.2 gpiod\_line\_bulk\_get\_line()

```
static struct gpiod_line * gpiod_line_bulk_get_line (
    struct gpiod_line_bulk * bulk,
    unsigned int offset) [inline], [static]
```

```
#include <gpiod.h>
```

Retrieve the line handle from a line bulk object at given offset.

## Parameters

<i>bulk</i>	Line bulk object.
<i>offset</i>	Line offset.

## Returns

Line handle at given offset.

**6.3.6.3.3 `gpiod_line_bulk_init()`**

```
static void gpiod_line_bulk_init (  
    struct gpiod\_line\_bulk * bulk)    [inline], [static]
```

```
#include <gpiod.h>
```

Initialize a GPIO bulk object.

## Parameters

<i>bulk</i>	Line bulk object.
-------------	-------------------

This routine simply sets the internally held number of lines to 0.

**6.3.6.3.4 `gpiod_line_bulk_num_lines()`**

```
static unsigned int gpiod_line_bulk_num_lines (  
    struct gpiod\_line\_bulk * bulk)    [inline], [static]
```

```
#include <gpiod.h>
```

Retrieve the number of GPIO lines held by this line bulk object.

## Parameters

<i>bulk</i>	Line bulk object.
-------------	-------------------

## Returns

Number of lines held by this line bulk.

**6.3.7 Reading & setting line values****Functions**

- int [gpiod\\_line\\_get\\_value](#) (struct [gpiod\\_line](#) \**line*) [GPIOD\\_API](#)  
*Read current value of a single GPIO line.*
- int [gpiod\\_line\\_get\\_value\\_bulk](#) (struct [gpiod\\_line\\_bulk](#) \**bulk*, int \**values*) [GPIOD\\_API](#)  
*Read current values of a set of GPIO lines.*
- int [gpiod\\_line\\_set\\_value](#) (struct [gpiod\\_line](#) \**line*, int *value*) [GPIOD\\_API](#)  
*Set the value of a single GPIO line.*
- int [gpiod\\_line\\_set\\_value\\_bulk](#) (struct [gpiod\\_line\\_bulk](#) \**bulk*, const int \**values*) [GPIOD\\_API](#)  
*Set the values of a set of GPIO lines.*

### 6.3.7.1 Detailed Description

Functions allowing to read and set GPIO line values for single lines and in bulk.

### 6.3.7.2 Function Documentation

#### 6.3.7.2.1 `gpiod_line_get_value()`

```
int gpiod_line_get_value (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Read current value of a single GPIO line.

##### Parameters

<i>line</i>	GPIO line object.
-------------	-------------------

##### Returns

0 or 1 if the operation succeeds. On error this routine returns -1 and sets the last error number.

#### 6.3.7.2.2 `gpiod_line_get_value_bulk()`

```
int gpiod_line_get_value_bulk (  
    struct gpiod_line_bulk * bulk,  
    int * values)
```

```
#include <gpiod.h>
```

Read current values of a set of GPIO lines.

##### Parameters

<i>bulk</i>	Set of GPIO lines to reserve.
<i>values</i>	An array big enough to hold <code>line_bulk-&gt;num_lines</code> values.

##### Returns

0 is the operation succeeds. In case of an error this routine returns -1 and sets the last error number.

If succeeds, this routine fills the values array with a set of values in the same order, the lines are added to `line_bulk`. If the lines were not previously requested together, the behavior is undefined.

#### 6.3.7.2.3 `gpiod_line_set_value()`

```
int gpiod_line_set_value (  
    struct gpiod_line * line,  
    int value)
```

```
#include <gpiod.h>
```

Set the value of a single GPIO line.

## Parameters

<i>line</i>	GPIO line object.
<i>value</i>	New value.

## Returns

0 is the operation succeeds. In case of an error this routine returns -1 and sets the last error number.

6.3.7.2.4 `gpiod_line_set_value_bulk()`

```
int gpiod_line_set_value_bulk (
    struct gpiod_line_bulk * bulk,
    const int * values)
```

```
#include <gpiod.h>
```

Set the values of a set of GPIO lines.

## Parameters

<i>bulk</i>	Set of GPIO lines to reserve.
<i>values</i>	An array holding <code>line_bulk-&gt;num_lines</code> new values for lines. A NULL pointer is interpreted as a logical low for all lines.

## Returns

0 is the operation succeeds. In case of an error this routine returns -1 and sets the last error number.

If the lines were not previously requested together, the behavior is undefined.

## 6.3.8 Setting line configuration

## Functions

- int `gpiod_line_set_config` (struct `gpiod_line` \**line*, int *direction*, int *flags*, int *value*) [GPIO\\_API](#)  
*Update the configuration of a single GPIO line.*
- int `gpiod_line_set_config_bulk` (struct `gpiod_line_bulk` \**bulk*, int *direction*, int *flags*, const int \**values*) [GPIO\\_API](#)  
*Update the configuration of a set of GPIO lines.*
- int `gpiod_line_set_flags` (struct `gpiod_line` \**line*, int *flags*) [GPIO\\_API](#)  
*Update the configuration flags of a single GPIO line.*
- int `gpiod_line_set_flags_bulk` (struct `gpiod_line_bulk` \**bulk*, int *flags*) [GPIO\\_API](#)  
*Update the configuration flags of a set of GPIO lines.*
- int `gpiod_line_set_direction_input` (struct `gpiod_line` \**line*) [GPIO\\_API](#)  
*Set the direction of a single GPIO line to input.*
- int `gpiod_line_set_direction_input_bulk` (struct `gpiod_line_bulk` \**bulk*) [GPIO\\_API](#)  
*Set the direction of a set of GPIO lines to input.*
- int `gpiod_line_set_direction_output` (struct `gpiod_line` \**line*, int *value*) [GPIO\\_API](#)  
*Set the direction of a single GPIO line to output.*
- int `gpiod_line_set_direction_output_bulk` (struct `gpiod_line_bulk` \**bulk*, const int \**values*) [GPIO\\_API](#)  
*Set the direction of a set of GPIO lines to output.*

### 6.3.8.1 Detailed Description

Functions allowing modification of config options of GPIO lines requested from user-space.

### 6.3.8.2 Function Documentation

#### 6.3.8.2.1 `gpiod_line_set_config()`

```
int gpiod_line_set_config (
    struct gpiod_line * line,
    int direction,
    int flags,
    int value)
```

```
#include <gpiod.h>
```

Update the configuration of a single GPIO line.

##### Parameters

<i>line</i>	GPIO line object.
<i>direction</i>	Updated direction which may be one of <code>GPIO_LINE_REQUEST_DIRECTION_AS_IS</code> , <code>GPIO_LINE_REQUEST_DIRECTION_INPUT</code> , or <code>GPIO_LINE_REQUEST_DIRECTION_OUTPUT</code> .
<i>flags</i>	Replacement flags.
<i>value</i>	The new output value for the line when direction is <code>GPIO_LINE_REQUEST_DIRECTION_OUTPUT</code> .

##### Returns

0 is the operation succeeds. In case of an error this routine returns -1 and sets the last error number.

#### 6.3.8.2.2 `gpiod_line_set_config_bulk()`

```
int gpiod_line_set_config_bulk (
    struct gpiod_line_bulk * bulk,
    int direction,
    int flags,
    const int * values)
```

```
#include <gpiod.h>
```

Update the configuration of a set of GPIO lines.

##### Parameters

<i>bulk</i>	Set of GPIO lines.
<i>direction</i>	Updated direction which may be one of <code>GPIO_LINE_REQUEST_DIRECTION_AS_IS</code> , <code>GPIO_LINE_REQUEST_DIRECTION_INPUT</code> , or <code>GPIO_LINE_REQUEST_DIRECTION_OUTPUT</code> .
<i>flags</i>	Replacement flags.
<i>values</i>	An array holding <code>line_bulk-&gt;num_lines</code> new logical values for lines when direction is <code>GPIO_LINE_REQUEST_DIRECTION_OUTPUT</code> . A NULL pointer is interpreted as a logical low for all lines.

##### Returns

0 is the operation succeeds. In case of an error this routine returns -1 and sets the last error number.

If the lines were not previously requested together, the behavior is undefined.

#### 6.3.8.2.3 `gpiod_line_set_direction_input()`

```
int gpiod_line_set_direction_input (  
    struct gpiod_line * line)
```

```
#include <gpiod.h>
```

Set the direction of a single GPIO line to input.

##### Parameters

<i>line</i>	GPIO line object.
-------------	-------------------

##### Returns

0 is the operation succeeds. In case of an error this routine returns -1 and sets the last error number.

#### 6.3.8.2.4 `gpiod_line_set_direction_input_bulk()`

```
int gpiod_line_set_direction_input_bulk (  
    struct gpiod_line_bulk * bulk)
```

```
#include <gpiod.h>
```

Set the direction of a set of GPIO lines to input.

##### Parameters

<i>bulk</i>	Set of GPIO lines.
-------------	--------------------

##### Returns

0 is the operation succeeds. In case of an error this routine returns -1 and sets the last error number.

If the lines were not previously requested together, the behavior is undefined.

#### 6.3.8.2.5 `gpiod_line_set_direction_output()`

```
int gpiod_line_set_direction_output (  
    struct gpiod_line * line,  
    int value)
```

```
#include <gpiod.h>
```

Set the direction of a single GPIO line to output.

##### Parameters

<i>line</i>	GPIO line object.
<i>value</i>	The logical value output on the line.

##### Returns

0 is the operation succeeds. In case of an error this routine returns -1 and sets the last error number.

#### 6.3.8.2.6 `gpiod_line_set_direction_output_bulk()`

```
int gpiod_line_set_direction_output_bulk (  
    struct gpiod_line_bulk * bulk,  
    const int * values)
```

```
#include <gpiod.h>
```

Set the direction of a set of GPIO lines to output.

##### Parameters

<i>bulk</i>	Set of GPIO lines.
<i>values</i>	An array holding <code>line_bulk-&gt;num_lines</code> new logical values for lines. A NULL pointer is interpreted as a logical low for all lines.

##### Returns

0 is the operation succeeds. In case of an error this routine returns -1 and sets the last error number.

If the lines were not previously requested together, the behavior is undefined.

#### 6.3.8.2.7 `gpiod_line_set_flags()`

```
int gpiod_line_set_flags (  
    struct gpiod_line * line,  
    int flags)
```

```
#include <gpiod.h>
```

Update the configuration flags of a single GPIO line.

##### Parameters

<i>line</i>	GPIO line object.
<i>flags</i>	Replacement flags.

##### Returns

0 is the operation succeeds. In case of an error this routine returns -1 and sets the last error number.

#### 6.3.8.2.8 `gpiod_line_set_flags_bulk()`

```
int gpiod_line_set_flags_bulk (  
    struct gpiod_line_bulk * bulk,  
    int flags)
```

```
#include <gpiod.h>
```

Update the configuration flags of a set of GPIO lines.



## Parameters

<i>bulk</i>	Set of GPIO lines.
<i>flags</i>	Replacement flags.

## Returns

0 is the operation succeeds. In case of an error this routine returns -1 and sets the last error number.

If the lines were not previously requested together, the behavior is undefined.

## 6.4 High-level API

## Classes

- struct [gpiod\\_ctxless\\_event\\_poll\\_fd](#)  
*Helper structure for the ctxless event loop poll callback.*

## Typedefs

- typedef void(\* [gpiod\\_ctxless\\_set\\_value\\_cb](#)) (void \*)  
*Simple set value callback signature.*
- typedef int(\* [gpiod\\_ctxless\\_event\\_handle\\_cb](#)) (int, unsigned int, const struct timespec \*, void \*)  
*Simple event callback signature.*
- typedef int(\* [gpiod\\_ctxless\\_event\\_poll\\_cb](#)) (unsigned int, struct [gpiod\\_ctxless\\_event\\_poll\\_fd](#) \*, const struct timespec \*, void \*)  
*Simple event poll callback signature.*

## Enumerations

- enum {  
  [GPIOD\\_CTXLESS\\_FLAG\\_OPEN\\_DRAIN](#) = [GPIOD\\_BIT](#)(0) , [GPIOD\\_CTXLESS\\_FLAG\\_OPEN\\_SOURCE](#) =  
  [GPIOD\\_BIT](#)(1) , [GPIOD\\_CTXLESS\\_FLAG\\_BIAS\\_DISABLE](#) = [GPIOD\\_BIT](#)(2) , [GPIOD\\_CTXLESS\\_FLAG\\_BIAS\\_PULL\\_DOWN](#)  
  = [GPIOD\\_BIT](#)(3) ,  
  [GPIOD\\_CTXLESS\\_FLAG\\_BIAS\\_PULL\\_UP](#) = [GPIOD\\_BIT](#)(4) }  
*Miscellaneous GPIO flags.*
- enum { [GPIOD\\_CTXLESS\\_EVENT\\_RISING\\_EDGE](#) = 1 , [GPIOD\\_CTXLESS\\_EVENT\\_FALLING\\_EDGE](#) ,  
  [GPIOD\\_CTXLESS\\_EVENT\\_BOTH\\_EDGES](#) }  
*Event types that the ctxless event monitor can wait for.*
- enum { [GPIOD\\_CTXLESS\\_EVENT\\_CB\\_TIMEOUT](#) = 1 , [GPIOD\\_CTXLESS\\_EVENT\\_CB\\_RISING\\_EDGE](#) ,  
  [GPIOD\\_CTXLESS\\_EVENT\\_CB\\_FALLING\\_EDGE](#) }  
*Event types that can be passed to the ctxless event callback.*
- enum { [GPIOD\\_CTXLESS\\_EVENT\\_CB\\_RET\\_ERR](#) = -1 , [GPIOD\\_CTXLESS\\_EVENT\\_CB\\_RET\\_OK](#) = 0 ,  
  [GPIOD\\_CTXLESS\\_EVENT\\_CB\\_RET\\_STOP](#) = 1 }  
*Return status values that the ctxless event callback can return.*
- enum { [GPIOD\\_CTXLESS\\_EVENT\\_POLL\\_RET\\_STOP](#) = -2 , [GPIOD\\_CTXLESS\\_EVENT\\_POLL\\_RET\\_ERR](#)  
  = -1 , [GPIOD\\_CTXLESS\\_EVENT\\_POLL\\_RET\\_TIMEOUT](#) = 0 }  
*Return status values that the ctxless event poll callback can return.*

## Functions

- `int gpiod_ctxless_get_value` (const char \*device, unsigned int offset, bool active\_low, const char \*consumer) [GPIO\\_API](#)  
*Read current value from a single GPIO line.*
- `int gpiod_ctxless_get_value_ext` (const char \*device, unsigned int offset, bool active\_low, const char \*consumer, int flags) [GPIO\\_API](#)  
*Read current value from a single GPIO line.*
- `int gpiod_ctxless_get_value_multiple` (const char \*device, const unsigned int \*offsets, int \*values, unsigned int num\_lines, bool active\_low, const char \*consumer) [GPIO\\_API](#)  
*Read current values from a set of GPIO lines.*
- `int gpiod_ctxless_get_value_multiple_ext` (const char \*device, const unsigned int \*offsets, int \*values, unsigned int num\_lines, bool active\_low, const char \*consumer, int flags) [GPIO\\_API](#)  
*Read current values from a set of GPIO lines.*
- `int gpiod_ctxless_set_value` (const char \*device, unsigned int offset, int value, bool active\_low, const char \*consumer, [gpiod\\_ctxless\\_set\\_value\\_cb](#) cb, void \*data) [GPIO\\_API](#)  
*Set value of a single GPIO line.*
- `int gpiod_ctxless_set_value_ext` (const char \*device, unsigned int offset, int value, bool active\_low, const char \*consumer, [gpiod\\_ctxless\\_set\\_value\\_cb](#) cb, void \*data, int flags) [GPIO\\_API](#)  
*Set value of a single GPIO line.*
- `int gpiod_ctxless_set_value_multiple` (const char \*device, const unsigned int \*offsets, const int \*values, unsigned int num\_lines, bool active\_low, const char \*consumer, [gpiod\\_ctxless\\_set\\_value\\_cb](#) cb, void \*data) [GPIO\\_API](#)  
*Set values of multiple GPIO lines.*
- `int gpiod_ctxless_set_value_multiple_ext` (const char \*device, const unsigned int \*offsets, const int \*values, unsigned int num\_lines, bool active\_low, const char \*consumer, [gpiod\\_ctxless\\_set\\_value\\_cb](#) cb, void \*data, int flags) [GPIO\\_API](#)  
*Set values of multiple GPIO lines.*
- `int gpiod_ctxless_event_loop` (const char \*device, unsigned int offset, bool active\_low, const char \*consumer, const struct timespec \*timeout, [gpiod\\_ctxless\\_event\\_poll\\_cb](#) poll\_cb, [gpiod\\_ctxless\\_event\\_handle\\_cb](#) event\_cb, void \*data) [GPIO\\_API](#) [GPIO\\_DEPRECATED](#)  
*Wait for events on a single GPIO line.*
- `int gpiod_ctxless_event_loop_multiple` (const char \*device, const unsigned int \*offsets, unsigned int num\_lines, bool active\_low, const char \*consumer, const struct timespec \*timeout, [gpiod\\_ctxless\\_event\\_poll\\_cb](#) poll\_cb, [gpiod\\_ctxless\\_event\\_handle\\_cb](#) event\_cb, void \*data) [GPIO\\_API](#) [GPIO\\_DEPRECATED](#)  
*Wait for events on multiple GPIO lines.*
- `int gpiod_ctxless_event_monitor` (const char \*device, int event\_type, unsigned int offset, bool active\_low, const char \*consumer, const struct timespec \*timeout, [gpiod\\_ctxless\\_event\\_poll\\_cb](#) poll\_cb, [gpiod\\_ctxless\\_event\\_handle\\_cb](#) event\_cb, void \*data) [GPIO\\_API](#)  
*Wait for events on a single GPIO line.*
- `int gpiod_ctxless_event_monitor_ext` (const char \*device, int event\_type, unsigned int offset, bool active\_low, const char \*consumer, const struct timespec \*timeout, [gpiod\\_ctxless\\_event\\_poll\\_cb](#) poll\_cb, [gpiod\\_ctxless\\_event\\_handle\\_cb](#) event\_cb, void \*data, int flags) [GPIO\\_API](#)  
*Wait for events on a single GPIO line.*
- `int gpiod_ctxless_event_monitor_multiple` (const char \*device, int event\_type, const unsigned int \*offsets, unsigned int num\_lines, bool active\_low, const char \*consumer, const struct timespec \*timeout, [gpiod\\_ctxless\\_event\\_poll\\_cb](#) poll\_cb, [gpiod\\_ctxless\\_event\\_handle\\_cb](#) event\_cb, void \*data) [GPIO\\_API](#)  
*Wait for events on multiple GPIO lines.*
- `int gpiod_ctxless_event_monitor_multiple_ext` (const char \*device, int event\_type, const unsigned int \*offsets, unsigned int num\_lines, bool active\_low, const char \*consumer, const struct timespec \*timeout, [gpiod\\_ctxless\\_event\\_poll\\_cb](#) poll\_cb, [gpiod\\_ctxless\\_event\\_handle\\_cb](#) event\_cb, void \*data, int flags) [GPIO\\_API](#)  
*Wait for events on multiple GPIO lines.*
- `int gpiod_ctxless_find_line` (const char \*name, char \*chipname, size\_t chipname\_size, unsigned int \*offset) [GPIO\\_API](#)  
*Determine the chip name and line offset of a line with given name.*

### 6.4.1 Detailed Description

Simple high-level routines for straightforward GPIO manipulation without the need to use the `gpiod_*` structures or to keep track of resources.

### 6.4.2 Typedef Documentation

#### 6.4.2.1 `gpiod_ctxless_event_handle_cb`

```
typedef int(* gpiod_ctxless_event_handle_cb) (int, unsigned int, const struct timespec *, void *)
```

```
#include <gpiod.h>
```

Simple event callback signature.

The callback function takes the following arguments: event type (int), GPIO line offset (unsigned int), event timestamp (const struct timespec \*) and a pointer to user data (void \*).

This callback is called by the ctxless event loop functions for each GPIO event. If the callback returns `GPIOD_CTXLESS_EVENT_CB_RET_ERR`, it should also set `errno`.

#### 6.4.2.2 `gpiod_ctxless_event_poll_cb`

```
typedef int(* gpiod_ctxless_event_poll_cb) (unsigned int, struct gpiod_ctxless_event_poll_fd *, const struct timespec *, void *)
```

```
#include <gpiod.h>
```

Simple event poll callback signature.

The poll callback function takes the following arguments: number of lines (unsigned int), an array of file descriptors on which input events should be monitored (struct `gpiod_ctxless_event_poll_fd` \*), poll timeout (const struct timespec \*) and a pointer to user data (void \*).

The callback should poll for input events on the set of descriptors and return an appropriate value that can be interpreted by the event loop routine.

### 6.4.3 Enumeration Type Documentation

#### 6.4.3.1 anonymous enum

```
anonymous enum
```

```
#include <gpiod.h>
```

Miscellaneous GPIO flags.

**Enumerator**

GPIOD_CTXLESS_FLAG_OPEN_DRAIN	The line is an open-drain port.
GPIOD_CTXLESS_FLAG_OPEN_SOURCE	The line is an open-source port.
GPIOD_CTXLESS_FLAG_BIAS_DISABLE	The line has neither either pull-up nor pull-down resistor
GPIOD_CTXLESS_FLAG_BIAS_PULL_DOWN	The line has pull-down resistor enabled
GPIOD_CTXLESS_FLAG_BIAS_PULL_UP	The line has pull-up resistor enabled

**6.4.3.2 anonymous enum**

```
anonymous enum
```

```
#include <gpio.h>
```

Event types that the ctxless event monitor can wait for.

**Enumerator**

GPIOD_CTXLESS_EVENT_RISING_EDGE	Wait for rising edge events only. Wait for falling edge events only.
GPIOD_CTXLESS_EVENT_FALLING_EDGE	Wait for both types of events.

**6.4.3.3 anonymous enum**

```
anonymous enum
```

```
#include <gpio.h>
```

Event types that can be passed to the ctxless event callback.

**Enumerator**

GPIOD_CTXLESS_EVENT_CB_TIMEOUT	Waiting for events timed out.
GPIOD_CTXLESS_EVENT_CB_RISING_EDGE	Rising edge event occurred.
GPIOD_CTXLESS_EVENT_CB_FALLING_EDGE	Falling edge event occurred.

**6.4.3.4 anonymous enum**

```
anonymous enum
```

```
#include <gpio.h>
```

Return status values that the ctxless event callback can return.

**Enumerator**

GPIOD_CTXLESS_EVENT_CB_RET_ERR	Stop processing events and indicate an error.
GPIOD_CTXLESS_EVENT_CB_RET_OK	Continue processing events.
GPIOD_CTXLESS_EVENT_CB_RET_STOP	Stop processing events.

### 6.4.3.5 anonymous enum

anonymous enum

```
#include <gpio.h>
```

Return status values that the ctxless event poll callback can return.

Positive value returned from the polling callback indicates the number of events that occurred on the set of monitored lines.

Enumerator

GPIOD_CTXLESS_EVENT_POLL_RET_STOP	The event loop should stop processing events.
GPIOD_CTXLESS_EVENT_POLL_RET_ERR	Polling error occurred (the polling function should set errno).
GPIOD_CTXLESS_EVENT_POLL_RET_TIMEOUT	Poll timed out.

## 6.4.4 Function Documentation

### 6.4.4.1 gpio\_ctxless\_event\_loop()

```
int gpio_ctxless_event_loop (
    const char * device,
    unsigned int offset,
    bool active_low,
    const char * consumer,
    const struct timespec * timeout,
    gpio_ctxless_event_poll_cb poll_cb,
    gpio_ctxless_event_handle_cb event_cb,
    void * data)
```

```
#include <gpio.h>
```

Wait for events on a single GPIO line.

Parameters

<i>device</i>	Name, path, number or label of the gpiochip.
<i>offset</i>	GPIO line offset to monitor.
<i>active_low</i>	The active state of this line - true if low.
<i>consumer</i>	Name of the consumer.
<i>timeout</i>	Maximum wait time for each iteration.
<i>poll_cb</i>	Callback function to call when waiting for events.
<i>event_cb</i>	Callback function to call for each line event.
<i>data</i>	User data passed to the callback.

Returns

0 if no errors were encountered, -1 if an error occurred.

Note

The way the ctxless event loop works is described in detail in [gpio\\_ctxless\\_event\\_loop\\_multiple](#) - this is just a wrapper around this routine which calls it for a single GPIO line.

**Deprecated** This function suffers from an issue where HW may not allow setting up both rising and falling edge interrupts at the same time.

#### 6.4.4.2 `gpiod_ctxless_event_loop_multiple()`

```
int gpiod_ctxless_event_loop_multiple (
    const char * device,
    const unsigned int * offsets,
    unsigned int num_lines,
    bool active_low,
    const char * consumer,
    const struct timespec * timeout,
    gpiod_ctxless_event_poll_cb poll_cb,
    gpiod_ctxless_event_handle_cb event_cb,
    void * data)
```

```
#include <gpiod.h>
```

Wait for events on multiple GPIO lines.

##### Parameters

<i>device</i>	Name, path, number or label of the gpiochip.
<i>offsets</i>	Array of GPIO line offsets to monitor.
<i>num_lines</i>	Number of lines to monitor.
<i>active_low</i>	The active state of this line - true if low.
<i>consumer</i>	Name of the consumer.
<i>timeout</i>	Maximum wait time for each iteration.
<i>poll_cb</i>	Callback function to call when waiting for events. Can be NULL.
<i>event_cb</i>	Callback function to call on event occurrence.
<i>data</i>	User data passed to the callback.

##### Returns

0 no errors were encountered, -1 if an error occurred.

##### Note

The poll callback can be NULL in which case the routine will fall back to a basic, `ppoll()` based callback.

**Deprecated** This function suffers from an issue where HW may not allow setting up both rising and falling edge interrupts at the same time.

Internally this routine opens the GPIO chip, requests the set of lines for both-edges events and calls the polling callback in a loop. The role of the polling callback is to detect input events on a set of file descriptors and notify the caller about the fds ready for reading.

The ctxless event loop then reads each queued event from marked descriptors and calls the event callback. Both callbacks can stop the loop at any point.

The `poll_cb` argument can be NULL in which case the function falls back to a default, `ppoll()` based callback.

#### 6.4.4.3 `gpiod_ctxless_event_monitor()`

```
int gpiod_ctxless_event_monitor (
    const char * device,
    int event_type,
    unsigned int offset,
    bool active_low,
    const char * consumer,
    const struct timespec * timeout,
    gpiod_ctxless_event_poll_cb poll_cb,
    gpiod_ctxless_event_handle_cb event_cb,
    void * data)

#include <gpiod.h>
```

Wait for events on a single GPIO line.

##### Parameters

<i>device</i>	Name, path, number or label of the gpiochip.
<i>event_type</i>	Type of events to listen for.
<i>offset</i>	GPIO line offset to monitor.
<i>active_low</i>	The active state of this line - true if low.
<i>consumer</i>	Name of the consumer.
<i>timeout</i>	Maximum wait time for each iteration.
<i>poll_cb</i>	Callback function to call when waiting for events.
<i>event_cb</i>	Callback function to call for each line event.
<i>data</i>	User data passed to the callback.

##### Returns

0 if no errors were encountered, -1 if an error occurred.

##### Note

The way the ctxless event loop works is described in detail in [gpiod\\_ctxless\\_event\\_monitor\\_multiple](#) - this is just a wrapper around this routine which calls it for a single GPIO line.

#### 6.4.4.4 `gpiod_ctxless_event_monitor_ext()`

```
int gpiod_ctxless_event_monitor_ext (
    const char * device,
    int event_type,
    unsigned int offset,
    bool active_low,
    const char * consumer,
    const struct timespec * timeout,
    gpiod_ctxless_event_poll_cb poll_cb,
    gpiod_ctxless_event_handle_cb event_cb,
    void * data,
    int flags)

#include <gpiod.h>
```

Wait for events on a single GPIO line.

## Parameters

<i>device</i>	Name, path, number or label of the gpiochip.
<i>event_type</i>	Type of events to listen for.
<i>offset</i>	GPIO line offset to monitor.
<i>active_low</i>	The active state of this line - true if low.
<i>consumer</i>	Name of the consumer.
<i>timeout</i>	Maximum wait time for each iteration.
<i>poll_cb</i>	Callback function to call when waiting for events.
<i>event_cb</i>	Callback function to call for each line event.
<i>data</i>	User data passed to the callback.
<i>flags</i>	The flags for the line.

## Returns

0 if no errors were encountered, -1 if an error occurred.

## Note

The way the ctxless event loop works is described in detail in [gpiod\\_ctxless\\_event\\_monitor\\_multiple](#) - this is just a wrapper around this routine which calls it for a single GPIO line.

## 6.4.4.5 gpiod\_ctxless\_event\_monitor\_multiple()

```
int gpiod_ctxless_event_monitor_multiple (
    const char * device,
    int event_type,
    const unsigned int * offsets,
    unsigned int num_lines,
    bool active_low,
    const char * consumer,
    const struct timespec * timeout,
    gpiod_ctxless_event_poll_cb poll_cb,
    gpiod_ctxless_event_handle_cb event_cb,
    void * data)
```

```
#include <gpiod.h>
```

Wait for events on multiple GPIO lines.

## Parameters

<i>device</i>	Name, path, number or label of the gpiochip.
<i>event_type</i>	Type of events to listen for.
<i>offsets</i>	Array of GPIO line offsets to monitor.
<i>num_lines</i>	Number of lines to monitor.
<i>active_low</i>	The active state of this line - true if low.
<i>consumer</i>	Name of the consumer.
<i>timeout</i>	Maximum wait time for each iteration.
<i>poll_cb</i>	Callback function to call when waiting for events. Can be NULL.
<i>event_cb</i>	Callback function to call on event occurrence.
<i>data</i>	User data passed to the callback.



**Returns**

0 no errors were encountered, -1 if an error occurred.

**Note**

The poll callback can be NULL in which case the routine will fall back to a basic, ppoll() based callback.

Internally this routine opens the GPIO chip, requests the set of lines for the type of events specified in the event\_type parameter and calls the polling callback in a loop. The role of the polling callback is to detect input events on a set of file descriptors and notify the caller about the fds ready for reading.

The ctxless event loop then reads each queued event from marked descriptors and calls the event callback. Both callbacks can stop the loop at any point.

The poll\_cb argument can be NULL in which case the function falls back to a default, ppoll() based callback.

**6.4.4.6 gpiod\_ctxless\_event\_monitor\_multiple\_ext()**

```
int gpiod_ctxless_event_monitor_multiple_ext (
    const char * device,
    int event_type,
    const unsigned int * offsets,
    unsigned int num_lines,
    bool active_low,
    const char * consumer,
    const struct timespec * timeout,
    gpiod_ctxless_event_poll_cb poll_cb,
    gpiod_ctxless_event_handle_cb event_cb,
    void * data,
    int flags)
```

```
#include <gpiod.h>
```

Wait for events on multiple GPIO lines.

**Parameters**

<i>device</i>	Name, path, number or label of the gpiochip.
<i>event_type</i>	Type of events to listen for.
<i>offsets</i>	Array of GPIO line offsets to monitor.
<i>num_lines</i>	Number of lines to monitor.
<i>active_low</i>	The active state of this line - true if low.
<i>consumer</i>	Name of the consumer.
<i>timeout</i>	Maximum wait time for each iteration.
<i>poll_cb</i>	Callback function to call when waiting for events. Can be NULL.
<i>event_cb</i>	Callback function to call on event occurrence.
<i>data</i>	User data passed to the callback.
<i>flags</i>	The flags for the lines.

**Returns**

0 no errors were encountered, -1 if an error occurred.

**Note**

The poll callback can be NULL in which case the routine will fall back to a basic, ppoll() based callback.

Internally this routine opens the GPIO chip, requests the set of lines for the type of events specified in the event\_type parameter and calls the polling callback in a loop. The role of the polling callback is to detect input events on a set of file descriptors and notify the caller about the fds ready for reading.

The ctxless event loop then reads each queued event from marked descriptors and calls the event callback. Both callbacks can stop the loop at any point.

The poll\_cb argument can be NULL in which case the function falls back to a default, ppoll() based callback.

**6.4.4.7 gpiod\_ctxless\_find\_line()**

```
int gpiod_ctxless_find_line (
    const char * name,
    char * chipname,
    size_t chipname_size,
    unsigned int * offset)
```

```
#include <gpiod.h>
```

Determine the chip name and line offset of a line with given name.

**Parameters**

<i>name</i>	The name of the GPIO line to lookup.
<i>chipname</i>	Buffer in which the name of the GPIO chip will be stored.
<i>chipname_size</i>	Size of the chip name buffer.
<i>offset</i>	Pointer to an integer in which the line offset will be stored.

**Returns**

-1 on error, 0 if the line with given name doesn't exist and 1 if the line was found. In the first two cases the contents of chipname and offset remain unchanged.

**Note**

The chip name is truncated if the buffer can't hold its entire size.

**Attention**

GPIO line names are not unique in the linux kernel, neither globally nor within a single chip. This function finds the first line with given name.

**6.4.4.8 gpiod\_ctxless\_get\_value()**

```
int gpiod_ctxless_get_value (
    const char * device,
    unsigned int offset,
    bool active_low,
    const char * consumer)
```

```
#include <gpiod.h>
```

Read current value from a single GPIO line.

## Parameters

<i>device</i>	Name, path, number or label of the gpiochip.
<i>offset</i>	Offset of the GPIO line.
<i>active_low</i>	The active state of this line - true if low.
<i>consumer</i>	Name of the consumer.

## Returns

0 or 1 (GPIO value) if the operation succeeds, -1 on error.

6.4.4.9 `gpiod_ctxless_get_value_ext()`

```
int gpiod_ctxless_get_value_ext (
    const char * device,
    unsigned int offset,
    bool active_low,
    const char * consumer,
    int flags)
```

```
#include <gpiod.h>
```

Read current value from a single GPIO line.

## Parameters

<i>device</i>	Name, path, number or label of the gpiochip.
<i>offset</i>	Offset of the GPIO line.
<i>active_low</i>	The active state of this line - true if low.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	The flags for the line.

## Returns

0 or 1 (GPIO value) if the operation succeeds, -1 on error.

6.4.4.10 `gpiod_ctxless_get_value_multiple()`

```
int gpiod_ctxless_get_value_multiple (
    const char * device,
    const unsigned int * offsets,
    int * values,
    unsigned int num_lines,
    bool active_low,
    const char * consumer)
```

```
#include <gpiod.h>
```

Read current values from a set of GPIO lines.

**Parameters**

<i>device</i>	Name, path, number or label of the gpiochip.
<i>offsets</i>	Array of offsets of lines whose values should be read.
<i>values</i>	Buffer in which the values will be stored.
<i>num_lines</i>	Number of lines, must be > 0.
<i>active_low</i>	The active state of the lines - true if low.
<i>consumer</i>	Name of the consumer.

**Returns**

0 if the operation succeeds, -1 on error.

**6.4.4.11 `gpiod_ctxless_get_value_multiple_ext()`**

```
int gpiod_ctxless_get_value_multiple_ext (
    const char * device,
    const unsigned int * offsets,
    int * values,
    unsigned int num_lines,
    bool active_low,
    const char * consumer,
    int flags)
```

```
#include <gpiod.h>
```

Read current values from a set of GPIO lines.

**Parameters**

<i>device</i>	Name, path, number or label of the gpiochip.
<i>offsets</i>	Array of offsets of lines whose values should be read.
<i>values</i>	Buffer in which the values will be stored.
<i>num_lines</i>	Number of lines, must be > 0.
<i>active_low</i>	The active state of this line - true if low.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	The flags for the lines.

**Returns**

0 if the operation succeeds, -1 on error.

**6.4.4.12 `gpiod_ctxless_set_value()`**

```
int gpiod_ctxless_set_value (
    const char * device,
    unsigned int offset,
    int value,
    bool active_low,
    const char * consumer,
    gpiod_ctxless_set_value_cb cb,
    void * data)
```

```
#include <gpiod.h>
```

Set value of a single GPIO line.

## Parameters

<i>device</i>	Name, path, number or label of the gpiochip.
<i>offset</i>	The offset of the GPIO line.
<i>value</i>	New value (0 or 1).
<i>active_low</i>	The active state of this line - true if low.
<i>consumer</i>	Name of the consumer.
<i>cb</i>	Optional callback function that will be called right after setting the value. Users can use this, for example, to pause the execution after toggling a GPIO.
<i>data</i>	Optional user data that will be passed to the callback function.

## Returns

0 if the operation succeeds, -1 on error.

6.4.4.13 `gpiod_ctxless_set_value_ext()`

```
int gpiod_ctxless_set_value_ext (
    const char * device,
    unsigned int offset,
    int value,
    bool active_low,
    const char * consumer,
    gpiod_ctxless_set_value_cb cb,
    void * data,
    int flags)
```

```
#include <gpiod.h>
```

Set value of a single GPIO line.

## Parameters

<i>device</i>	Name, path, number or label of the gpiochip.
<i>offset</i>	The offset of the GPIO line.
<i>value</i>	New value (0 or 1).
<i>active_low</i>	The active state of this line - true if low.
<i>consumer</i>	Name of the consumer.
<i>cb</i>	Optional callback function that will be called right after setting the value. Users can use this, for example, to pause the execution after toggling a GPIO.
<i>data</i>	Optional user data that will be passed to the callback function.
<i>flags</i>	The flags for the line.

## Returns

0 if the operation succeeds, -1 on error.

#### 6.4.4.14 `gpiod_ctxless_set_value_multiple()`

```
int gpiod_ctxless_set_value_multiple (
    const char * device,
    const unsigned int * offsets,
    const int * values,
    unsigned int num_lines,
    bool active_low,
    const char * consumer,
    gpiod_ctxless_set_value_cb cb,
    void * data)
```

```
#include <gpiod.h>
```

Set values of multiple GPIO lines.

##### Parameters

<i>device</i>	Name, path, number or label of the gpiochip.
<i>offsets</i>	Array of offsets of lines the values of which should be set.
<i>values</i>	Array of integers containing new values.
<i>num_lines</i>	Number of lines, must be > 0.
<i>active_low</i>	The active state of the lines - true if low.
<i>consumer</i>	Name of the consumer.
<i>cb</i>	Optional callback function that will be called right after setting all values. Works the same as in <a href="#">gpiod_ctxless_set_value</a> .
<i>data</i>	Optional user data that will be passed to the callback function.

##### Returns

0 if the operation succeeds, -1 on error.

#### 6.4.4.15 `gpiod_ctxless_set_value_multiple_ext()`

```
int gpiod_ctxless_set_value_multiple_ext (
    const char * device,
    const unsigned int * offsets,
    const int * values,
    unsigned int num_lines,
    bool active_low,
    const char * consumer,
    gpiod_ctxless_set_value_cb cb,
    void * data,
    int flags)
```

```
#include <gpiod.h>
```

Set values of multiple GPIO lines.

##### Parameters

<i>device</i>	Name, path, number or label of the gpiochip.
<i>offsets</i>	Array of offsets of lines the values of which should be set.

## Parameters

<i>values</i>	Array of integers containing new values.
<i>num_lines</i>	Number of lines, must be > 0.
<i>active_low</i>	The active state of this line - true if low.
<i>consumer</i>	Name of the consumer.
<i>cb</i>	Optional callback function that will be called right after setting all values. Works the same as in <a href="#">gpiod_ctxless_set_value</a> .
<i>data</i>	Optional user data that will be passed to the callback function.
<i>flags</i>	The flags for the lines.

## Returns

0 if the operation succeeds, -1 on error.

## 6.5 Iterators for GPIO chips and lines

## Macros

- `#define gpiod\_foreach\_chip(iter, chip)`  
*Iterate over all GPIO chips present in the system.*
- `#define gpiod\_foreach\_chip\_noclose(iter, chip)`  
*Iterate over all chips present in the system without closing them.*
- `#define gpiod\_foreach\_line(iter, line)`  
*Iterate over all GPIO lines of a single chip.*

## Functions

- `struct gpiod_chip_iter * gpiod\_chip\_iter\_new (void) GPIOD\_API`  
*Create a new gpiochip iterator.*
- `void gpiod\_chip\_iter\_free (struct gpiod_chip_iter *iter) GPIOD\_API`  
*Release all resources allocated for the gpiochip iterator and close the most recently opened gpiochip (if any).*
- `void gpiod\_chip\_iter\_free\_noclose (struct gpiod_chip_iter *iter) GPIOD\_API`  
*Release all resources allocated for the gpiochip iterator but don't close the most recently opened gpiochip (if any).*
- `struct gpiod_chip * gpiod\_chip\_iter\_next (struct gpiod_chip_iter *iter) GPIOD\_API`  
*Get the next gpiochip handle.*
- `struct gpiod_chip * gpiod\_chip\_iter\_next\_noclose (struct gpiod_chip_iter *iter) GPIOD\_API`  
*Get the next gpiochip handle without closing the previous one.*
- `struct gpiod_line_iter * gpiod\_line\_iter\_new (struct gpiod_chip *chip) GPIOD\_API`  
*Create a new line iterator.*
- `void gpiod\_line\_iter\_free (struct gpiod_line_iter *iter) GPIOD\_API`  
*Free all resources associated with a GPIO line iterator.*
- `struct gpiod_line * gpiod\_line\_iter\_next (struct gpiod_line_iter *iter) GPIOD\_API`  
*Get the next GPIO line handle.*

### 6.5.1 Detailed Description

These functions and data structures allow easy iterating over GPIO chips and lines.

## 6.5.2 Macro Definition Documentation

### 6.5.2.1 `gpiod_foreach_chip`

```
#define gpiod_foreach_chip(
    iter,
    chip)
```

```
#include <gpiod.h>
```

**Value:**

```
for ((chip) = gpiod_chip_iter_next(iter); \
    (chip); \
    (chip) = gpiod_chip_iter_next(iter))
```

Iterate over all GPIO chips present in the system.

**Parameters**

<i>iter</i>	An initialized GPIO chip iterator.
<i>chip</i>	Pointer to a GPIO chip handle. On each iteration the newly opened chip handle is assigned to this argument.

The user must not close the GPIO chip manually - instead the previous chip handle is closed automatically on the next iteration. The last chip to be opened is closed internally by `gpiod_chip_iter_free`.

### 6.5.2.2 `gpiod_foreach_chip_noclose`

```
#define gpiod_foreach_chip_noclose(
    iter,
    chip)
```

```
#include <gpiod.h>
```

**Value:**

```
for ((chip) = gpiod_chip_iter_next_noclose(iter); \
    (chip); \
    (chip) = gpiod_chip_iter_next_noclose(iter))
```

Iterate over all chips present in the system without closing them.

**Parameters**

<i>iter</i>	An initialized GPIO chip iterator.
<i>chip</i>	Pointer to a GPIO chip handle. On each iteration the newly opened chip handle is assigned to this argument.

The user must close all the GPIO chips manually after use, until then, the chips remain open. Free the iterator by calling `gpiod_chip_iter_free_noclose` to avoid closing the last chip automatically.

### 6.5.2.3 `gpiod_foreach_line`

```
#define gpiod_foreach_line(
    iter,
    line)
```

```
#include <gpiod.h>
```

**Value:**

```
for ((line) = gpiod_line_iter_next(iter); \
    (line); \
    (line) = gpiod_line_iter_next(iter))
```

Iterate over all GPIO lines of a single chip.



## Parameters

<i>iter</i>	An initialized GPIO line iterator.
<i>line</i>	Pointer to a GPIO line handle - on each iteration, the next GPIO line will be assigned to this argument.

### 6.5.3 Function Documentation

#### 6.5.3.1 `gpiod_chip_iter_free()`

```
void gpiod_chip_iter_free (  
    struct gpiod_chip_iter * iter)
```

```
#include <gpiod.h>
```

Release all resources allocated for the gpiochip iterator and close the most recently opened gpiochip (if any).

## Parameters

<i>iter</i>	The gpiochip iterator object.
-------------	-------------------------------

#### 6.5.3.2 `gpiod_chip_iter_free_noclose()`

```
void gpiod_chip_iter_free_noclose (  
    struct gpiod_chip_iter * iter)
```

```
#include <gpiod.h>
```

Release all resources allocated for the gpiochip iterator but don't close the most recently opened gpiochip (if any).

## Parameters

<i>iter</i>	The gpiochip iterator object.
-------------	-------------------------------

Users may want to break the loop when iterating over gpiochips and keep the most recently opened chip active while freeing the iterator data. This routine enables that.

#### 6.5.3.3 `gpiod_chip_iter_new()`

```
struct gpiod_chip_iter * gpiod_chip_iter_new (  
    void )
```

```
#include <gpiod.h>
```

Create a new gpiochip iterator.

## Returns

Pointer to a new chip iterator object or NULL if an error occurred.

Internally this routine scans the `/dev/` directory for GPIO chip device files, opens them and stores their the handles until `gpiod_chip_iter_free` or `gpiod_chip_iter_free_noclose` is called.

#### 6.5.3.4 `gpiod_chip_iter_next()`

```
struct gpiod_chip * gpiod_chip_iter_next (  
    struct gpiod_chip_iter * iter)
```

```
#include <gpiod.h>
```

Get the next gpiochip handle.

**Parameters**

<i>iter</i>	The gpiochip iterator object.
-------------	-------------------------------

**Returns**

Pointer to the next open gpiochip handle or NULL if no more chips are present in the system.

**Note**

The previous chip handle will be closed using [gpiod\\_chip\\_iter\\_free](#).

**6.5.3.5 gpiod\_chip\_iter\_next\_noclose()**

```
struct gpiochip * gpiod_chip_iter_next_noclose (  
    struct gpiochip_iter * iter)
```

```
#include <gpiod.h>
```

Get the next gpiochip handle without closing the previous one.

**Parameters**

<i>iter</i>	The gpiochip iterator object.
-------------	-------------------------------

**Returns**

Pointer to the next open gpiochip handle or NULL if no more chips are present in the system.

**Note**

This function works just like [gpiod\\_chip\\_iter\\_next](#) but doesn't close the most recently opened chip handle.

**6.5.3.6 gpiod\_line\_iter\_free()**

```
void gpiod_line_iter_free (  
    struct gpiod_line_iter * iter)
```

```
#include <gpiod.h>
```

Free all resources associated with a GPIO line iterator.

**Parameters**

<i>iter</i>	Line iterator object.
-------------	-----------------------

**6.5.3.7 gpiod\_line\_iter\_new()**

```
struct gpiod_line_iter * gpiod_line_iter_new (  
    struct gpiochip * chip)
```

```
#include <gpiod.h>
```

Create a new line iterator.

**Parameters**

<i>chip</i>	Active gpiochip handle over the lines of which we want to iterate.
-------------	--

**Returns**

New line iterator or NULL if an error occurred.

**6.5.3.8 gpiod\_line\_iter\_next()**

```
struct gpiod_line * gpiod_line_iter_next (
    struct gpiod_line_iter * iter)
```

```
#include <gpiod.h>
```

Get the next GPIO line handle.

**Parameters**

<i>iter</i>	The GPIO line iterator object.
-------------	--------------------------------

**Returns**

Pointer to the next GPIO line handle or NULL if there are no more lines left.

**6.6 Stuff that didn't fit anywhere else****Functions**

- `const char * gpiod_version_string (void) GPIOD_API`  
*Get the API version of the library as a human-readable string.*

**6.6.1 Detailed Description**

Various libgpiod-related functions.

**6.6.2 Function Documentation****6.6.2.1 gpiod\_version\_string()**

```
const char * gpiod_version_string (
    void )
```

```
#include <gpiod.h>
```

Get the API version of the library as a human-readable string.

**Returns**

Human-readable string containing the library version.



# Chapter 7

## Class Documentation

### 7.1 `gpiod_ctxless_event_poll_fd` Struct Reference

Helper structure for the ctxless event loop poll callback.

```
#include <gpiod.h>
```

#### Public Attributes

- int `fd`
- bool `event`

#### 7.1.1 Detailed Description

Helper structure for the ctxless event loop poll callback.

#### 7.1.2 Member Data Documentation

##### 7.1.2.1 `event`

```
bool gpiod_ctxless_event_poll_fd::event
```

Indicates whether an event occurred on this file descriptor.

##### 7.1.2.2 `fd`

```
int gpiod_ctxless_event_poll_fd::fd
```

File descriptor number.

The documentation for this struct was generated from the following file:

- [gpiod.h](#)

## 7.2 gpiod\_line\_bulk Struct Reference

Helper structure for storing a set of GPIO line objects.

```
#include <gpiod.h>
```

### Public Attributes

- struct gpiod\_line \* [lines](#) [[GPIOD\\_LINE\\_BULK\\_MAX\\_LINES](#)]
- unsigned int [num\\_lines](#)

### 7.2.1 Detailed Description

Helper structure for storing a set of GPIO line objects.

This structure is used in all operations involving sets of GPIO lines. If a bulk object is being passed to a function while containing zero lines, the result is undefined.

### 7.2.2 Member Data Documentation

#### 7.2.2.1 [lines](#)

```
struct gpiod_line* gpiod_line_bulk::lines[GPIOD\_LINE\_BULK\_MAX\_LINES]
```

Buffer for line pointers.

#### 7.2.2.2 [num\\_lines](#)

```
unsigned int gpiod_line_bulk::num_lines
```

Number of lines currently held in this structure.

The documentation for this struct was generated from the following file:

- [gpiod.h](#)

## 7.3 gpiod\_line\_event Struct Reference

Structure holding event info.

```
#include <gpiod.h>
```

### Public Attributes

- struct timespec [ts](#)
- int [event\\_type](#)

### 7.3.1 Detailed Description

Structure holding event info.

### 7.3.2 Member Data Documentation

#### 7.3.2.1 event\_type

```
int gpiod_line_event::event_type
```

Type of the event that occurred.

#### 7.3.2.2 ts

```
struct timespec gpiod_line_event::ts
```

Best estimate of time of event occurrence.

The documentation for this struct was generated from the following file:

- [gpiod.h](#)

## 7.4 gpiod\_line\_request\_config Struct Reference

Structure holding configuration of a line request.

```
#include <gpiod.h>
```

### Public Attributes

- const char \* [consumer](#)
- int [request\\_type](#)
- int [flags](#)

### 7.4.1 Detailed Description

Structure holding configuration of a line request.

### 7.4.2 Member Data Documentation

#### 7.4.2.1 consumer

```
const char* gpiod_line_request_config::consumer
```

Name of the consumer.

#### 7.4.2.2 flags

```
int gpiod_line_request_config::flags
```

Other configuration flags.

#### 7.4.2.3 request\_type

```
int gpiod_line_request_config::request_type
```

Request type.

The documentation for this struct was generated from the following file:

- [gpiod.h](#)



# Chapter 8

## File Documentation

### 8.1 gpiod.h File Reference

```
#include <stdbool.h>
#include <stdlib.h>
#include <time.h>
```

#### Classes

- struct [gpiod\\_ctxless\\_event\\_poll\\_fd](#)  
*Helper structure for the ctxless event loop poll callback.*
- struct [gpiod\\_line\\_bulk](#)  
*Helper structure for storing a set of GPIO line objects.*
- struct [gpiod\\_line\\_request\\_config](#)  
*Structure holding configuration of a line request.*
- struct [gpiod\\_line\\_event](#)  
*Structure holding event info.*

#### Macros

- #define **GPIOD\_API** \_\_attribute\_\_((visibility("default")))  
*Makes symbol visible.*
- #define **GPIOD\_UNUSED** \_\_attribute\_\_((unused))  
*Marks a function argument or variable as potentially unused.*
- #define [GPIOD\\_BIT](#)(nr)  
*Shift 1 by given offset.*
- #define **GPIOD\_DEPRECATED** \_\_attribute\_\_((deprecated))  
*Marks a public function as deprecated.*
- #define **GPIOD\_LINE\_BULK\_MAX\_LINES** 64  
*Maximum number of GPIO lines that can be requested at once.*
- #define [GPIOD\\_LINE\\_BULK\\_INITIALIZER](#) { { NULL }, 0 }  
*Static initializer for GPIO bulk objects.*
- #define [gpiod\\_line\\_bulk\\_foreach\\_line](#)(bulk, line, lineptr)  
*Iterate over all line handles held by a line bulk object.*

- `#define gpiod_line_bulk_foreach_line_off(bulk, line, offset)`  
*Iterate over all line handles held by a line bulk object (integer counter variant).*
- `#define gpiod_foreach_chip(iter, chip)`  
*Iterate over all GPIO chips present in the system.*
- `#define gpiod_foreach_chip_noclose(iter, chip)`  
*Iterate over all chips present in the system without closing them.*
- `#define gpiod_foreach_line(iter, line)`  
*Iterate over all GPIO lines of a single chip.*

## Typedefs

- `typedef void(* gpiod_ctxless_set_value_cb) (void *)`  
*Simple set value callback signature.*
- `typedef int(* gpiod_ctxless_event_handle_cb) (int, unsigned int, const struct timespec *, void *)`  
*Simple event callback signature.*
- `typedef int(* gpiod_ctxless_event_poll_cb) (unsigned int, struct gpiod_ctxless_event_poll_fd *, const struct timespec *, void *)`  
*Simple event poll callback signature.*

## Enumerations

- `enum {`  
`GPIOD_CTXLESS_FLAG_OPEN_DRAIN = GPIOD_BIT(0) , GPIOD_CTXLESS_FLAG_OPEN_SOURCE =`  
`GPIOD_BIT(1) , GPIOD_CTXLESS_FLAG_BIAS_DISABLE = GPIOD_BIT(2) , GPIOD_CTXLESS_FLAG_BIAS_PULL_DOWN`  
`= GPIOD_BIT(3) ,`  
`GPIOD_CTXLESS_FLAG_BIAS_PULL_UP = GPIOD_BIT(4) }`  
*Miscellaneous GPIO flags.*
- `enum { GPIOD_CTXLESS_EVENT_RISING_EDGE = 1 , GPIOD_CTXLESS_EVENT_FALLING_EDGE ,`  
`GPIOD_CTXLESS_EVENT_BOTH_EDGES }`  
*Event types that the ctxless event monitor can wait for.*
- `enum { GPIOD_CTXLESS_EVENT_CB_TIMEOUT = 1 , GPIOD_CTXLESS_EVENT_CB_RISING_EDGE ,`  
`GPIOD_CTXLESS_EVENT_CB_FALLING_EDGE }`  
*Event types that can be passed to the ctxless event callback.*
- `enum { GPIOD_CTXLESS_EVENT_CB_RET_ERR = -1 , GPIOD_CTXLESS_EVENT_CB_RET_OK = 0 ,`  
`GPIOD_CTXLESS_EVENT_CB_RET_STOP = 1 }`  
*Return status values that the ctxless event callback can return.*
- `enum { GPIOD_CTXLESS_EVENT_POLL_RET_STOP = -2 , GPIOD_CTXLESS_EVENT_POLL_RET_ERR`  
`= -1 , GPIOD_CTXLESS_EVENT_POLL_RET_TIMEOUT = 0 }`  
*Return status values that the ctxless event poll callback can return.*
- `enum { GPIOD_LINE_DIRECTION_INPUT = 1 , GPIOD_LINE_DIRECTION_OUTPUT }`  
*Possible direction settings.*
- `enum { GPIOD_LINE_ACTIVE_STATE_HIGH = 1 , GPIOD_LINE_ACTIVE_STATE_LOW }`  
*Possible active state settings.*
- `enum { GPIOD_LINE_BIAS_AS_IS = 1 , GPIOD_LINE_BIAS_DISABLE , GPIOD_LINE_BIAS_PULL_UP ,`  
`GPIOD_LINE_BIAS_PULL_DOWN }`  
*Possible internal bias settings.*
- `enum {`  
`GPIOD_LINE_REQUEST_DIRECTION_AS_IS = 1 , GPIOD_LINE_REQUEST_DIRECTION_INPUT ,`  
`GPIOD_LINE_REQUEST_DIRECTION_OUTPUT , GPIOD_LINE_REQUEST_EVENT_FALLING_EDGE`  
`,`  
`GPIOD_LINE_REQUEST_EVENT_RISING_EDGE , GPIOD_LINE_REQUEST_EVENT_BOTH_EDGES }`

*Available types of requests.*

- enum {  
[GPIOD\\_LINE\\_REQUEST\\_FLAG\\_OPEN\\_DRAIN](#) = [GPIOD\\_BIT\(0\)](#) , [GPIOD\\_LINE\\_REQUEST\\_FLAG\\_OPEN\\_SOURCE](#)  
= [GPIOD\\_BIT\(1\)](#) , [GPIOD\\_LINE\\_REQUEST\\_FLAG\\_ACTIVE\\_LOW](#) = [GPIOD\\_BIT\(2\)](#) , [GPIOD\\_LINE\\_REQUEST\\_FLAG\\_BIAS](#)  
= [GPIOD\\_BIT\(3\)](#) ,  
[GPIOD\\_LINE\\_REQUEST\\_FLAG\\_BIAS\\_PULL\\_DOWN](#) = [GPIOD\\_BIT\(4\)](#) , [GPIOD\\_LINE\\_REQUEST\\_FLAG\\_BIAS\\_PULL\\_UP](#)  
= [GPIOD\\_BIT\(5\)](#) }

*Miscellaneous GPIO request flags.*

- enum { [GPIOD\\_LINE\\_EVENT\\_RISING\\_EDGE](#) = 1 , [GPIOD\\_LINE\\_EVENT\\_FALLING\\_EDGE](#) }

*Event types.*

## Functions

- int [gpiod\\_ctxless\\_get\\_value](#) (const char \*device, unsigned int offset, bool active\_low, const char \*consumer)  
[GPIOD\\_API](#)

*Read current value from a single GPIO line.*

- int [gpiod\\_ctxless\\_get\\_value\\_ext](#) (const char \*device, unsigned int offset, bool active\_low, const char \*consumer, int flags) [GPIOD\\_API](#)

*Read current value from a single GPIO line.*

- int [gpiod\\_ctxless\\_get\\_value\\_multiple](#) (const char \*device, const unsigned int \*offsets, int \*values, unsigned int num\_lines, bool active\_low, const char \*consumer) [GPIOD\\_API](#)

*Read current values from a set of GPIO lines.*

- int [gpiod\\_ctxless\\_get\\_value\\_multiple\\_ext](#) (const char \*device, const unsigned int \*offsets, int \*values, unsigned int num\_lines, bool active\_low, const char \*consumer, int flags) [GPIOD\\_API](#)

*Read current values from a set of GPIO lines.*

- int [gpiod\\_ctxless\\_set\\_value](#) (const char \*device, unsigned int offset, int value, bool active\_low, const char \*consumer, [gpiod\\_ctxless\\_set\\_value\\_cb](#) cb, void \*data) [GPIOD\\_API](#)

*Set value of a single GPIO line.*

- int [gpiod\\_ctxless\\_set\\_value\\_ext](#) (const char \*device, unsigned int offset, int value, bool active\_low, const char \*consumer, [gpiod\\_ctxless\\_set\\_value\\_cb](#) cb, void \*data, int flags) [GPIOD\\_API](#)

*Set value of a single GPIO line.*

- int [gpiod\\_ctxless\\_set\\_value\\_multiple](#) (const char \*device, const unsigned int \*offsets, const int \*values, unsigned int num\_lines, bool active\_low, const char \*consumer, [gpiod\\_ctxless\\_set\\_value\\_cb](#) cb, void \*data) [GPIOD\\_API](#)

*Set values of multiple GPIO lines.*

- int [gpiod\\_ctxless\\_set\\_value\\_multiple\\_ext](#) (const char \*device, const unsigned int \*offsets, const int \*values, unsigned int num\_lines, bool active\_low, const char \*consumer, [gpiod\\_ctxless\\_set\\_value\\_cb](#) cb, void \*data, int flags) [GPIOD\\_API](#)

*Set values of multiple GPIO lines.*

- int [gpiod\\_ctxless\\_event\\_loop](#) (const char \*device, unsigned int offset, bool active\_low, const char \*consumer, const struct timespec \*timeout, [gpiod\\_ctxless\\_event\\_poll\\_cb](#) poll\_cb, [gpiod\\_ctxless\\_event\\_handle\\_cb](#) event\_cb, void \*data) [GPIOD\\_API](#) [GPIOD\\_DEPRECATED](#)

*Wait for events on a single GPIO line.*

- int [gpiod\\_ctxless\\_event\\_loop\\_multiple](#) (const char \*device, const unsigned int \*offsets, unsigned int num\_lines, bool active\_low, const char \*consumer, const struct timespec \*timeout, [gpiod\\_ctxless\\_event\\_poll\\_cb](#) poll\_cb, [gpiod\\_ctxless\\_event\\_handle\\_cb](#) event\_cb, void \*data) [GPIOD\\_API](#) [GPIOD\\_DEPRECATED](#)

*Wait for events on multiple GPIO lines.*

- int [gpiod\\_ctxless\\_event\\_monitor](#) (const char \*device, int event\_type, unsigned int offset, bool active\_low, const char \*consumer, const struct timespec \*timeout, [gpiod\\_ctxless\\_event\\_poll\\_cb](#) poll\_cb, [gpiod\\_ctxless\\_event\\_handle\\_cb](#) event\_cb, void \*data) [GPIOD\\_API](#)

*Wait for events on a single GPIO line.*

- int [gpiod\\_ctxless\\_event\\_monitor\\_ext](#) (const char \*device, int event\_type, unsigned int offset, bool active\_low, const char \*consumer, const struct timespec \*timeout, [gpiod\\_ctxless\\_event\\_poll\\_cb](#) poll\_cb, [gpiod\\_ctxless\\_event\\_handle\\_cb](#) event\_cb, void \*data, int flags) [GPIOD\\_API](#)

*Wait for events on a single GPIO line.*

- int [gpiod\\_ctxless\\_event\\_monitor\\_multiple](#) (const char \*device, int event\_type, const unsigned int \*offsets, unsigned int num\_lines, bool active\_low, const char \*consumer, const struct timespec \*timeout, [gpiod\\_ctxless\\_event\\_poll\\_cb](#) poll\_cb, [gpiod\\_ctxless\\_event\\_handle\\_cb](#) event\_cb, void \*data) [GPIO\\_API](#)

*Wait for events on multiple GPIO lines.*

- int [gpiod\\_ctxless\\_event\\_monitor\\_multiple\\_ext](#) (const char \*device, int event\_type, const unsigned int \*offsets, unsigned int num\_lines, bool active\_low, const char \*consumer, const struct timespec \*timeout, [gpiod\\_ctxless\\_event\\_poll\\_cb](#) poll\_cb, [gpiod\\_ctxless\\_event\\_handle\\_cb](#) event\_cb, void \*data, int flags) [GPIO\\_API](#)

*Wait for events on multiple GPIO lines.*

- int [gpiod\\_ctxless\\_find\\_line](#) (const char \*name, char \*chipname, size\_t chipname\_size, unsigned int \*offset) [GPIO\\_API](#)

*Determine the chip name and line offset of a line with given name.*

- struct gpiod\_chip \* [gpiod\\_chip\\_open](#) (const char \*path) [GPIO\\_API](#)

*Open a gpiochip by path.*

- struct gpiod\_chip \* [gpiod\\_chip\\_open\\_by\\_name](#) (const char \*name) [GPIO\\_API](#)

*Open a gpiochip by name.*

- struct gpiod\_chip \* [gpiod\\_chip\\_open\\_by\\_number](#) (unsigned int num) [GPIO\\_API](#)

*Open a gpiochip by number.*

- struct gpiod\_chip \* [gpiod\\_chip\\_open\\_by\\_label](#) (const char \*label) [GPIO\\_API](#)

*Open a gpiochip by label.*

- struct gpiod\_chip \* [gpiod\\_chip\\_open\\_lookup](#) (const char \*descr) [GPIO\\_API](#)

*Open a gpiochip based on the best guess what the path is.*

- void [gpiod\\_chip\\_close](#) (struct gpiod\_chip \*chip) [GPIO\\_API](#)

*Close a GPIO chip handle and release all allocated resources.*

- const char \* [gpiod\\_chip\\_name](#) (struct gpiod\_chip \*chip) [GPIO\\_API](#)

*Get the GPIO chip name as represented in the kernel.*

- const char \* [gpiod\\_chip\\_label](#) (struct gpiod\_chip \*chip) [GPIO\\_API](#)

*Get the GPIO chip label as represented in the kernel.*

- unsigned int [gpiod\\_chip\\_num\\_lines](#) (struct gpiod\_chip \*chip) [GPIO\\_API](#)

*Get the number of GPIO lines exposed by this chip.*

- struct gpiod\_line \* [gpiod\\_chip\\_get\\_line](#) (struct gpiod\_chip \*chip, unsigned int offset) [GPIO\\_API](#)

*Get the handle to the GPIO line at given offset.*

- int [gpiod\\_chip\\_get\\_lines](#) (struct gpiod\_chip \*chip, unsigned int \*offsets, unsigned int num\_offsets, struct [gpiod\\_line\\_bulk](#) \*bulk) [GPIO\\_API](#)

*Retrieve a set of lines and store them in a line bulk object.*

- int [gpiod\\_chip\\_get\\_all\\_lines](#) (struct gpiod\_chip \*chip, struct [gpiod\\_line\\_bulk](#) \*bulk) [GPIO\\_API](#)

*Retrieve all lines exposed by a chip and store them in a bulk object.*

- struct gpiod\_line \* [gpiod\\_chip\\_find\\_line](#) (struct gpiod\_chip \*chip, const char \*name) [GPIO\\_API](#)

*Find a GPIO line by name among lines associated with given GPIO chip.*

- int [gpiod\\_chip\\_find\\_lines](#) (struct gpiod\_chip \*chip, const char \*\*names, struct [gpiod\\_line\\_bulk](#) \*bulk) [GPIO\\_API](#)

*Find a set of GPIO lines by names among lines exposed by this chip.*

- static void [gpiod\\_line\\_bulk\\_init](#) (struct [gpiod\\_line\\_bulk](#) \*bulk)

*Initialize a GPIO bulk object.*

- static void [gpiod\\_line\\_bulk\\_add](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, struct gpiod\_line \*line)

*Add a single line to a GPIO bulk object.*

- static struct gpiod\_line \* [gpiod\\_line\\_bulk\\_get\\_line](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, unsigned int offset)

*Retrieve the line handle from a line bulk object at given offset.*

- static unsigned int [gpiod\\_line\\_bulk\\_num\\_lines](#) (struct [gpiod\\_line\\_bulk](#) \*bulk)

*Retrieve the number of GPIO lines held by this line bulk object.*

- unsigned int [gpiod\\_line\\_offset](#) (struct gpiod\_line \*line) [GPIO\\_API](#)

- Read the GPIO line offset.*
- const char \* [gpiod\\_line\\_name](#) (struct gpiod\_line \*line) [GPIOD\\_API](#)  
*Read the GPIO line name.*
- const char \* [gpiod\\_line\\_consumer](#) (struct gpiod\_line \*line) [GPIOD\\_API](#)  
*Read the GPIO line consumer name.*
- int [gpiod\\_line\\_direction](#) (struct gpiod\_line \*line) [GPIOD\\_API](#)  
*Read the GPIO line direction setting.*
- int [gpiod\\_line\\_active\\_state](#) (struct gpiod\_line \*line) [GPIOD\\_API](#)  
*Read the GPIO line active state setting.*
- int [gpiod\\_line\\_bias](#) (struct gpiod\_line \*line) [GPIOD\\_API](#)  
*Read the GPIO line bias setting.*
- bool [gpiod\\_line\\_is\\_used](#) (struct gpiod\_line \*line) [GPIOD\\_API](#)  
*Check if the line is currently in use.*
- bool [gpiod\\_line\\_is\\_open\\_drain](#) (struct gpiod\_line \*line) [GPIOD\\_API](#)  
*Check if the line is an open-drain GPIO.*
- bool [gpiod\\_line\\_is\\_open\\_source](#) (struct gpiod\_line \*line) [GPIOD\\_API](#)  
*Check if the line is an open-source GPIO.*
- int [gpiod\\_line\\_update](#) (struct gpiod\_line \*line) [GPIOD\\_API](#)  
*Re-read the line info.*
- bool [gpiod\\_line\\_needs\\_update](#) (struct gpiod\_line \*line) [GPIOD\\_API](#) [GPIOD\\_DEPRECATED](#)  
*Check if the line info needs to be updated.*
- int [gpiod\\_line\\_request](#) (struct gpiod\_line \*line, const struct [gpiod\\_line\\_request\\_config](#) \*config, int default\_val) [GPIOD\\_API](#)  
*Reserve a single line.*
- int [gpiod\\_line\\_request\\_input](#) (struct gpiod\_line \*line, const char \*consumer) [GPIOD\\_API](#)  
*Reserve a single line, set the direction to input.*
- int [gpiod\\_line\\_request\\_output](#) (struct gpiod\_line \*line, const char \*consumer, int default\_val) [GPIOD\\_API](#)  
*Reserve a single line, set the direction to output.*
- int [gpiod\\_line\\_request\\_rising\\_edge\\_events](#) (struct gpiod\_line \*line, const char \*consumer) [GPIOD\\_API](#)  
*Request rising edge event notifications on a single line.*
- int [gpiod\\_line\\_request\\_falling\\_edge\\_events](#) (struct gpiod\_line \*line, const char \*consumer) [GPIOD\\_API](#)  
*Request falling edge event notifications on a single line.*
- int [gpiod\\_line\\_request\\_both\\_edges\\_events](#) (struct gpiod\_line \*line, const char \*consumer) [GPIOD\\_API](#)  
*Request all event type notifications on a single line.*
- int [gpiod\\_line\\_request\\_input\\_flags](#) (struct gpiod\_line \*line, const char \*consumer, int flags) [GPIOD\\_API](#)  
*Reserve a single line, set the direction to input.*
- int [gpiod\\_line\\_request\\_output\\_flags](#) (struct gpiod\_line \*line, const char \*consumer, int flags, int default\_val) [GPIOD\\_API](#)  
*Reserve a single line, set the direction to output.*
- int [gpiod\\_line\\_request\\_rising\\_edge\\_events\\_flags](#) (struct gpiod\_line \*line, const char \*consumer, int flags) [GPIOD\\_API](#)  
*Request rising edge event notifications on a single line.*
- int [gpiod\\_line\\_request\\_falling\\_edge\\_events\\_flags](#) (struct gpiod\_line \*line, const char \*consumer, int flags) [GPIOD\\_API](#)  
*Request falling edge event notifications on a single line.*
- int [gpiod\\_line\\_request\\_both\\_edges\\_events\\_flags](#) (struct gpiod\_line \*line, const char \*consumer, int flags) [GPIOD\\_API](#)  
*Request all event type notifications on a single line.*
- int [gpiod\\_line\\_request\\_bulk](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const struct [gpiod\\_line\\_request\\_config](#) \*config, const int \*default\_vals) [GPIOD\\_API](#)  
*Reserve a set of GPIO lines.*

- int [gpiod\\_line\\_request\\_bulk\\_input](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer) [GPIO\\_API](#)  
*Reserve a set of GPIO lines, set the direction to input.*
- int [gpiod\\_line\\_request\\_bulk\\_output](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer, const int \*default\_↵  
\_↵vals) [GPIO\\_API](#)  
*Reserve a set of GPIO lines, set the direction to output.*
- int [gpiod\\_line\\_request\\_bulk\\_rising\\_edge\\_events](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer)  
[GPIO\\_API](#)  
*Request rising edge event notifications on a set of lines.*
- int [gpiod\\_line\\_request\\_bulk\\_falling\\_edge\\_events](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer)  
[GPIO\\_API](#)  
*Request falling edge event notifications on a set of lines.*
- int [gpiod\\_line\\_request\\_bulk\\_both\\_edges\\_events](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer)  
[GPIO\\_API](#)  
*Request all event type notifications on a set of lines.*
- int [gpiod\\_line\\_request\\_bulk\\_input\\_flags](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer, int flags)  
[GPIO\\_API](#)  
*Reserve a set of GPIO lines, set the direction to input.*
- int [gpiod\\_line\\_request\\_bulk\\_output\\_flags](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer, int flags, const  
int \*default\_↵vals) [GPIO\\_API](#)  
*Reserve a set of GPIO lines, set the direction to output.*
- int [gpiod\\_line\\_request\\_bulk\\_rising\\_edge\\_events\\_flags](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer,  
int flags) [GPIO\\_API](#)  
*Request rising edge event notifications on a set of lines.*
- int [gpiod\\_line\\_request\\_bulk\\_falling\\_edge\\_events\\_flags](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer,  
int flags) [GPIO\\_API](#)  
*Request falling edge event notifications on a set of lines.*
- int [gpiod\\_line\\_request\\_bulk\\_both\\_edges\\_events\\_flags](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const char \*consumer,  
int flags) [GPIO\\_API](#)  
*Request all event type notifications on a set of lines.*
- void [gpiod\\_line\\_release](#) (struct [gpiod\\_line](#) \*line) [GPIO\\_API](#)  
*Release a previously reserved line.*
- void [gpiod\\_line\\_release\\_bulk](#) (struct [gpiod\\_line\\_bulk](#) \*bulk) [GPIO\\_API](#)  
*Release a set of previously reserved lines.*
- bool [gpiod\\_line\\_is\\_requested](#) (struct [gpiod\\_line](#) \*line) [GPIO\\_API](#)  
*Check if the calling user has ownership of this line.*
- bool [gpiod\\_line\\_is\\_free](#) (struct [gpiod\\_line](#) \*line) [GPIO\\_API](#)  
*Check if the calling user has neither requested ownership of this line nor configured any event notifications.*
- int [gpiod\\_line\\_get\\_value](#) (struct [gpiod\\_line](#) \*line) [GPIO\\_API](#)  
*Read current value of a single GPIO line.*
- int [gpiod\\_line\\_get\\_value\\_bulk](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, int \*values) [GPIO\\_API](#)  
*Read current values of a set of GPIO lines.*
- int [gpiod\\_line\\_set\\_value](#) (struct [gpiod\\_line](#) \*line, int value) [GPIO\\_API](#)  
*Set the value of a single GPIO line.*
- int [gpiod\\_line\\_set\\_value\\_bulk](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const int \*values) [GPIO\\_API](#)  
*Set the values of a set of GPIO lines.*
- int [gpiod\\_line\\_set\\_config](#) (struct [gpiod\\_line](#) \*line, int direction, int flags, int value) [GPIO\\_API](#)  
*Update the configuration of a single GPIO line.*
- int [gpiod\\_line\\_set\\_config\\_bulk](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, int direction, int flags, const int \*values)  
[GPIO\\_API](#)  
*Update the configuration of a set of GPIO lines.*
- int [gpiod\\_line\\_set\\_flags](#) (struct [gpiod\\_line](#) \*line, int flags) [GPIO\\_API](#)  
*Update the configuration flags of a single GPIO line.*

- int [gpiod\\_line\\_set\\_flags\\_bulk](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, int flags) [GPIO\\_API](#)  
*Update the configuration flags of a set of GPIO lines.*
- int [gpiod\\_line\\_set\\_direction\\_input](#) (struct [gpiod\\_line](#) \*line) [GPIO\\_API](#)  
*Set the direction of a single GPIO line to input.*
- int [gpiod\\_line\\_set\\_direction\\_input\\_bulk](#) (struct [gpiod\\_line\\_bulk](#) \*bulk) [GPIO\\_API](#)  
*Set the direction of a set of GPIO lines to input.*
- int [gpiod\\_line\\_set\\_direction\\_output](#) (struct [gpiod\\_line](#) \*line, int value) [GPIO\\_API](#)  
*Set the direction of a single GPIO line to output.*
- int [gpiod\\_line\\_set\\_direction\\_output\\_bulk](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const int \*values) [GPIO\\_API](#)  
*Set the direction of a set of GPIO lines to output.*
- int [gpiod\\_line\\_event\\_wait](#) (struct [gpiod\\_line](#) \*line, const struct timespec \*timeout) [GPIO\\_API](#)  
*Wait for an event on a single line.*
- int [gpiod\\_line\\_event\\_wait\\_bulk](#) (struct [gpiod\\_line\\_bulk](#) \*bulk, const struct timespec \*timeout, struct [gpiod\\_line\\_bulk](#) \*event\_bulk) [GPIO\\_API](#)  
*Wait for events on a set of lines.*
- int [gpiod\\_line\\_event\\_read](#) (struct [gpiod\\_line](#) \*line, struct [gpiod\\_line\\_event](#) \*event) [GPIO\\_API](#)  
*Read next pending event from the GPIO line.*
- int [gpiod\\_line\\_event\\_read\\_multiple](#) (struct [gpiod\\_line](#) \*line, struct [gpiod\\_line\\_event](#) \*events, unsigned int num\_events) [GPIO\\_API](#)  
*Read up to a certain number of events from the GPIO line.*
- int [gpiod\\_line\\_event\\_get\\_fd](#) (struct [gpiod\\_line](#) \*line) [GPIO\\_API](#)  
*Get the event file descriptor.*
- int [gpiod\\_line\\_event\\_read\\_fd](#) (int fd, struct [gpiod\\_line\\_event](#) \*event) [GPIO\\_API](#)  
*Read the last GPIO event directly from a file descriptor.*
- int [gpiod\\_line\\_event\\_read\\_fd\\_multiple](#) (int fd, struct [gpiod\\_line\\_event](#) \*events, unsigned int num\_events) [GPIO\\_API](#)  
*Read up to a certain number of events directly from a file descriptor.*
- struct [gpiod\\_line](#) \* [gpiod\\_line\\_get](#) (const char \*device, unsigned int offset) [GPIO\\_API](#)  
*Get a GPIO line handle by GPIO chip description and offset.*
- struct [gpiod\\_line](#) \* [gpiod\\_line\\_find](#) (const char \*name) [GPIO\\_API](#)  
*Find a GPIO line by its name.*
- void [gpiod\\_line\\_close\\_chip](#) (struct [gpiod\\_line](#) \*line) [GPIO\\_API](#)  
*Close a GPIO chip owning this line and release all resources.*
- struct [gpiod\\_chip](#) \* [gpiod\\_line\\_get\\_chip](#) (struct [gpiod\\_line](#) \*line) [GPIO\\_API](#)  
*Get the handle to the GPIO chip controlling this line.*
- struct [gpiod\\_chip\\_iter](#) \* [gpiod\\_chip\\_iter\\_new](#) (void) [GPIO\\_API](#)  
*Create a new gpiochip iterator.*
- void [gpiod\\_chip\\_iter\\_free](#) (struct [gpiod\\_chip\\_iter](#) \*iter) [GPIO\\_API](#)  
*Release all resources allocated for the gpiochip iterator and close the most recently opened gpiochip (if any).*
- void [gpiod\\_chip\\_iter\\_free\\_noclose](#) (struct [gpiod\\_chip\\_iter](#) \*iter) [GPIO\\_API](#)  
*Release all resources allocated for the gpiochip iterator but don't close the most recently opened gpiochip (if any).*
- struct [gpiod\\_chip](#) \* [gpiod\\_chip\\_iter\\_next](#) (struct [gpiod\\_chip\\_iter](#) \*iter) [GPIO\\_API](#)  
*Get the next gpiochip handle.*
- struct [gpiod\\_chip](#) \* [gpiod\\_chip\\_iter\\_next\\_noclose](#) (struct [gpiod\\_chip\\_iter](#) \*iter) [GPIO\\_API](#)  
*Get the next gpiochip handle without closing the previous one.*
- struct [gpiod\\_line\\_iter](#) \* [gpiod\\_line\\_iter\\_new](#) (struct [gpiod\\_chip](#) \*chip) [GPIO\\_API](#)  
*Create a new line iterator.*
- void [gpiod\\_line\\_iter\\_free](#) (struct [gpiod\\_line\\_iter](#) \*iter) [GPIO\\_API](#)  
*Free all resources associated with a GPIO line iterator.*
- struct [gpiod\\_line](#) \* [gpiod\\_line\\_iter\\_next](#) (struct [gpiod\\_line\\_iter](#) \*iter) [GPIO\\_API](#)  
*Get the next GPIO line handle.*
- const char \* [gpiod\\_version\\_string](#) (void) [GPIO\\_API](#)  
*Get the API version of the library as a human-readable string.*



## 8.2 gpiod.h

[Go to the documentation of this file.](#)

```

00001 /* SPDX-License-Identifier: LGPL-2.1-or-later */
00002 /*
00003  * This file is part of libgpiod.
00004  *
00005  * Copyright (C) 2017-2018 Bartosz Golaszewski <bartekgola@gmail.com>
00006  */
00007
00008 #ifndef __LIBGPIOD_GPIOD_H__
00009 #define __LIBGPIOD_GPIOD_H__
00010
00011 #include <stdbool.h>
00012 #include <stdlib.h>
00013 #include <time.h>
00014
00015 #ifdef __cplusplus
00016 extern "C" {
00017 #endif
00018
00046 struct gpiod_chip;
00047 struct gpiod_line;
00048 struct gpiod_chip_iter;
00049 struct gpiod_line_iter;
00050 struct gpiod_line_bulk;
00051
00062 #define GPIOD_API      __attribute__((visibility("default")))
00063
00067 #define GPIOD_UNUSED   __attribute__((unused))
00068
00074 #define GPIOD_BIT(nr)  (1UL < (nr))
00075
00079 #define GPIOD_DEPRECATED __attribute__((deprecated))
00080
00094 enum {
00095     GPIOD_CTXLESS_FLAG_OPEN_DRAIN    = GPIOD_BIT(0),
00097     GPIOD_CTXLESS_FLAG_OPEN_SOURCE   = GPIOD_BIT(1),
00099     GPIOD_CTXLESS_FLAG_BIAS_DISABLE  = GPIOD_BIT(2),
00101     GPIOD_CTXLESS_FLAG_BIAS_PULL_DOWN = GPIOD_BIT(3),
00103     GPIOD_CTXLESS_FLAG_BIAS_PULL_UP  = GPIOD_BIT(4),
00105 };
00106
00115 int gpiod_ctxless_get_value(const char *device, unsigned int offset,
00116                             bool active_low, const char *consumer) GPIOD_API;
00117
00127 int gpiod_ctxless_get_value_ext(const char *device, unsigned int offset,
00128                                 bool active_low, const char *consumer,
00129                                 int flags) GPIOD_API;
00130
00141 int gpiod_ctxless_get_value_multiple(const char *device,
00142                                     const unsigned int *offsets, int *values,
00143                                     unsigned int num_lines, bool active_low,
00144                                     const char *consumer) GPIOD_API;
00145
00157 int gpiod_ctxless_get_value_multiple_ext(const char *device,
00158                                           const unsigned int *offsets,
00159                                           int *values, unsigned int num_lines,
00160                                           bool active_low, const char *consumer,
00161                                           int flags) GPIOD_API;
00162
00166 typedef void (*gpiod_ctxless_set_value_cb)(void *);
00167
00181 int gpiod_ctxless_set_value(const char *device, unsigned int offset, int value,
00182                             bool active_low, const char *consumer,
00183                             gpiod_ctxless_set_value_cb cb,
00184                             void *data) GPIOD_API;
00185
00200 int gpiod_ctxless_set_value_ext(const char *device, unsigned int offset,
00201                                 int value, bool active_low,
00202                                 const char *consumer,
00203                                 gpiod_ctxless_set_value_cb cb,
00204                                 void *data, int flags) GPIOD_API;
00205
00219 int gpiod_ctxless_set_value_multiple(const char *device,
00220                                     const unsigned int *offsets,
00221                                     const int *values, unsigned int num_lines,
00222                                     bool active_low, const char *consumer,
00223                                     gpiod_ctxless_set_value_cb cb,
00224                                     void *data) GPIOD_API;
00225
00240 int gpiod_ctxless_set_value_multiple_ext(const char *device,
00241                                           const unsigned int *offsets,
00242                                           const int *values,
00243                                           unsigned int num_lines,

```



```

00244         bool active_low,
00245         const char *consumer,
00246         gpiod_ctxless_set_value_cb cb,
00247         void *data, int flags) GPIOD_API;
00248
00252     enum {
00254         GPIOD_CTXLESS_EVENT_RISING_EDGE = 1,
00256         GPIOD_CTXLESS_EVENT_FALLING_EDGE,
00258         GPIOD_CTXLESS_EVENT_BOTH_EDGES,
00259     };
00260
00264     enum {
00265         GPIOD_CTXLESS_EVENT_CB_TIMEOUT = 1,
00267         GPIOD_CTXLESS_EVENT_CB_RISING_EDGE,
00269         GPIOD_CTXLESS_EVENT_CB_FALLING_EDGE,
00271     };
00272
00276     enum {
00277         GPIOD_CTXLESS_EVENT_CB_RET_ERR = -1,
00279         GPIOD_CTXLESS_EVENT_CB_RET_OK = 0,
00281         GPIOD_CTXLESS_EVENT_CB_RET_STOP = 1,
00283     };
00284
00296     typedef int (*gpiod_ctxless_event_handle_cb)(int, unsigned int,
00297         const struct timespec *, void *);
00298
00305     enum {
00306         GPIOD_CTXLESS_EVENT_POLL_RET_STOP = -2,
00308         GPIOD_CTXLESS_EVENT_POLL_RET_ERR = -1,
00310         GPIOD_CTXLESS_EVENT_POLL_RET_TIMEOUT = 0,
00312     };
00313
00317     struct gpiod_ctxless_event_poll_fd {
00318         int fd;
00320         bool event;
00322     };
00323
00336     typedef int (*gpiod_ctxless_event_poll_cb)(unsigned int,
00337         struct gpiod_ctxless_event_poll_fd *,
00338         const struct timespec *, void *);
00339
00358     int gpiod_ctxless_event_loop(const char *device, unsigned int offset,
00359         bool active_low, const char *consumer,
00360         const struct timespec *timeout,
00361         gpiod_ctxless_event_poll_cb poll_cb,
00362         gpiod_ctxless_event_handle_cb event_cb,
00363         void *data) GPIOD_API GPIOD_DEPRECATED;
00364
00396     int gpiod_ctxless_event_loop_multiple(const char *device,
00397         const unsigned int *offsets,
00398         unsigned int num_lines, bool active_low,
00399         const char *consumer,
00400         const struct timespec *timeout,
00401         gpiod_ctxless_event_poll_cb poll_cb,
00402         gpiod_ctxless_event_handle_cb event_cb,
00403         void *data) GPIOD_API GPIOD_DEPRECATED;
00404
00421     int gpiod_ctxless_event_monitor(const char *device, int event_type,
00422         unsigned int offset, bool active_low,
00423         const char *consumer,
00424         const struct timespec *timeout,
00425         gpiod_ctxless_event_poll_cb poll_cb,
00426         gpiod_ctxless_event_handle_cb event_cb,
00427         void *data) GPIOD_API;
00428
00446     int gpiod_ctxless_event_monitor_ext(const char *device, int event_type,
00447         unsigned int offset, bool active_low,
00448         const char *consumer,
00449         const struct timespec *timeout,
00450         gpiod_ctxless_event_poll_cb poll_cb,
00451         gpiod_ctxless_event_handle_cb event_cb,
00452         void *data, int flags) GPIOD_API;
00453
00484     int gpiod_ctxless_event_monitor_multiple(
00485         const char *device, int event_type,
00486         const unsigned int *offsets,
00487         unsigned int num_lines, bool active_low,
00488         const char *consumer, const struct timespec *timeout,
00489         gpiod_ctxless_event_poll_cb poll_cb,
00490         gpiod_ctxless_event_handle_cb event_cb,
00491         void *data) GPIOD_API;
00492
00524     int gpiod_ctxless_event_monitor_multiple_ext(
00525         const char *device, int event_type,
00526         const unsigned int *offsets,
00527         unsigned int num_lines, bool active_low,
00528         const char *consumer, const struct timespec *timeout,

```

```

00529         gpiod_ctxless_event_poll_cb poll_cb,
00530         gpiod_ctxless_event_handle_cb event_cb,
00531         void *data, int flags) GPIOD_API;
00532
00533
00548 int gpiod_ctxless_find_line(const char *name, char *chipname,
00549                             size_t chipname_size,
00550                             unsigned int *offset) GPIOD_API;
00551
00566 struct gpiod_chip *gpiod_chip_open(const char *path) GPIOD_API;
00567
00575 struct gpiod_chip *gpiod_chip_open_by_name(const char *name) GPIOD_API;
00576
00584 struct gpiod_chip *gpiod_chip_open_by_number(unsigned int num) GPIOD_API;
00585
00594 struct gpiod_chip *gpiod_chip_open_by_label(const char *label) GPIOD_API;
00595
00605 struct gpiod_chip *gpiod_chip_open_lookup(const char *descr) GPIOD_API;
00606
00611 void gpiod_chip_close(struct gpiod_chip *chip) GPIOD_API;
00612
00618 const char *gpiod_chip_name(struct gpiod_chip *chip) GPIOD_API;
00619
00625 const char *gpiod_chip_label(struct gpiod_chip *chip) GPIOD_API;
00626
00632 unsigned int gpiod_chip_num_lines(struct gpiod_chip *chip) GPIOD_API;
00633
00640 struct gpiod_line *
00641 gpiod_chip_get_line(struct gpiod_chip *chip, unsigned int offset) GPIOD_API;
00642
00651 int gpiod_chip_get_lines(struct gpiod_chip *chip,
00652                          unsigned int *offsets, unsigned int num_offsets,
00653                          struct gpiod_line_bulk *bulk) GPIOD_API;
00654
00661 int gpiod_chip_get_all_lines(struct gpiod_chip *chip,
00662                             struct gpiod_line_bulk *bulk) GPIOD_API;
00663
00676 struct gpiod_line *
00677 gpiod_chip_find_line(struct gpiod_chip *chip, const char *name) GPIOD_API;
00678
00692 int gpiod_chip_find_lines(struct gpiod_chip *chip, const char **names,
00693                          struct gpiod_line_bulk *bulk) GPIOD_API;
00694
00713 #define GPIOD_LINE_BULK_MAX_LINES    64
00714
00722 struct gpiod_line_bulk {
00723     struct gpiod_line *lines[GPIOD_LINE_BULK_MAX_LINES];
00724     unsigned int num_lines;
00725 };
00726
00734 #define GPIOD_LINE_BULK_INITIALIZER { { NULL }, 0 }
00735
00742 static inline void gpiod_line_bulk_init(struct gpiod_line_bulk *bulk)
00743 {
00744     bulk->num_lines = 0;
00745 }
00746
00752 static inline void gpiod_line_bulk_add(struct gpiod_line_bulk *bulk,
00753                                       struct gpiod_line *line)
00754 {
00755     bulk->lines[bulk->num_lines++] = line;
00756 }
00757
00764 static inline struct gpiod_line *
00765 gpiod_line_bulk_get_line(struct gpiod_line_bulk *bulk, unsigned int offset)
00766 {
00767     return bulk->lines[offset];
00768 }
00769
00775 static inline unsigned int
00776 gpiod_line_bulk_num_lines(struct gpiod_line_bulk *bulk)
00777 {
00778     return bulk->num_lines;
00779 }
00780
00788 #define gpiod_line_bulk_foreach_line(bulk, line, lineptr) \
00789     for ((lineptr) = (bulk)->lines, (line) = *(lineptr); \
00790          (lineptr) <= (bulk)->lines + ((bulk)->num_lines - 1); \
00791          (lineptr)++, (line) = *(lineptr))
00792
00806 #define gpiod_line_bulk_foreach_line_off(bulk, line, offset) \
00807     for ((offset) = 0, (line) = (bulk)->lines[0]; \
00808          (offset) < (bulk)->num_lines; \
00809          (offset)++, (line) = (bulk)->lines[(offset)])
00810
00824 enum {
00825     GPIOD_LINE_DIRECTION_INPUT = 1,

```

```

00827     GPIOD_LINE_DIRECTION_OUTPUT,
00829 };
00830
00834 enum {
00835     GPIOD_LINE_ACTIVE_STATE_HIGH = 1,
00837     GPIOD_LINE_ACTIVE_STATE_LOW,
00839 };
00840
00844 enum {
00845     GPIOD_LINE_BIAS_AS_IS = 1,
00847     GPIOD_LINE_BIAS_DISABLE,
00849     GPIOD_LINE_BIAS_PULL_UP,
00851     GPIOD_LINE_BIAS_PULL_DOWN,
00853 };
00854
00860 unsigned int gpiod_line_offset(struct gpiod_line *line) GPIOD_API;
00861
00869 const char *gpiod_line_name(struct gpiod_line *line) GPIOD_API;
00870
00878 const char *gpiod_line_consumer(struct gpiod_line *line) GPIOD_API;
00879
00885 int gpiod_line_direction(struct gpiod_line *line) GPIOD_API;
00886
00892 int gpiod_line_active_state(struct gpiod_line *line) GPIOD_API;
00893
00900 int gpiod_line_bias(struct gpiod_line *line) GPIOD_API;
00901
00911 bool gpiod_line_is_used(struct gpiod_line *line) GPIOD_API;
00912
00918 bool gpiod_line_is_open_drain(struct gpiod_line *line) GPIOD_API;
00919
00925 bool gpiod_line_is_open_source(struct gpiod_line *line) GPIOD_API;
00926
00947 int gpiod_line_update(struct gpiod_line *line) GPIOD_API;
00948
00956 bool
00957 gpiod_line_needs_update(struct gpiod_line *line) GPIOD_API GPIOD_DEPRECATED;
00958
00972 enum {
00973     GPIOD_LINE_REQUEST_DIRECTION_AS_IS = 1,
00975     GPIOD_LINE_REQUEST_DIRECTION_INPUT,
00977     GPIOD_LINE_REQUEST_DIRECTION_OUTPUT,
00979     GPIOD_LINE_REQUEST_EVENT_FALLING_EDGE,
00981     GPIOD_LINE_REQUEST_EVENT_RISING_EDGE,
00983     GPIOD_LINE_REQUEST_EVENT_BOTH_EDGES,
00985 };
00986
00990 enum {
00991     GPIOD_LINE_REQUEST_FLAG_OPEN_DRAIN = GPIOD_BIT(0),
00993     GPIOD_LINE_REQUEST_FLAG_OPEN_SOURCE = GPIOD_BIT(1),
00995     GPIOD_LINE_REQUEST_FLAG_ACTIVE_LOW = GPIOD_BIT(2),
00997     GPIOD_LINE_REQUEST_FLAG_BIAS_DISABLE = GPIOD_BIT(3),
00999     GPIOD_LINE_REQUEST_FLAG_BIAS_PULL_DOWN = GPIOD_BIT(4),
01001     GPIOD_LINE_REQUEST_FLAG_BIAS_PULL_UP = GPIOD_BIT(5),
01003 };
01004
01008 struct gpiod_line_request_config {
01009     const char *consumer;
01011     int request_type;
01013     int flags;
01015 };
01016
01029 int gpiod_line_request(struct gpiod_line *line,
01030     const struct gpiod_line_request_config *config,
01031     int default_val) GPIOD_API;
01032
01039 int gpiod_line_request_input(struct gpiod_line *line,
01040     const char *consumer) GPIOD_API;
01041
01049 int gpiod_line_request_output(struct gpiod_line *line,
01050     const char *consumer, int default_val) GPIOD_API;
01051
01058 int gpiod_line_request_rising_edge_events(struct gpiod_line *line,
01059     const char *consumer) GPIOD_API;
01060
01067 int gpiod_line_request_falling_edge_events(struct gpiod_line *line,
01068     const char *consumer) GPIOD_API;
01069
01076 int gpiod_line_request_both_edges_events(struct gpiod_line *line,
01077     const char *consumer) GPIOD_API;
01078
01086 int gpiod_line_request_input_flags(struct gpiod_line *line,
01087     const char *consumer, int flags) GPIOD_API;
01088
01097 int gpiod_line_request_output_flags(struct gpiod_line *line,
01098     const char *consumer, int flags,
01099     int default_val) GPIOD_API;

```

```
01100
01108 int gpiod_line_request_rising_edge_events_flags(struct gpiod_line *line,
01109             const char *consumer,
01110             int flags) GPIOD_API;
01111
01119 int gpiod_line_request_falling_edge_events_flags(struct gpiod_line *line,
01120             const char *consumer,
01121             int flags) GPIOD_API;
01122
01130 int gpiod_line_request_both_edges_events_flags(struct gpiod_line *line,
01131             const char *consumer,
01132             int flags) GPIOD_API;
01133
01147 int gpiod_line_request_bulk(struct gpiod_line_bulk *bulk,
01148             const struct gpiod_line_request_config *config,
01149             const int *default_vals) GPIOD_API;
01150
01157 int gpiod_line_request_bulk_input(struct gpiod_line_bulk *bulk,
01158             const char *consumer) GPIOD_API;
01159
01167 int gpiod_line_request_bulk_output(struct gpiod_line_bulk *bulk,
01168             const char *consumer,
01169             const int *default_vals) GPIOD_API;
01170
01177 int gpiod_line_request_bulk_rising_edge_events(struct gpiod_line_bulk *bulk,
01178             const char *consumer) GPIOD_API;
01179
01186 int gpiod_line_request_bulk_falling_edge_events(struct gpiod_line_bulk *bulk,
01187             const char *consumer) GPIOD_API;
01188
01195 int gpiod_line_request_bulk_both_edges_events(struct gpiod_line_bulk *bulk,
01196             const char *consumer) GPIOD_API;
01197
01205 int gpiod_line_request_bulk_input_flags(struct gpiod_line_bulk *bulk,
01206             const char *consumer,
01207             int flags) GPIOD_API;
01208
01217 int gpiod_line_request_bulk_output_flags(struct gpiod_line_bulk *bulk,
01218             const char *consumer, int flags,
01219             const int *default_vals) GPIOD_API;
01220
01228 int gpiod_line_request_bulk_rising_edge_events_flags(
01229             struct gpiod_line_bulk *bulk,
01230             const char *consumer,
01231             int flags) GPIOD_API;
01232
01240 int gpiod_line_request_bulk_falling_edge_events_flags(
01241             struct gpiod_line_bulk *bulk,
01242             const char *consumer,
01243             int flags) GPIOD_API;
01244
01252 int gpiod_line_request_bulk_both_edges_events_flags(
01253             struct gpiod_line_bulk *bulk,
01254             const char *consumer,
01255             int flags) GPIOD_API;
01256
01261 void gpiod_line_release(struct gpiod_line *line) GPIOD_API;
01262
01270 void gpiod_line_release_bulk(struct gpiod_line_bulk *bulk) GPIOD_API;
01271
01277 bool gpiod_line_is_requested(struct gpiod_line *line) GPIOD_API;
01278
01285 bool gpiod_line_is_free(struct gpiod_line *line) GPIOD_API;
01286
01303 int gpiod_line_get_value(struct gpiod_line *line) GPIOD_API;
01304
01316 int gpiod_line_get_value_bulk(struct gpiod_line_bulk *bulk,
01317             int *values) GPIOD_API;
01318
01326 int gpiod_line_set_value(struct gpiod_line *line, int value) GPIOD_API;
01327
01339 int gpiod_line_set_value_bulk(struct gpiod_line_bulk *bulk,
01340             const int *values) GPIOD_API;
01341
01365 int gpiod_line_set_config(struct gpiod_line *line, int direction,
01366             int flags, int value) GPIOD_API;
01367
01386 int gpiod_line_set_config_bulk(struct gpiod_line_bulk *bulk,
01387             int direction, int flags,
01388             const int *values) GPIOD_API;
01389
01390
01398 int gpiod_line_set_flags(struct gpiod_line *line, int flags) GPIOD_API;
01399
01410 int gpiod_line_set_flags_bulk(struct gpiod_line_bulk *bulk,
01411             int flags) GPIOD_API;
01412
```

```

01419 int gpiod_line_set_direction_input(struct gpiod_line *line) GPIOD_API;
01420
01430 int
01431 gpiod_line_set_direction_input_bulk(struct gpiod_line_bulk *bulk) GPIOD_API;
01432
01440 int gpiod_line_set_direction_output(struct gpiod_line *line,
01441                                     int value) GPIOD_API;
01442
01455 int gpiod_line_set_direction_output_bulk(struct gpiod_line_bulk *bulk,
01456                                           const int *values) GPIOD_API;
01457
01473 enum {
01474     GPIOD_LINE_EVENT_RISING_EDGE = 1,
01476     GPIOD_LINE_EVENT_FALLING_EDGE,
01478 };
01479
01483 struct gpiod_line_event {
01484     struct timespec ts;
01486     int event_type;
01488 };
01489
01497 int gpiod_line_event_wait(struct gpiod_line *line,
01498                           const struct timespec *timeout) GPIOD_API;
01499
01509 int gpiod_line_event_wait_bulk(struct gpiod_line_bulk *bulk,
01510                                const struct timespec *timeout,
01511                                struct gpiod_line_bulk *event_bulk) GPIOD_API;
01512
01520 int gpiod_line_event_read(struct gpiod_line *line,
01521                           struct gpiod_line_event *event) GPIOD_API;
01522
01532 int gpiod_line_event_read_multiple(struct gpiod_line *line,
01533                                    struct gpiod_line_event *events,
01534                                    unsigned int num_events) GPIOD_API;
01535
01546 int gpiod_line_event_get_fd(struct gpiod_line *line) GPIOD_API;
01547
01558 int gpiod_line_event_read_fd(int fd, struct gpiod_line_event *event) GPIOD_API;
01559
01569 int gpiod_line_event_read_fd_multiple(int fd, struct gpiod_line_event *events,
01570                                       unsigned int num_events) GPIOD_API;
01571
01593 struct gpiod_line *
01594 gpiod_line_get(const char *device, unsigned int offset) GPIOD_API;
01595
01609 struct gpiod_line *gpiod_line_find(const char *name) GPIOD_API;
01610
01617 void gpiod_line_close_chip(struct gpiod_line *line) GPIOD_API;
01618
01624 struct gpiod_chip *gpiod_line_get_chip(struct gpiod_line *line) GPIOD_API;
01625
01646 struct gpiod_chip_iter *gpiod_chip_iter_new(void) GPIOD_API;
01647
01653 void gpiod_chip_iter_free(struct gpiod_chip_iter *iter) GPIOD_API;
01654
01664 void gpiod_chip_iter_free_noclose(struct gpiod_chip_iter *iter) GPIOD_API;
01665
01673 struct gpiod_chip *
01674 gpiod_chip_iter_next(struct gpiod_chip_iter *iter) GPIOD_API;
01675
01684 struct gpiod_chip *
01685 gpiod_chip_iter_next_noclose(struct gpiod_chip_iter *iter) GPIOD_API;
01686
01697 #define gpiod_foreach_chip(iter, chip) \
01698     for ((chip) = gpiod_chip_iter_next(iter); \
01699          (chip); \
01700          (chip) = gpiod_chip_iter_next(iter))
01701
01712 #define gpiod_foreach_chip_noclose(iter, chip) \
01713     for ((chip) = gpiod_chip_iter_next_noclose(iter); \
01714          (chip); \
01715          (chip) = gpiod_chip_iter_next_noclose(iter))
01716
01723 struct gpiod_line_iter *
01724 gpiod_line_iter_new(struct gpiod_chip *chip) GPIOD_API;
01725
01730 void gpiod_line_iter_free(struct gpiod_line_iter *iter) GPIOD_API;
01731
01738 struct gpiod_line *
01739 gpiod_line_iter_next(struct gpiod_line_iter *iter) GPIOD_API;
01740
01747 #define gpiod_foreach_line(iter, line) \
01748     for ((line) = gpiod_line_iter_next(iter); \
01749          (line); \
01750          (line) = gpiod_line_iter_next(iter))
01751
01765 const char *gpiod_version_string(void) GPIOD_API;

```

```
01766
01771 #ifdef __cplusplus
01772 } /* extern "C" */
01773 #endif
01774
01775 #endif /* __LIBGPIOD_GPIOD_H__ */
```

# Index

- Common helper macros, [11](#)
  - GPIOD\_BIT, [11](#)
- consumer
  - gpiod\_line\_request\_config, [73](#)
- Deprecated List, [3](#)
- event
  - gpiod\_ctxless\_event\_poll\_fd, [71](#)
- event\_type
  - gpiod\_line\_event, [73](#)
- fd
  - gpiod\_ctxless\_event\_poll\_fd, [71](#)
- flags
  - gpiod\_line\_request\_config, [73](#)
- GPIO chip operations, [12](#)
  - gpiod\_chip\_close, [12](#)
  - gpiod\_chip\_find\_line, [13](#)
  - gpiod\_chip\_find\_lines, [13](#)
  - gpiod\_chip\_get\_all\_lines, [14](#)
  - gpiod\_chip\_get\_line, [14](#)
  - gpiod\_chip\_get\_lines, [14](#)
  - gpiod\_chip\_label, [15](#)
  - gpiod\_chip\_name, [15](#)
  - gpiod\_chip\_num\_lines, [16](#)
  - gpiod\_chip\_open, [16](#)
  - gpiod\_chip\_open\_by\_label, [16](#)
  - gpiod\_chip\_open\_by\_name, [17](#)
  - gpiod\_chip\_open\_by\_number, [17](#)
  - gpiod\_chip\_open\_lookup, [17](#)
- GPIO line operations, [18](#)
- gpiod.h, [75](#)
- GPIOD\_BIT
  - Common helper macros, [11](#)
- gpiod\_chip\_close
  - GPIO chip operations, [12](#)
- gpiod\_chip\_find\_line
  - GPIO chip operations, [13](#)
- gpiod\_chip\_find\_lines
  - GPIO chip operations, [13](#)
- gpiod\_chip\_get\_all\_lines
  - GPIO chip operations, [14](#)
- gpiod\_chip\_get\_line
  - GPIO chip operations, [14](#)
- gpiod\_chip\_get\_lines
  - GPIO chip operations, [14](#)
- gpiod\_chip\_iter\_free
  - Iterators for GPIO chips and lines, [67](#)
- gpiod\_chip\_iter\_free\_noclose
  - Iterators for GPIO chips and lines, [67](#)
- gpiod\_chip\_iter\_new
  - Iterators for GPIO chips and lines, [67](#)
- gpiod\_chip\_iter\_next
  - Iterators for GPIO chips and lines, [67](#)
- gpiod\_chip\_iter\_next\_noclose
  - Iterators for GPIO chips and lines, [68](#)
- gpiod\_chip\_label
  - GPIO chip operations, [15](#)
- gpiod\_chip\_name
  - GPIO chip operations, [15](#)
- gpiod\_chip\_num\_lines
  - GPIO chip operations, [16](#)
- gpiod\_chip\_open
  - GPIO chip operations, [16](#)
- gpiod\_chip\_open\_by\_label
  - GPIO chip operations, [16](#)
- gpiod\_chip\_open\_by\_name
  - GPIO chip operations, [17](#)
- gpiod\_chip\_open\_by\_number
  - GPIO chip operations, [17](#)
- gpiod\_chip\_open\_lookup
  - GPIO chip operations, [17](#)
- GPIOD\_CTXLESS\_EVENT\_CB\_FALLING\_EDGE
  - High-level API, [54](#)
- GPIOD\_CTXLESS\_EVENT\_CB\_RET\_ERR
  - High-level API, [54](#)
- GPIOD\_CTXLESS\_EVENT\_CB\_RET\_OK
  - High-level API, [54](#)
- GPIOD\_CTXLESS\_EVENT\_CB\_RET\_STOP
  - High-level API, [54](#)
- GPIOD\_CTXLESS\_EVENT\_CB\_RISING\_EDGE
  - High-level API, [54](#)
- GPIOD\_CTXLESS\_EVENT\_CB\_TIMEOUT
  - High-level API, [54](#)
- GPIOD\_CTXLESS\_EVENT\_FALLING\_EDGE
  - High-level API, [54](#)
- gpiod\_ctxless\_event\_handle\_cb
  - High-level API, [53](#)
- gpiod\_ctxless\_event\_loop
  - High-level API, [55](#)
- gpiod\_ctxless\_event\_loop\_multiple
  - High-level API, [55](#)
- gpiod\_ctxless\_event\_monitor
  - High-level API, [56](#)
- gpiod\_ctxless\_event\_monitor\_ext
  - High-level API, [57](#)
- gpiod\_ctxless\_event\_monitor\_multiple

- High-level API, [58](#)
- `gpiod_ctxless_event_monitor_multiple_ext`
  - High-level API, [59](#)
- `gpiod_ctxless_event_poll_cb`
  - High-level API, [53](#)
- `gpiod_ctxless_event_poll_fd`, [71](#)
  - event, [71](#)
  - fd, [71](#)
- `GPIOD_CTXLESS_EVENT_POLL_RET_ERR`
  - High-level API, [55](#)
- `GPIOD_CTXLESS_EVENT_POLL_RET_STOP`
  - High-level API, [55](#)
- `GPIOD_CTXLESS_EVENT_POLL_RET_TIMEOUT`
  - High-level API, [55](#)
- `GPIOD_CTXLESS_EVENT_RISING_EDGE`
  - High-level API, [54](#)
- `gpiod_ctxless_find_line`
  - High-level API, [60](#)
- `GPIOD_CTXLESS_FLAG_BIAS_DISABLE`
  - High-level API, [54](#)
- `GPIOD_CTXLESS_FLAG_BIAS_PULL_DOWN`
  - High-level API, [54](#)
- `GPIOD_CTXLESS_FLAG_BIAS_PULL_UP`
  - High-level API, [54](#)
- `GPIOD_CTXLESS_FLAG_OPEN_DRAIN`
  - High-level API, [54](#)
- `GPIOD_CTXLESS_FLAG_OPEN_SOURCE`
  - High-level API, [54](#)
- `gpiod_ctxless_get_value`
  - High-level API, [60](#)
- `gpiod_ctxless_get_value_ext`
  - High-level API, [61](#)
- `gpiod_ctxless_get_value_multiple`
  - High-level API, [61](#)
- `gpiod_ctxless_get_value_multiple_ext`
  - High-level API, [62](#)
- `gpiod_ctxless_set_value`
  - High-level API, [62](#)
- `gpiod_ctxless_set_value_ext`
  - High-level API, [63](#)
- `gpiod_ctxless_set_value_multiple`
  - High-level API, [63](#)
- `gpiod_ctxless_set_value_multiple_ext`
  - High-level API, [64](#)
- `gpiod_foreach_chip`
  - Iterators for GPIO chips and lines, [66](#)
- `gpiod_foreach_chip_noclose`
  - Iterators for GPIO chips and lines, [66](#)
- `gpiod_foreach_line`
  - Iterators for GPIO chips and lines, [66](#)
- `gpiod_line_active_state`
  - Line info, [24](#)
- `GPIOD_LINE_ACTIVE_STATE_HIGH`
  - Line info, [24](#)
- `GPIOD_LINE_ACTIVE_STATE_LOW`
  - Line info, [24](#)
- `gpiod_line_bias`
  - Line info, [24](#)
- `GPIOD_LINE_BIAS_AS_IS`
  - Line info, [24](#)
- `GPIOD_LINE_BIAS_DISABLE`
  - Line info, [24](#)
- `GPIOD_LINE_BIAS_PULL_DOWN`
  - Line info, [24](#)
- `GPIOD_LINE_BIAS_PULL_UP`
  - Line info, [24](#)
- `gpiod_line_bulk`, [72](#)
  - lines, [72](#)
  - num\_lines, [72](#)
- `gpiod_line_bulk_add`
  - Operating on multiple lines, [44](#)
- `gpiod_line_bulk_foreach_line`
  - Operating on multiple lines, [43](#)
- `gpiod_line_bulk_foreach_line_off`
  - Operating on multiple lines, [43](#)
- `gpiod_line_bulk_get_line`
  - Operating on multiple lines, [44](#)
- `gpiod_line_bulk_init`
  - Operating on multiple lines, [45](#)
- `GPIOD_LINE_BULK_INITIALIZER`
  - Operating on multiple lines, [44](#)
- `gpiod_line_bulk_num_lines`
  - Operating on multiple lines, [45](#)
- `gpiod_line_close_chip`
  - Misc line functions, [41](#)
- `gpiod_line_consumer`
  - Line info, [25](#)
- `gpiod_line_direction`
  - Line info, [25](#)
- `GPIOD_LINE_DIRECTION_INPUT`
  - Line info, [23](#)
- `GPIOD_LINE_DIRECTION_OUTPUT`
  - Line info, [23](#)
- `gpiod_line_event`, [72](#)
  - event\_type, [73](#)
  - ts, [73](#)
- `GPIOD_LINE_EVENT_FALLING_EDGE`
  - Line events handling, [19](#)
- `gpiod_line_event_get_fd`
  - Line events handling, [19](#)
- `gpiod_line_event_read`
  - Line events handling, [20](#)
- `gpiod_line_event_read_fd`
  - Line events handling, [20](#)
- `gpiod_line_event_read_fd_multiple`
  - Line events handling, [20](#)
- `gpiod_line_event_read_multiple`
  - Line events handling, [21](#)
- `GPIOD_LINE_EVENT_RISING_EDGE`
  - Line events handling, [19](#)
- `gpiod_line_event_wait`
  - Line events handling, [21](#)
- `gpiod_line_event_wait_bulk`
  - Line events handling, [22](#)
- `gpiod_line_find`
  - Misc line functions, [41](#)



- gpiod\_line\_get
  - Misc line functions, [41](#)
- gpiod\_line\_get\_chip
  - Misc line functions, [42](#)
- gpiod\_line\_get\_value
  - Reading & setting line values, [46](#)
- gpiod\_line\_get\_value\_bulk
  - Reading & setting line values, [46](#)
- gpiod\_line\_is\_free
  - Line requests, [31](#)
- gpiod\_line\_is\_open\_drain
  - Line info, [25](#)
- gpiod\_line\_is\_open\_source
  - Line info, [26](#)
- gpiod\_line\_is\_requested
  - Line requests, [31](#)
- gpiod\_line\_is\_used
  - Line info, [26](#)
- gpiod\_line\_iter\_free
  - Iterators for GPIO chips and lines, [68](#)
- gpiod\_line\_iter\_new
  - Iterators for GPIO chips and lines, [68](#)
- gpiod\_line\_iter\_next
  - Iterators for GPIO chips and lines, [69](#)
- gpiod\_line\_name
  - Line info, [26](#)
- gpiod\_line\_needs\_update
  - Line info, [27](#)
- gpiod\_line\_offset
  - Line info, [27](#)
- gpiod\_line\_release
  - Line requests, [31](#)
- gpiod\_line\_release\_bulk
  - Line requests, [32](#)
- gpiod\_line\_request
  - Line requests, [32](#)
- gpiod\_line\_request\_both\_edges\_events
  - Line requests, [32](#)
- gpiod\_line\_request\_both\_edges\_events\_flags
  - Line requests, [33](#)
- gpiod\_line\_request\_bulk
  - Line requests, [33](#)
- gpiod\_line\_request\_bulk\_both\_edges\_events
  - Line requests, [33](#)
- gpiod\_line\_request\_bulk\_both\_edges\_events\_flags
  - Line requests, [34](#)
- gpiod\_line\_request\_bulk\_falling\_edge\_events
  - Line requests, [34](#)
- gpiod\_line\_request\_bulk\_falling\_edge\_events\_flags
  - Line requests, [35](#)
- gpiod\_line\_request\_bulk\_input
  - Line requests, [35](#)
- gpiod\_line\_request\_bulk\_input\_flags
  - Line requests, [35](#)
- gpiod\_line\_request\_bulk\_output
  - Line requests, [36](#)
- gpiod\_line\_request\_bulk\_output\_flags
  - Line requests, [36](#)
- gpiod\_line\_request\_bulk\_rising\_edge\_events
  - Line requests, [36](#)
- gpiod\_line\_request\_bulk\_rising\_edge\_events\_flags
  - Line requests, [37](#)
- gpiod\_line\_request\_config, [73](#)
  - consumer, [73](#)
  - flags, [73](#)
  - request\_type, [74](#)
- GPIOD\_LINE\_REQUEST\_DIRECTION\_AS\_IS
  - Line requests, [30](#)
- GPIOD\_LINE\_REQUEST\_DIRECTION\_INPUT
  - Line requests, [30](#)
- GPIOD\_LINE\_REQUEST\_DIRECTION\_OUTPUT
  - Line requests, [30](#)
- GPIOD\_LINE\_REQUEST\_EVENT\_BOTH\_EDGES
  - Line requests, [30](#)
- GPIOD\_LINE\_REQUEST\_EVENT\_FALLING\_EDGE
  - Line requests, [30](#)
- GPIOD\_LINE\_REQUEST\_EVENT\_RISING\_EDGE
  - Line requests, [30](#)
- gpiod\_line\_request\_falling\_edge\_events
  - Line requests, [37](#)
- gpiod\_line\_request\_falling\_edge\_events\_flags
  - Line requests, [38](#)
- GPIOD\_LINE\_REQUEST\_FLAG\_ACTIVE\_LOW
  - Line requests, [31](#)
- GPIOD\_LINE\_REQUEST\_FLAG\_BIAS\_DISABLE
  - Line requests, [31](#)
- GPIOD\_LINE\_REQUEST\_FLAG\_BIAS\_PULL\_DOWN
  - Line requests, [31](#)
- GPIOD\_LINE\_REQUEST\_FLAG\_BIAS\_PULL\_UP
  - Line requests, [31](#)
- GPIOD\_LINE\_REQUEST\_FLAG\_OPEN\_DRAIN
  - Line requests, [31](#)
- GPIOD\_LINE\_REQUEST\_FLAG\_OPEN\_SOURCE
  - Line requests, [31](#)
- gpiod\_line\_request\_input
  - Line requests, [38](#)
- gpiod\_line\_request\_input\_flags
  - Line requests, [38](#)
- gpiod\_line\_request\_output
  - Line requests, [39](#)
- gpiod\_line\_request\_output\_flags
  - Line requests, [39](#)
- gpiod\_line\_request\_rising\_edge\_events
  - Line requests, [39](#)
- gpiod\_line\_request\_rising\_edge\_events\_flags
  - Line requests, [40](#)
- gpiod\_line\_set\_config
  - Setting line configuration, [48](#)
- gpiod\_line\_set\_config\_bulk
  - Setting line configuration, [48](#)
- gpiod\_line\_set\_direction\_input
  - Setting line configuration, [48](#)
- gpiod\_line\_set\_direction\_input\_bulk
  - Setting line configuration, [49](#)
- gpiod\_line\_set\_direction\_output
  - Setting line configuration, [49](#)

- gpiod\_line\_set\_direction\_output\_bulk
    - Setting line configuration, 49
  - gpiod\_line\_set\_flags
    - Setting line configuration, 50
  - gpiod\_line\_set\_flags\_bulk
    - Setting line configuration, 50
  - gpiod\_line\_set\_value
    - Reading & setting line values, 46
  - gpiod\_line\_set\_value\_bulk
    - Reading & setting line values, 47
  - gpiod\_line\_update
    - Line info, 27
  - gpiod\_version\_string
    - Stuff that didn't fit anywhere else, 69
- High-level API, 51
  - GPIOD\_CTXLESS\_EVENT\_CB\_FALLING\_EDGE, 54
  - GPIOD\_CTXLESS\_EVENT\_CB\_RET\_ERR, 54
  - GPIOD\_CTXLESS\_EVENT\_CB\_RET\_OK, 54
  - GPIOD\_CTXLESS\_EVENT\_CB\_RET\_STOP, 54
  - GPIOD\_CTXLESS\_EVENT\_CB\_RISING\_EDGE, 54
  - GPIOD\_CTXLESS\_EVENT\_CB\_TIMEOUT, 54
  - GPIOD\_CTXLESS\_EVENT\_FALLING\_EDGE, 54
  - gpiod\_ctxless\_event\_handle\_cb, 53
  - gpiod\_ctxless\_event\_loop, 55
  - gpiod\_ctxless\_event\_loop\_multiple, 55
  - gpiod\_ctxless\_event\_monitor, 56
  - gpiod\_ctxless\_event\_monitor\_ext, 57
  - gpiod\_ctxless\_event\_monitor\_multiple, 58
  - gpiod\_ctxless\_event\_monitor\_multiple\_ext, 59
  - gpiod\_ctxless\_event\_poll\_cb, 53
  - GPIOD\_CTXLESS\_EVENT\_POLL\_RET\_ERR, 55
  - GPIOD\_CTXLESS\_EVENT\_POLL\_RET\_STOP, 55
  - GPIOD\_CTXLESS\_EVENT\_POLL\_RET\_TIMEOUT, 55
  - GPIOD\_CTXLESS\_EVENT\_RISING\_EDGE, 54
  - gpiod\_ctxless\_find\_line, 60
  - GPIOD\_CTXLESS\_FLAG\_BIAS\_DISABLE, 54
  - GPIOD\_CTXLESS\_FLAG\_BIAS\_PULL\_DOWN, 54
  - GPIOD\_CTXLESS\_FLAG\_BIAS\_PULL\_UP, 54
  - GPIOD\_CTXLESS\_FLAG\_OPEN\_DRAIN, 54
  - GPIOD\_CTXLESS\_FLAG\_OPEN\_SOURCE, 54
  - gpiod\_ctxless\_get\_value, 60
  - gpiod\_ctxless\_get\_value\_ext, 61
  - gpiod\_ctxless\_get\_value\_multiple, 61
  - gpiod\_ctxless\_get\_value\_multiple\_ext, 62
  - gpiod\_ctxless\_set\_value, 62
  - gpiod\_ctxless\_set\_value\_ext, 63
  - gpiod\_ctxless\_set\_value\_multiple, 63
  - gpiod\_ctxless\_set\_value\_multiple\_ext, 64
- Iterators for GPIO chips and lines, 65
  - gpiod\_chip\_iter\_free, 67
  - gpiod\_chip\_iter\_free\_noclose, 67
  - gpiod\_chip\_iter\_new, 67
  - gpiod\_chip\_iter\_next, 67
  - gpiod\_chip\_iter\_next\_noclose, 68
  - gpiod\_foreach\_chip, 66
  - gpiod\_foreach\_chip\_noclose, 66
  - gpiod\_foreach\_line, 66
  - gpiod\_line\_iter\_free, 68
  - gpiod\_line\_iter\_new, 68
  - gpiod\_line\_iter\_next, 69
- libgpiod public API, 1
- Line events handling, 18
  - GPIOD\_LINE\_EVENT\_FALLING\_EDGE, 19
  - gpiod\_line\_event\_get\_fd, 19
  - gpiod\_line\_event\_read, 20
  - gpiod\_line\_event\_read\_fd, 20
  - gpiod\_line\_event\_read\_fd\_multiple, 20
  - gpiod\_line\_event\_read\_multiple, 21
  - GPIOD\_LINE\_EVENT\_RISING\_EDGE, 19
  - gpiod\_line\_event\_wait, 21
  - gpiod\_line\_event\_wait\_bulk, 22
- Line info, 22
  - gpiod\_line\_active\_state, 24
  - GPIOD\_LINE\_ACTIVE\_STATE\_HIGH, 24
  - GPIOD\_LINE\_ACTIVE\_STATE\_LOW, 24
  - gpiod\_line\_bias, 24
  - GPIOD\_LINE\_BIAS\_AS\_IS, 24
  - GPIOD\_LINE\_BIAS\_DISABLE, 24
  - GPIOD\_LINE\_BIAS\_PULL\_DOWN, 24
  - GPIOD\_LINE\_BIAS\_PULL\_UP, 24
  - gpiod\_line\_consumer, 25
  - gpiod\_line\_direction, 25
  - GPIOD\_LINE\_DIRECTION\_INPUT, 23
  - GPIOD\_LINE\_DIRECTION\_OUTPUT, 23
  - gpiod\_line\_is\_open\_drain, 25
  - gpiod\_line\_is\_open\_source, 26
  - gpiod\_line\_is\_used, 26
  - gpiod\_line\_name, 26
  - gpiod\_line\_needs\_update, 27
  - gpiod\_line\_offset, 27
  - gpiod\_line\_update, 27
- Line requests, 28
  - gpiod\_line\_is\_free, 31
  - gpiod\_line\_is\_requested, 31
  - gpiod\_line\_release, 31
  - gpiod\_line\_release\_bulk, 32
  - gpiod\_line\_request, 32
  - gpiod\_line\_request\_both\_edges\_events, 32
  - gpiod\_line\_request\_both\_edges\_events\_flags, 33
  - gpiod\_line\_request\_bulk, 33
  - gpiod\_line\_request\_bulk\_both\_edges\_events, 33
  - gpiod\_line\_request\_bulk\_both\_edges\_events\_flags, 34
  - gpiod\_line\_request\_bulk\_falling\_edge\_events, 34
  - gpiod\_line\_request\_bulk\_falling\_edge\_events\_flags, 35
  - gpiod\_line\_request\_bulk\_input, 35
  - gpiod\_line\_request\_bulk\_input\_flags, 35
  - gpiod\_line\_request\_bulk\_output, 36
  - gpiod\_line\_request\_bulk\_output\_flags, 36

- gpiod\_line\_request\_bulk\_rising\_edge\_events, [36](#)
  - gpiod\_line\_request\_bulk\_rising\_edge\_events\_flags, [37](#)
  - GPIOD\_LINE\_REQUEST\_DIRECTION\_AS\_IS, [30](#)
  - GPIOD\_LINE\_REQUEST\_DIRECTION\_INPUT, [30](#)
  - GPIOD\_LINE\_REQUEST\_DIRECTION\_OUTPUT, [30](#)
  - GPIOD\_LINE\_REQUEST\_EVENT\_BOTH\_EDGES, [30](#)
  - GPIOD\_LINE\_REQUEST\_EVENT\_FALLING\_EDGE, [30](#)
  - GPIOD\_LINE\_REQUEST\_EVENT\_RISING\_EDGE, [30](#)
  - gpiod\_line\_request\_falling\_edge\_events, [37](#)
  - gpiod\_line\_request\_falling\_edge\_events\_flags, [38](#)
  - GPIOD\_LINE\_REQUEST\_FLAG\_ACTIVE\_LOW, [31](#)
  - GPIOD\_LINE\_REQUEST\_FLAG\_BIAS\_DISABLE, [31](#)
  - GPIOD\_LINE\_REQUEST\_FLAG\_BIAS\_PULL\_DOWN, [31](#)
  - GPIOD\_LINE\_REQUEST\_FLAG\_BIAS\_PULL\_UP, [31](#)
  - GPIOD\_LINE\_REQUEST\_FLAG\_OPEN\_DRAIN, [31](#)
  - GPIOD\_LINE\_REQUEST\_FLAG\_OPEN\_SOURCE, [31](#)
  - gpiod\_line\_request\_input, [38](#)
  - gpiod\_line\_request\_input\_flags, [38](#)
  - gpiod\_line\_request\_output, [39](#)
  - gpiod\_line\_request\_output\_flags, [39](#)
  - gpiod\_line\_request\_rising\_edge\_events, [39](#)
  - gpiod\_line\_request\_rising\_edge\_events\_flags, [40](#)
- lines
  - gpiod\_line\_bulk, [72](#)
- Misc line functions, [40](#)
  - gpiod\_line\_close\_chip, [41](#)
  - gpiod\_line\_find, [41](#)
  - gpiod\_line\_get, [41](#)
  - gpiod\_line\_get\_chip, [42](#)
- num\_lines
  - gpiod\_line\_bulk, [72](#)
- Operating on multiple lines, [42](#)
  - gpiod\_line\_bulk\_add, [44](#)
  - gpiod\_line\_bulk\_foreach\_line, [43](#)
  - gpiod\_line\_bulk\_foreach\_line\_off, [43](#)
  - gpiod\_line\_bulk\_get\_line, [44](#)
  - gpiod\_line\_bulk\_init, [45](#)
  - GPIOD\_LINE\_BULK\_INITIALIZER, [44](#)
  - gpiod\_line\_bulk\_num\_lines, [45](#)
- Reading & setting line values, [45](#)
  - gpiod\_line\_get\_value, [46](#)
  - gpiod\_line\_get\_value\_bulk, [46](#)
  - gpiod\_line\_set\_value, [46](#)
  - gpiod\_line\_set\_value\_bulk, [47](#)
- request\_type
  - gpiod\_line\_request\_config, [74](#)
- Setting line configuration, [47](#)
  - gpiod\_line\_set\_config, [48](#)
  - gpiod\_line\_set\_config\_bulk, [48](#)
  - gpiod\_line\_set\_direction\_input, [48](#)
  - gpiod\_line\_set\_direction\_input\_bulk, [49](#)
  - gpiod\_line\_set\_direction\_output, [49](#)
  - gpiod\_line\_set\_direction\_output\_bulk, [49](#)
  - gpiod\_line\_set\_flags, [50](#)
  - gpiod\_line\_set\_flags\_bulk, [50](#)
- Stuff that didn't fit anywhere else, [69](#)
  - gpiod\_version\_string, [69](#)
- ts
  - gpiod\_line\_event, [73](#)