

My Project

Generated by Doxygen 1.9.4

1 libgpiod public API	1
2 Deprecated List	3
3 Module Index	5
3.1 Modules	5
4 Class Index	7
4.1 Class List	7
5 File Index	9
5.1 File List	9
6 Module Documentation	11
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 High-level API	12
6.2.1 Detailed Description	14
6.2.2 Typedef Documentation	14
6.2.2.1 gpiod_ctxless_event_handle_cb	14
6.2.2.2 gpiod_ctxless_event_poll_cb	14
6.2.3 Enumeration Type Documentation	14
6.2.3.1 anonymous enum	14
6.2.3.2 anonymous enum	15
6.2.3.3 anonymous enum	15
6.2.3.4 anonymous enum	15
6.2.3.5 anonymous enum	16
6.2.4 Function Documentation	16
6.2.4.1 gpiod_ctxless_event_loop()	16
6.2.4.2 gpiod_ctxless_event_loop_multiple()	17
6.2.4.3 gpiod_ctxless_event_monitor()	18
6.2.4.4 gpiod_ctxless_event_monitor_ext()	18
6.2.4.5 gpiod_ctxless_event_monitor_multiple()	19
6.2.4.6 gpiod_ctxless_event_monitor_multiple_ext()	20
6.2.4.7 gpiod_ctxless_find_line()	21
6.2.4.8 gpiod_ctxless_get_value()	22
6.2.4.9 gpiod_ctxless_get_value_ext()	22
6.2.4.10 gpiod_ctxless_get_value_multiple()	23
6.2.4.11 gpiod_ctxless_get_value_multiple_ext()	23
6.2.4.12 gpiod_ctxless_set_value()	24
6.2.4.13 gpiod_ctxless_set_value_ext()	24
6.2.4.14 gpiod_ctxless_set_value_multiple()	25

6.2.4.15 <code>gpiod_ctxless_set_value_multiple_ext()</code>	26
6.3 GPIO chip operations	26
6.3.1 Detailed Description	27
6.3.2 Function Documentation	27
6.3.2.1 <code>gpiod_chip_close()</code>	27
6.3.2.2 <code>gpiod_chip_find_line()</code>	27
6.3.2.3 <code>gpiod_chip_find_lines()</code>	28
6.3.2.4 <code>gpiod_chip_get_all_lines()</code>	29
6.3.2.5 <code>gpiod_chip_get_line()</code>	29
6.3.2.6 <code>gpiod_chip_get_lines()</code>	29
6.3.2.7 <code>gpiod_chip_label()</code>	30
6.3.2.8 <code>gpiod_chip_name()</code>	30
6.3.2.9 <code>gpiod_chip_num_lines()</code>	31
6.3.2.10 <code>gpiod_chip_open()</code>	31
6.3.2.11 <code>gpiod_chip_open_by_label()</code>	31
6.3.2.12 <code>gpiod_chip_open_by_name()</code>	32
6.3.2.13 <code>gpiod_chip_open_by_number()</code>	32
6.3.2.14 <code>gpiod_chip_open_lookup()</code>	32
6.4 GPIO line operations	33
6.4.1 Detailed Description	34
6.5 Operating on multiple lines	34
6.5.1 Detailed Description	34
6.5.2 Macro Definition Documentation	34
6.5.2.1 <code>gpiod_line_bulk_foreach_line</code>	34
6.5.2.2 <code>gpiod_line_bulk_foreach_line_off</code>	35
6.5.2.3 <code>GPIOD_LINE_BULK_INITIALIZER</code>	35
6.6 Line info	35
6.6.1 Detailed Description	36
6.6.2 Enumeration Type Documentation	36
6.6.2.1 anonymous enum	36
6.6.2.2 anonymous enum	37
6.6.2.3 anonymous enum	37
6.6.3 Function Documentation	37
6.6.3.1 <code>gpiod_line_active_state()</code>	37
6.6.3.2 <code>gpiod_line_bias()</code>	38
6.6.3.3 <code>gpiod_line_consumer()</code>	38
6.6.3.4 <code>gpiod_line_direction()</code>	39
6.6.3.5 <code>gpiod_line_is_open_drain()</code>	39
6.6.3.6 <code>gpiod_line_is_open_source()</code>	39
6.6.3.7 <code>gpiod_line_is_used()</code>	40
6.6.3.8 <code>gpiod_line_name()</code>	40
6.6.3.9 <code>gpiod_line_needs_update()</code>	40

6.6.3.10 <code>gpiod_line_offset()</code>	41
6.6.3.11 <code>gpiod_line_update()</code>	41
6.7 Line requests	42
6.7.1 Detailed Description	44
6.7.2 Enumeration Type Documentation	44
6.7.2.1 anonymous enum	44
6.7.2.2 anonymous enum	44
6.7.3 Function Documentation	44
6.7.3.1 <code>gpiod_line_is_free()</code>	45
6.7.3.2 <code>gpiod_line_is_requested()</code>	45
6.7.3.3 <code>gpiod_line_release()</code>	45
6.7.3.4 <code>gpiod_line_release_bulk()</code>	46
6.7.3.5 <code>gpiod_line_request()</code>	46
6.7.3.6 <code>gpiod_line_request_both_edges_events()</code>	46
6.7.3.7 <code>gpiod_line_request_both_edges_events_flags()</code>	47
6.7.3.8 <code>gpiod_line_request_bulk()</code>	47
6.7.3.9 <code>gpiod_line_request_bulk_both_edges_events()</code>	48
6.7.3.10 <code>gpiod_line_request_bulk_both_edges_events_flags()</code>	48
6.7.3.11 <code>gpiod_line_request_bulk_falling_edge_events()</code>	48
6.7.3.12 <code>gpiod_line_request_bulk_falling_edge_events_flags()</code>	49
6.7.3.13 <code>gpiod_line_request_bulk_input()</code>	49
6.7.3.14 <code>gpiod_line_request_bulk_input_flags()</code>	50
6.7.3.15 <code>gpiod_line_request_bulk_output()</code>	50
6.7.3.16 <code>gpiod_line_request_bulk_output_flags()</code>	50
6.7.3.17 <code>gpiod_line_request_bulk_rising_edge_events()</code>	51
6.7.3.18 <code>gpiod_line_request_bulk_rising_edge_events_flags()</code>	51
6.7.3.19 <code>gpiod_line_request_falling_edge_events()</code>	52
6.7.3.20 <code>gpiod_line_request_falling_edge_events_flags()</code>	52
6.7.3.21 <code>gpiod_line_request_input()</code>	52
6.7.3.22 <code>gpiod_line_request_input_flags()</code>	53
6.7.3.23 <code>gpiod_line_request_output()</code>	53
6.7.3.24 <code>gpiod_line_request_output_flags()</code>	54
6.7.3.25 <code>gpiod_line_request_rising_edge_events()</code>	54
6.7.3.26 <code>gpiod_line_request_rising_edge_events_flags()</code>	55
6.8 Reading & setting line values	55
6.8.1 Detailed Description	55
6.8.2 Function Documentation	56
6.8.2.1 <code>gpiod_line_get_value()</code>	56
6.8.2.2 <code>gpiod_line_get_value_bulk()</code>	56
6.8.2.3 <code>gpiod_line_set_value()</code>	56
6.8.2.4 <code>gpiod_line_set_value_bulk()</code>	57
6.9 Setting line configuration	57

6.9.1 Detailed Description	58
6.9.2 Function Documentation	58
6.9.2.1 <code>gpiod_line_set_config()</code>	58
6.9.2.2 <code>gpiod_line_set_config_bulk()</code>	59
6.9.2.3 <code>gpiod_line_set_direction_input()</code>	59
6.9.2.4 <code>gpiod_line_set_direction_input_bulk()</code>	60
6.9.2.5 <code>gpiod_line_set_direction_output()</code>	60
6.9.2.6 <code>gpiod_line_set_direction_output_bulk()</code>	60
6.9.2.7 <code>gpiod_line_set_flags()</code>	61
6.9.2.8 <code>gpiod_line_set_flags_bulk()</code>	61
6.10 Line events handling	62
6.10.1 Detailed Description	62
6.10.2 Enumeration Type Documentation	63
6.10.2.1 anonymous enum	63
6.10.3 Function Documentation	63
6.10.3.1 <code>gpiod_line_event_get_fd()</code>	63
6.10.3.2 <code>gpiod_line_event_read()</code>	63
6.10.3.3 <code>gpiod_line_event_read_fd()</code>	64
6.10.3.4 <code>gpiod_line_event_read_fd_multiple()</code>	64
6.10.3.5 <code>gpiod_line_event_read_multiple()</code>	65
6.10.3.6 <code>gpiod_line_event_wait()</code>	65
6.10.3.7 <code>gpiod_line_event_wait_bulk()</code>	66
6.11 Misc line functions	66
6.11.1 Detailed Description	66
6.11.2 Function Documentation	67
6.11.2.1 <code>gpiod_line_close_chip()</code>	67
6.11.2.2 <code>gpiod_line_find()</code>	67
6.11.2.3 <code>gpiod_line_get()</code>	67
6.11.2.4 <code>gpiod_line_get_chip()</code>	68
6.12 Iterators for GPIO chips and lines	68
6.12.1 Detailed Description	69
6.12.2 Macro Definition Documentation	69
6.12.2.1 <code>gpiod_foreach_chip</code>	69
6.12.2.2 <code>gpiod_foreach_chip_noclose</code>	69
6.12.2.3 <code>gpiod_foreach_line</code>	70
6.12.3 Function Documentation	70
6.12.3.1 <code>gpiod_chip_iter_free()</code>	70
6.12.3.2 <code>gpiod_chip_iter_free_noclose()</code>	71
6.12.3.3 <code>gpiod_chip_iter_new()</code>	71
6.12.3.4 <code>gpiod_chip_iter_next()</code>	71
6.12.3.5 <code>gpiod_chip_iter_next_noclose()</code>	72
6.12.3.6 <code>gpiod_line_iter_free()</code>	72

6.12.3.7 <code>gpiod_line_iter_new()</code>	72
6.12.3.8 <code>gpiod_line_iter_next()</code>	73
6.13 Stuff that didn't fit anywhere else	73
6.13.1 Detailed Description	73
6.13.2 Function Documentation	73
6.13.2.1 <code>gpiod_version_string()</code>	73
7 Class Documentation	75
7.1 <code>gpiod_ctxless_event_poll_fd</code> Struct Reference	75
7.1.1 Detailed Description	75
7.1.2 Member Data Documentation	75
7.1.2.1 <code>event</code>	75
7.1.2.2 <code>fd</code>	75
7.2 <code>gpiod_line_bulk</code> Struct Reference	76
7.2.1 Detailed Description	76
7.2.2 Member Data Documentation	76
7.2.2.1 <code>lines</code>	76
7.2.2.2 <code>num_lines</code>	76
7.3 <code>gpiod_line_event</code> Struct Reference	76
7.3.1 Detailed Description	77
7.3.2 Member Data Documentation	77
7.3.2.1 <code>event_type</code>	77
7.3.2.2 <code>ts</code>	77
7.4 <code>gpiod_line_request_config</code> Struct Reference	77
7.4.1 Detailed Description	77
7.4.2 Member Data Documentation	78
7.4.2.1 <code>consumer</code>	78
7.4.2.2 <code>flags</code>	78
7.4.2.3 <code>request_type</code>	78
8 File Documentation	79
8.1 <code>gpiod.h</code> File Reference	79
8.2 <code>gpiod.h</code>	86
Index	93

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

Module Index

3.1 Modules

Here is a list of all modules:

Common helper macros	11
High-level API	12
GPIO chip operations	26
GPIO line operations	33
Operating on multiple lines	34
Line info	35
Line requests	42
Reading & setting line values	55
Setting line configuration	57
Line events handling	62
Misc line functions	66
Iterators for GPIO chips and lines	68
Stuff that didn't fit anywhere else	73

Chapter 4

Class Index

4.1 Class List

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

gpiod_ctxless_event_poll_fd	Helper structure for the ctxless event loop poll callback	75
gpiod_line_bulk	Helper structure for storing a set of GPIO line objects	76
gpiod_line_event	Structure holding event info	76
gpiod_line_request_config	Structure holding configuration of a line request	77

Chapter 5

File Index

5.1 File List

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

gpiod.h	79
-------------------------	----

Chapter 6

Module 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) (1UL << (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 ) (1UL << (nr))
```

Shift 1 by given offset.

Parameters

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

Returns

1 shifted by nr.

6.2 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.

6.2.1 Detailed Description

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

6.2.2 Typedef Documentation

6.2.2.1 `gpiod_ctxless_event_handle_cb`

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

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`.

6.2.2.2 `gpiod_ctxless_event_poll_cb`

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

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.

6.2.3 Enumeration Type Documentation

6.2.3.1 anonymous enum

```
anonymous enum
```

Miscellaneous GPIO flags.

Enumerator

GPIOCTXLESS_FLAG_OPEN_DRAIN	The line is an open-drain port.
GPIOCTXLESS_FLAG_OPEN_SOURCE	The line is an open-source port.
GPIOCTXLESS_FLAG_BIAS_DISABLE	The line has neither either pull-up nor pull-down resistor
GPIOCTXLESS_FLAG_BIAS_PULL_DOWN	The line has pull-down resistor enabled
GPIOCTXLESS_FLAG_BIAS_PULL_UP	The line has pull-up resistor enabled

6.2.3.2 anonymous enum

```
anonymous enum
```

Event types that the ctxless event monitor can wait for.

Enumerator

GPIOCTXLESS_EVENT_RISING_EDGE	Wait for rising edge events only. Wait for falling edge events only.
GPIOCTXLESS_EVENT_FALLING_EDGE	Wait for both types of events.

6.2.3.3 anonymous enum

```
anonymous enum
```

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

Enumerator

GPIOCTXLESS_EVENT_CB_TIMEOUT	Waiting for events timed out.
GPIOCTXLESS_EVENT_CB_RISING_EDGE	Rising edge event occurred.
GPIOCTXLESS_EVENT_CB_FALLING_EDGE	Falling edge event occurred.

6.2.3.4 anonymous enum

```
anonymous enum
```

Return status values that the ctxless event callback can return.

Enumerator

GPIOCTXLESS_EVENT_CB_RET_ERR	Stop processing events and indicate an error.
GPIOCTXLESS_EVENT_CB_RET_OK	Continue processing events.
GPIOCTXLESS_EVENT_CB_RET_STOP	Stop processing events.

6.2.3.5 anonymous enum

anonymous enum

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.

6.2.4 Function Documentation

6.2.4.1 gpiod_ctxless_event_loop()

```
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 )
```

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 [gpiod_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.

6.2.4.2 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 )
```

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.

6.2.4.3 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 )
```

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.

6.2.4.4 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 )

```

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.

6.2.4.5 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 )

```

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.

6.2.4.6 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 )
```

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.

Parameters

<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.

6.2.4.7 gpiod_ctxless_find_line()

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

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.

6.2.4.8 gpiod_ctxless_get_value()

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

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.

6.2.4.9 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 )
```

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.

6.2.4.10 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 )
```

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.

6.2.4.11 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 )
```

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.

6.2.4.12 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 )
```

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.

6.2.4.13 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 )
```

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.

6.2.4.14 `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 )
```

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 gpiod_ctxless_set_value .
<i>data</i>	Optional user data that will be passed to the callback function.

Returns

0 if the operation succeeds, -1 on error.

6.2.4.15 `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 )
```

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 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 gpiod_ctxless_set_value .
<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.

6.3 GPIO chip operations

Functions

- 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`) [GPIOD_API](#)
Get the GPIO chip label as represented in the kernel.
- `unsigned int gpiod_chip_num_lines` (`struct gpiod_chip *chip`) [GPIOD_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`) [GPIOD_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`) [GPIOD_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`) [GPIOD_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`) [GPIOD_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`) [GPIOD_API](#)
Find a set of GPIO lines by names among lines exposed by this chip.

6.3.1 Detailed Description

Functions and data structures dealing with GPIO chips.

6.3.2 Function Documentation

6.3.2.1 `gpiod_chip_close()`

```
void gpiod_chip_close (
    struct gpiod_chip * chip )
```

Close a GPIO chip handle and release all allocated resources.

Parameters

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

6.3.2.2 `gpiod_chip_find_line()`

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

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.

6.3.2.3 gpiod_chip_find_lines()

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

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.

6.3.2.4 `gpiod_chip_get_all_lines()`

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

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.

6.3.2.5 `gpiod_chip_get_line()`

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

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.

6.3.2.6 `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 )
```

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.

6.3.2.7 `gpiod_chip_label()`

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

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.

6.3.2.8 `gpiod_chip_name()`

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

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.

6.3.2.9 `gpiod_chip_num_lines()`

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

Get the number of GPIO lines exposed by this chip.

Parameters

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

Returns

Number of GPIO lines.

6.3.2.10 `gpiod_chip_open()`

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

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.

6.3.2.11 `gpiod_chip_open_by_label()`

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

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`.

6.3.2.12 `gpiod_chip_open_by_name()`

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

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.

6.3.2.13 `gpiod_chip_open_by_number()`

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

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.

6.3.2.14 `gpiod_chip_open_lookup()`

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

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

Parameters

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

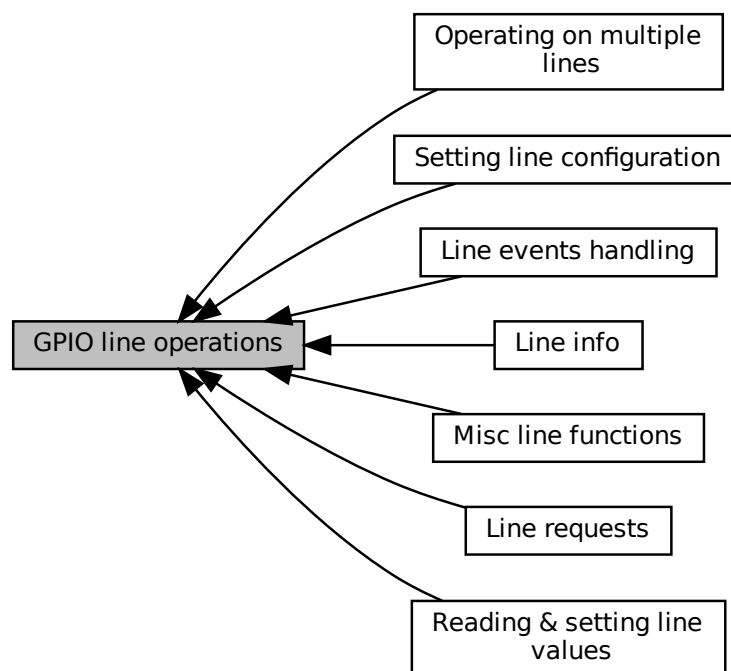
Returns

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.

6.4 GPIO line operations

Collaboration diagram for GPIO line operations:

**Modules**

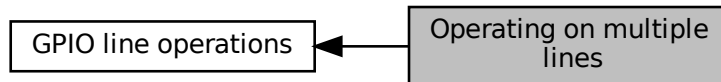
- [Operating on multiple lines](#)
- [Line info](#)
- [Line requests](#)
- [Reading & setting line values](#)
- [Setting line configuration](#)
- [Line events handling](#)
- [Misc line functions](#)

6.4.1 Detailed Description

Functions and data structures dealing with GPIO lines.

6.5 Operating on multiple lines

Collaboration diagram for Operating on multiple lines:



Classes

- struct [gpiod_line_bulk](#)
Helper structure for storing a set of GPIO line objects.

Macros

- #define **GPIO_LINE_BULK_MAX_LINES** 64
Maximum number of GPIO lines that can be requested at once.
- #define [GPIO_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).

6.5.1 Detailed Description

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

6.5.2 Macro Definition Documentation

6.5.2.1 gpiod_line_bulk_foreach_line

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

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.

6.5.2.2 `gpiod_line_bulk_foreach_line_off`

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

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.

6.5.2.3 `GPIOD_LINE_BULK_INITIALIZER`

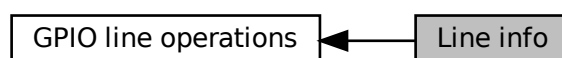
```
#define GPIOD_LINE_BULK_INITIALIZER { { NULL }, 0 }
```

Static initializer for GPIO bulk objects.

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

6.6 Line info

Collaboration diagram for Line info:



Enumerations

- enum { `GPIO_LINE_DIRECTION_INPUT` = 1 , `GPIO_LINE_DIRECTION_OUTPUT` }
Possible direction settings.
- enum { `GPIO_LINE_ACTIVE_STATE_HIGH` = 1 , `GPIO_LINE_ACTIVE_STATE_LOW` }
Possible active state settings.
- enum { `GPIO_LINE_BIAS_AS_IS` = 1 , `GPIO_LINE_BIAS_DISABLE` , `GPIO_LINE_BIAS_PULL_UP` , `GPIO_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.

6.6.1 Detailed Description

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

6.6.2 Enumeration Type Documentation

6.6.2.1 anonymous enum

anonymous enum

Possible direction settings.

Enumerator

GPIOD_LINE_DIRECTION_INPUT	Direction is input - we're reading the state of a GPIO line.
GPIOD_LINE_DIRECTION_OUTPUT	Direction is output - we're driving the GPIO line.

6.6.2.2 anonymous enum

```
anonymous enum
```

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.

6.6.2.3 anonymous enum

```
anonymous enum
```

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.

6.6.3 Function Documentation

6.6.3.1 gpiod_line_active_state()

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

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.

6.6.3.2 gpiod_line_bias()

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

Read the GPIO line bias setting.

Parameters

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

Returns

Returns GPIOD_LINE_BIAS_PULL_UP, GPIOD_LINE_BIAS_PULL_DOWN, GPIOD_LINE_BIAS_DISABLE or GPIOD_LINE_BIAS_AS_IS.

6.6.3.3 gpiod_line_consumer()

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

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.

6.6.3.4 `gpiod_line_direction()`

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

Read the GPIO line direction setting.

Parameters

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

Returns

Returns `GPIO_LINE_DIRECTION_INPUT` or `GPIO_LINE_DIRECTION_OUTPUT`.

6.6.3.5 `gpiod_line_is_open_drain()`

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

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.

6.6.3.6 `gpiod_line_is_open_source()`

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

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.

6.6.3.7 `gpiod_line_is_used()`

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

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.

6.6.3.8 `gpiod_line_name()`

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

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.

6.6.3.9 `gpiod_line_needs_update()`

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

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.

6.6.3.10 gpiod_line_offset()

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

Read the GPIO line offset.

Parameters

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

Returns

Line offset.

6.6.3.11 gpiod_line_update()

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

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.

6.7 Line requests

Collaboration diagram for 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.

6.7.1 Detailed Description

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

6.7.2 Enumeration Type Documentation

6.7.2.1 anonymous enum

`anonymous enum`

Available types of requests.

Enumerator

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

6.7.2.2 anonymous enum

`anonymous enum`

Miscellaneous GPIO request flags.

Enumerator

<code>GPIOD_LINE_REQUEST_FLAG_OPEN_DRAIN</code>	The line is an open-drain port.
<code>GPIOD_LINE_REQUEST_FLAG_OPEN_SOURCE</code>	The line is an open-source port.
<code>GPIOD_LINE_REQUEST_FLAG_ACTIVE_LOW</code>	The active state of the line is low (high is the default).
<code>GPIOD_LINE_REQUEST_FLAG_BIAS_DISABLE</code>	The line has neither either pull-up nor pull-down resistor.
<code>GPIOD_LINE_REQUEST_FLAG_BIAS_PULL_DOWN</code>	The line has pull-down resistor enabled.
<code>GPIOD_LINE_REQUEST_FLAG_BIAS_PULL_UP</code>	The line has pull-up resistor enabled.

6.7.3 Function Documentation

6.7.3.1 `gpiod_line_is_free()`

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

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.

6.7.3.2 `gpiod_line_is_requested()`

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

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.

6.7.3.3 `gpiod_line_release()`

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

Release a previously reserved line.

Parameters

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

6.7.3.4 `gpiod_line_release_bulk()`

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

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.

6.7.3.5 `gpiod_line_request()`

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

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.

6.7.3.6 `gpiod_line_request_both_edges_events()`

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

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.

6.7.3.7 `gpiod_line_request_both_edges_events_flags()`

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

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.

6.7.3.8 `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 )
```

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.

6.7.3.9 `gpiod_line_request_bulk_both_edges_events()`

```
int gpiod_line_request_bulk_both_edges_events (
    struct gpiod\_line\_bulk * bulk,
    const char * consumer )
```

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.

6.7.3.10 `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 )
```

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.

6.7.3.11 `gpiod_line_request_bulk_falling_edge_events()`

```
int gpiod_line_request_bulk_falling_edge_events (
    struct gpiod\_line\_bulk * bulk,
    const char * consumer )
```

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.

6.7.3.12 `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 )
```

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.

6.7.3.13 `gpiod_line_request_bulk_input()`

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

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.

6.7.3.14 `gpiod_line_request_bulk_input_flags()`

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

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.

6.7.3.15 `gpiod_line_request_bulk_output()`

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

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.

6.7.3.16 `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 )

```

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.

6.7.3.17 `gpiod_line_request_bulk_rising_edge_events()`

```

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

```

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.

6.7.3.18 `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 )

```

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.

6.7.3.19 `gpiod_line_request_falling_edge_events()`

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

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.

6.7.3.20 `gpiod_line_request_falling_edge_events_flags()`

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

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.

6.7.3.21 `gpiod_line_request_input()`

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

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.

6.7.3.22 `gpiod_line_request_input_flags()`

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

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.

6.7.3.23 `gpiod_line_request_output()`

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

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.

6.7.3.24 `gpiod_line_request_output_flags()`

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

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.

6.7.3.25 `gpiod_line_request_rising_edge_events()`

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

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.

6.7.3.26 `gpiod_line_request_rising_edge_events_flags()`

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

Request rising edge event notifications on a single line.

Parameters

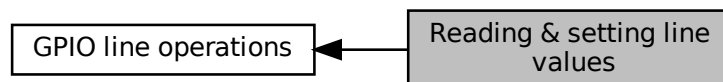
<i>line</i>	GPIO line object.
<i>consumer</i>	Name of the consumer.
<i>flags</i>	Additional request flags.

Returns

0 if the operation succeeds, -1 on failure.

6.8 Reading & setting line values

Collaboration diagram for 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.

6.8.1 Detailed Description

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

6.8.2 Function Documentation

6.8.2.1 `gpiod_line_get_value()`

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

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.

6.8.2.2 `gpiod_line_get_value_bulk()`

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

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->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.

6.8.2.3 `gpiod_line_set_value()`

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

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.

6.8.2.4 `gpiod_line_set_value_bulk()`

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

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->num_lines</code> new values for lines. A NULL pointer is interpreted as a logical low for all lines.

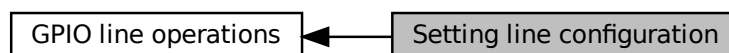
Returns

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.

6.9 Setting line configuration

Collaboration diagram for Setting line configuration:



Functions

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

6.9.1 Detailed Description

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

6.9.2 Function Documentation

6.9.2.1 [gpiod_line_set_config\(\)](#)

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

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>GPIOD_LINE_REQUEST_DIRECTION_AS_IS</code> , <code>GPIOD_LINE_REQUEST_DIRECTION_INPUT</code> , or <code>GPIOD_LINE_REQUEST_DIRECTION_OUTPUT</code> .
<i>flags</i>	Replacement flags.
<i>value</i>	The new output value for the line when direction is <code>GPIOD_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.

6.9.2.2 gpiod_line_set_config_bulk()

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

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 GPIOD_LINE_REQUEST_DIRECTION_AS_IS, GPIOD_LINE_REQUEST_DIRECTION_INPUT, or GPIOD_LINE_REQUEST_DIRECTION_OUTPUT.
<i>flags</i>	Replacement flags.
<i>values</i>	An array holding line_bulk->num_lines new logical values for lines when direction is GPIOD_LINE_REQUEST_DIRECTION_OUTPUT. 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.

6.9.2.3 gpiod_line_set_direction_input()

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

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.

6.9.2.4 `gpiod_line_set_direction_input_bulk()`

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

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.

6.9.2.5 `gpiod_line_set_direction_output()`

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

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.

6.9.2.6 `gpiod_line_set_direction_output_bulk()`

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

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->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.

6.9.2.7 gpiod_line_set_flags()

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

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.

6.9.2.8 gpiod_line_set_flags_bulk()

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

Update the configuration flags of a set of GPIO lines.

Parameters

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

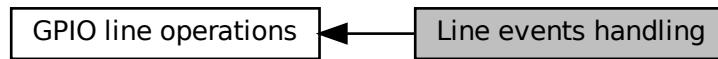
Returns

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.

6.10 Line events handling

Collaboration diagram for 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.

6.10.1 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.

6.10.2 Enumeration Type Documentation

6.10.2.1 anonymous enum

anonymous enum

Event types.

Enumerator

GPIOD_LINE_EVENT_RISING_EDGE	Rising edge event.
GPIOD_LINE_EVENT_FALLING_EDGE	Falling edge event.

6.10.3 Function Documentation

6.10.3.1 `gpiod_line_event_get_fd()`

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

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.

6.10.3.2 `gpiod_line_event_read()`

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

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.

6.10.3.3 `gpiod_line_event_read_fd()`

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

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.

6.10.3.4 `gpiod_line_event_read_fd_multiple()`

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

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 num_events.
<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.

6.10.3.5 gpiod_line_event_read_multiple()

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

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 num_events.
<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.

6.10.3.6 gpiod_line_event_wait()

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

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.

6.10.3.7 `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 )
```

Wait for events on a set of lines.

Parameters

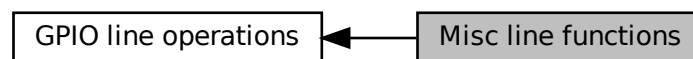
<i>bulk</i>	Set of GPIO lines to monitor.
<i>timeout</i>	Wait time limit.
<i>event_bulk</i>	Bulk object in which to store the line handles on which events occurred. Can be NULL.

Returns

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

6.11 Misc line functions

Collaboration diagram for 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.

6.11.1 Detailed Description

Functions that didn't fit anywhere else.

6.11.2 Function Documentation

6.11.2.1 `gpiod_line_close_chip()`

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

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.

6.11.2.2 `gpiod_line_find()`

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

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`.

6.11.2.3 `gpiod_line_get()`

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

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.

6.11.2.4 gpiod_line_get_chip()

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

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.

6.12 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) [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.

6.12.1 Detailed Description

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

6.12.2 Macro Definition Documentation

6.12.2.1 gpiod_foreach_chip

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

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](#).

6.12.2.2 gpiod_foreach_chip_noclose

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

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.

6.12.2.3 gpiod_foreach_line

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

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.

6.12.3 Function Documentation

6.12.3.1 gpiod_chip_iter_free()

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

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.
-------------	-------------------------------

6.12.3.2 `gpiod_chip_iter_free_noclose()`

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

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.

6.12.3.3 `gpiod_chip_iter_new()`

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

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.

6.12.3.4 `gpiod_chip_iter_next()`

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

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](#).

6.12.3.5 `gpiod_chip_iter_next_noclose()`

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

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.

6.12.3.6 `gpiod_line_iter_free()`

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

Free all resources associated with a GPIO line iterator.

Parameters

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

6.12.3.7 `gpiod_line_iter_new()`

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

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.

6.12.3.8 `gpiod_line_iter_next()`

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

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.

6.13 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.

6.13.1 Detailed Description

Various libgpiod-related functions.

6.13.2 Function Documentation

6.13.2.1 `gpiod_version_string()`

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

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

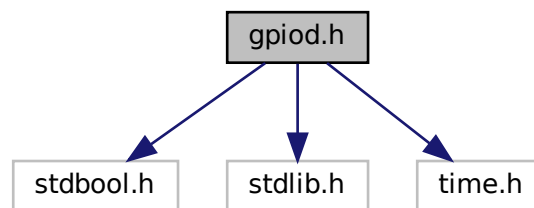
8.1 gpiod.h File Reference

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

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

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

Include dependency graph for gpiod.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) (1UL << (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_DISABLE` = `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) [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.

- 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.
- 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.
- 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) `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) `GPIOD_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) `GPIOD_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) `GPIOD_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) `GPIOD_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) `GPIOD_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) `GPIOD_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) `GPIOD_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) `GPIOD_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) `GPIOD_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) `GPIOD_API`
Request all event type notifications on a set of lines.
- `void gpiod_line_release` (struct `gpiod_line` *line) `GPIOD_API`
Release a previously reserved line.
- `void gpiod_line_release_bulk` (struct `gpiod_line_bulk` *bulk) `GPIOD_API`
Release a set of previously reserved lines.
- `bool gpiod_line_is_requested` (struct `gpiod_line` *line) `GPIOD_API`
Check if the calling user has ownership of this line.
- `bool gpiod_line_is_free` (struct `gpiod_line` *line) `GPIOD_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) `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) [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 gpiodchip iterator but don't close the most recently opened gpiodchip (if any).
- struct [gpiod_chip](#) * [gpiod_chip_iter_next](#) (struct [gpiod_chip_iter](#) *iter) [GPIO_API](#)
Get the next gpiodchip handle.
- struct [gpiod_chip](#) * [gpiod_chip_iter_next_noclose](#) (struct [gpiod_chip_iter](#) *iter) [GPIO_API](#)
Get the next gpiodchip 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.](#)

```

1 /* SPDX-License-Identifier: LGPL-2.1-or-later */
2 /*
3  * This file is part of libgpiod.
4  *
5  * Copyright (C) 2017-2018 Bartosz Golaszewski <bartekgola@gmail.com>
6  */
7
8 #ifndef __LIBGPIOD_GPIO_H__
9 #define __LIBGPIOD_GPIO_H__
10
11 #include <stdbool.h>
12 #include <stdlib.h>
13 #include <time.h>
14
15 #ifdef __cplusplus
16 extern "C" {
17 #endif
18
19 struct gpiod_chip;
20 struct gpiod_line;
21 struct gpiod_chip_iter;
22 struct gpiod_line_iter;
23 struct gpiod_line_bulk;
24
25 #define GPIO_API __attribute__((visibility("default")))
26
27 #define GPIO_UNUSED __attribute__((unused))
28
29 #define GPIO_BIT(nr) (1UL << (nr))
30
31 #define GPIO_DEPRECATED __attribute__((deprecated))
32
33 enum {
34     GPIO_CTXLESS_FLAG_OPEN_DRAIN = GPIO_BIT(0),
35     GPIO_CTXLESS_FLAG_OPEN_SOURCE = GPIO_BIT(1),
36     GPIO_CTXLESS_FLAG_BIAS_DISABLE = GPIO_BIT(2),
37     GPIO_CTXLESS_FLAG_BIAS_PULL_DOWN = GPIO_BIT(3),
38     GPIO_CTXLESS_FLAG_BIAS_PULL_UP = GPIO_BIT(4),
39 };
40
41 int gpiod_ctxless_get_value(const char *device, unsigned int offset,
42                             bool active_low, const char *consumer) GPIO_API;
43
44 int gpiod_ctxless_get_value_ext(const char *device, unsigned int offset,
45                                 bool active_low, const char *consumer,
46                                 int flags) GPIO_API;
47
48 int gpiod_ctxless_get_value_multiple(const char *device,
49                                     const unsigned int *offsets, int *values,
50                                     unsigned int num_lines, bool active_low,
51                                     const char *consumer) GPIO_API;
52
53 int gpiod_ctxless_get_value_multiple_ext(const char *device,
54                                     const unsigned int *offsets,

```

```

159         int *values, unsigned int num_lines,
160         bool active_low, const char *consumer,
161         int flags) GPIOD_API;
162
163 typedef void (*gpiod_ctxless_set_value_cb)(void *);
164
165 int gpiod_ctxless_set_value(const char *device, unsigned int offset, int value,
166     bool active_low, const char *consumer,
167     gpiod_ctxless_set_value_cb cb,
168     void *data) GPIOD_API;
169
170 int gpiod_ctxless_set_value_ext(const char *device, unsigned int offset,
171     int value, bool active_low,
172     const char *consumer,
173     gpiod_ctxless_set_value_cb cb,
174     void *data, int flags) GPIOD_API;
175
176 int gpiod_ctxless_set_value_multiple(const char *device,
177     const unsigned int *offsets,
178     const int *values, unsigned int num_lines,
179     bool active_low, const char *consumer,
180     gpiod_ctxless_set_value_cb cb,
181     void *data) GPIOD_API;
182
183 int gpiod_ctxless_set_value_multiple_ext(const char *device,
184     const unsigned int *offsets,
185     const int *values,
186     unsigned int num_lines,
187     bool active_low,
188     const char *consumer,
189     gpiod_ctxless_set_value_cb cb,
190     void *data, int flags) GPIOD_API;
191
192 enum {
193     GPIOD_CTXLESS_EVENT_RISING_EDGE = 1,
194     GPIOD_CTXLESS_EVENT_FALLING_EDGE,
195     GPIOD_CTXLESS_EVENT_BOTH_EDGES,
196 };
197
198 enum {
199     GPIOD_CTXLESS_EVENT_CB_TIMEOUT = 1,
200     GPIOD_CTXLESS_EVENT_CB_RISING_EDGE,
201     GPIOD_CTXLESS_EVENT_CB_FALLING_EDGE,
202 };
203
204 enum {
205     GPIOD_CTXLESS_EVENT_CB_RET_ERR = -1,
206     GPIOD_CTXLESS_EVENT_CB_RET_OK = 0,
207     GPIOD_CTXLESS_EVENT_CB_RET_STOP = 1,
208 };
209
210 typedef int (*gpiod_ctxless_event_handle_cb)(int, unsigned int,
211     const struct timespec *, void *);
212
213 enum {
214     GPIOD_CTXLESS_EVENT_POLL_RET_STOP = -2,
215     GPIOD_CTXLESS_EVENT_POLL_RET_ERR = -1,
216     GPIOD_CTXLESS_EVENT_POLL_RET_TIMEOUT = 0,
217 };
218
219 struct gpiod_ctxless_event_poll_fd {
220     int fd;
221     bool event;
222 };
223
224 typedef int (*gpiod_ctxless_event_poll_cb)(unsigned int,
225     struct gpiod_ctxless_event_poll_fd *,
226     const struct timespec *, void *);
227
228 int gpiod_ctxless_event_loop(const char *device, unsigned int offset,
229     bool active_low, const char *consumer,
230     const struct timespec *timeout,
231     gpiod_ctxless_event_poll_cb poll_cb,
232     gpiod_ctxless_event_handle_cb event_cb,
233     void *data) GPIOD_API GPIOD_DEPRECATED;
234
235 int gpiod_ctxless_event_loop_multiple(const char *device,
236     const unsigned int *offsets,
237     unsigned int num_lines, bool active_low,
238     const char *consumer,
239     const struct timespec *timeout,
240     gpiod_ctxless_event_poll_cb poll_cb,
241     gpiod_ctxless_event_handle_cb event_cb,
242     void *data) GPIOD_API GPIOD_DEPRECATED;
243
244 int gpiod_ctxless_event_monitor(const char *device, int event_type,
245     unsigned int offset, bool active_low,

```



```

423         const char *consumer,
424         const struct timespec *timeout,
425         gpiod_ctxless_event_poll_cb poll_cb,
426         gpiod_ctxless_event_handle_cb event_cb,
427         void *data) GPIO_API;
428
446 int gpiod_ctxless_event_monitor_ext(const char *device, int event_type,
447                                     unsigned int offset, bool active_low,
448                                     const char *consumer,
449                                     const struct timespec *timeout,
450                                     gpiod_ctxless_event_poll_cb poll_cb,
451                                     gpiod_ctxless_event_handle_cb event_cb,
452                                     void *data, int flags) GPIO_API;
453
484 int gpiod_ctxless_event_monitor_multiple(
485     const char *device, int event_type,
486     const unsigned int *offsets,
487     unsigned int num_lines, bool active_low,
488     const char *consumer, const struct timespec *timeout,
489     gpiod_ctxless_event_poll_cb poll_cb,
490     gpiod_ctxless_event_handle_cb event_cb,
491     void *data) GPIO_API;
492
524 int gpiod_ctxless_event_monitor_multiple_ext(
525     const char *device, int event_type,
526     const unsigned int *offsets,
527     unsigned int num_lines, bool active_low,
528     const char *consumer, const struct timespec *timeout,
529     gpiod_ctxless_event_poll_cb poll_cb,
530     gpiod_ctxless_event_handle_cb event_cb,
531     void *data, int flags) GPIO_API;
532
533
548 int gpiod_ctxless_find_line(const char *name, char *chipname,
549                             size_t chipname_size,
550                             unsigned int *offset) GPIO_API;
551
566 struct gpiod_chip *gpiod_chip_open(const char *path) GPIO_API;
567
575 struct gpiod_chip *gpiod_chip_open_by_name(const char *name) GPIO_API;
576
584 struct gpiod_chip *gpiod_chip_open_by_number(unsigned int num) GPIO_API;
585
594 struct gpiod_chip *gpiod_chip_open_by_label(const char *label) GPIO_API;
595
605 struct gpiod_chip *gpiod_chip_open_lookup(const char *descr) GPIO_API;
606
611 void gpiod_chip_close(struct gpiod_chip *chip) GPIO_API;
612
618 const char *gpiod_chip_name(struct gpiod_chip *chip) GPIO_API;
619
625 const char *gpiod_chip_label(struct gpiod_chip *chip) GPIO_API;
626
632 unsigned int gpiod_chip_num_lines(struct gpiod_chip *chip) GPIO_API;
633
640 struct gpiod_line *
641 gpiod_chip_get_line(struct gpiod_chip *chip, unsigned int offset) GPIO_API;
642
651 int gpiod_chip_get_lines(struct gpiod_chip *chip,
652                          unsigned int *offsets, unsigned int num_offsets,
653                          struct gpiod_line_bulk *bulk) GPIO_API;
654
661 int gpiod_chip_get_all_lines(struct gpiod_chip *chip,
662                              struct gpiod_line_bulk *bulk) GPIO_API;
663
676 struct gpiod_line *
677 gpiod_chip_find_line(struct gpiod_chip *chip, const char *name) GPIO_API;
678
692 int gpiod_chip_find_lines(struct gpiod_chip *chip, const char **names,
693                          struct gpiod_line_bulk *bulk) GPIO_API;
694
713 #define GPIOD_LINE_BULK_MAX_LINES    64
714
722 struct gpiod_line_bulk {
723     struct gpiod_line *lines[GPIOD_LINE_BULK_MAX_LINES];
725     unsigned int num_lines;
727 };
728
734 #define GPIOD_LINE_BULK_INITIALIZER { { NULL }, 0 }
735
742 static inline void gpiod_line_bulk_init(struct gpiod_line_bulk *bulk)
743 {
744     bulk->num_lines = 0;
745 }
746
752 static inline void gpiod_line_bulk_add(struct gpiod_line_bulk *bulk,
753                                         struct gpiod_line *line)

```



```

754 {
755     bulk->lines[bulk->num_lines++] = line;
756 }
757
758 static inline struct gpiod_line *
759 gpiod_line_bulk_get_line(struct gpiod_line_bulk *bulk, unsigned int offset)
760 {
761     return bulk->lines[offset];
762 }
763
764 static inline unsigned int
765 gpiod_line_bulk_num_lines(struct gpiod_line_bulk *bulk)
766 {
767     return bulk->num_lines;
768 }
769
770 #define gpiod_line_bulk_foreach_line(bulk, line, lineptr) \
771     for ((lineptr) = (bulk)->lines, (line) = *(lineptr); \
772          (lineptr) <= (bulk)->lines + ((bulk)->num_lines - 1); \
773          (lineptr)++, (line) = *(lineptr))
774
775 #define gpiod_line_bulk_foreach_line_off(bulk, line, offset) \
776     for ((offset) = 0, (line) = (bulk)->lines[0]; \
777          (offset) < (bulk)->num_lines; \
778          (offset)++, (line) = (bulk)->lines[(offset)])
779
780 enum {
781     GPIOD_LINE_DIRECTION_INPUT = 1,
782     GPIOD_LINE_DIRECTION_OUTPUT,
783 };
784
785 enum {
786     GPIOD_LINE_ACTIVE_STATE_HIGH = 1,
787     GPIOD_LINE_ACTIVE_STATE_LOW,
788 };
789
790 enum {
791     GPIOD_LINE_BIAS_AS_IS = 1,
792     GPIOD_LINE_BIAS_DISABLE,
793     GPIOD_LINE_BIAS_PULL_UP,
794     GPIOD_LINE_BIAS_PULL_DOWN,
795 };
796
797 unsigned int gpiod_line_offset(struct gpiod_line *line) GPIOD_API;
798
799 const char *gpiod_line_name(struct gpiod_line *line) GPIOD_API;
800
801 const char *gpiod_line_consumer(struct gpiod_line *line) GPIOD_API;
802
803 int gpiod_line_direction(struct gpiod_line *line) GPIOD_API;
804
805 int gpiod_line_active_state(struct gpiod_line *line) GPIOD_API;
806
807 int gpiod_line_bias(struct gpiod_line *line) GPIOD_API;
808
809 bool gpiod_line_is_used(struct gpiod_line *line) GPIOD_API;
810
811 bool gpiod_line_is_open_drain(struct gpiod_line *line) GPIOD_API;
812
813 bool gpiod_line_is_open_source(struct gpiod_line *line) GPIOD_API;
814
815 int gpiod_line_update(struct gpiod_line *line) GPIOD_API;
816
817 bool
818 gpiod_line_needs_update(struct gpiod_line *line) GPIOD_API GPIOD_DEPRECATED;
819
820 enum {
821     GPIOD_LINE_REQUEST_DIRECTION_AS_IS = 1,
822     GPIOD_LINE_REQUEST_DIRECTION_INPUT,
823     GPIOD_LINE_REQUEST_DIRECTION_OUTPUT,
824     GPIOD_LINE_REQUEST_EVENT_FALLING_EDGE,
825     GPIOD_LINE_REQUEST_EVENT_RISING_EDGE,
826     GPIOD_LINE_REQUEST_EVENT_BOTH_EDGES,
827 };
828
829 enum {
830     GPIOD_LINE_REQUEST_FLAG_OPEN_DRAIN = GPIOD_BIT(0),
831     GPIOD_LINE_REQUEST_FLAG_OPEN_SOURCE = GPIOD_BIT(1),
832     GPIOD_LINE_REQUEST_FLAG_ACTIVE_LOW = GPIOD_BIT(2),
833     GPIOD_LINE_REQUEST_FLAG_BIAS_DISABLE = GPIOD_BIT(3),
834     GPIOD_LINE_REQUEST_FLAG_BIAS_PULL_DOWN = GPIOD_BIT(4),
835     GPIOD_LINE_REQUEST_FLAG_BIAS_PULL_UP = GPIOD_BIT(5),
836 };
837
838 struct gpiod_line_request_config {
839     const char *consumer;
840     int request_type;
841 };

```

```
1013     int flags;
1015 };
1016
1029 int gpiod_line_request(struct gpiod_line *line,
1030     const struct gpiod_line_request_config *config,
1031     int default_val) GPIOD_API;
1032
1039 int gpiod_line_request_input(struct gpiod_line *line,
1040     const char *consumer) GPIOD_API;
1041
1049 int gpiod_line_request_output(struct gpiod_line *line,
1050     const char *consumer, int default_val) GPIOD_API;
1051
1058 int gpiod_line_request_rising_edge_events(struct gpiod_line *line,
1059     const char *consumer) GPIOD_API;
1060
1067 int gpiod_line_request_falling_edge_events(struct gpiod_line *line,
1068     const char *consumer) GPIOD_API;
1069
1076 int gpiod_line_request_both_edges_events(struct gpiod_line *line,
1077     const char *consumer) GPIOD_API;
1078
1086 int gpiod_line_request_input_flags(struct gpiod_line *line,
1087     const char *consumer, int flags) GPIOD_API;
1088
1097 int gpiod_line_request_output_flags(struct gpiod_line *line,
1098     const char *consumer, int flags,
1099     int default_val) GPIOD_API;
1100
1108 int gpiod_line_request_rising_edge_events_flags(struct gpiod_line *line,
1109     const char *consumer,
1110     int flags) GPIOD_API;
1111
1119 int gpiod_line_request_falling_edge_events_flags(struct gpiod_line *line,
1120     const char *consumer,
1121     int flags) GPIOD_API;
1122
1130 int gpiod_line_request_both_edges_events_flags(struct gpiod_line *line,
1131     const char *consumer,
1132     int flags) GPIOD_API;
1133
1147 int gpiod_line_request_bulk(struct gpiod_line_bulk *bulk,
1148     const struct gpiod_line_request_config *config,
1149     const int *default_vals) GPIOD_API;
1150
1157 int gpiod_line_request_bulk_input(struct gpiod_line_bulk *bulk,
1158     const char *consumer) GPIOD_API;
1159
1167 int gpiod_line_request_bulk_output(struct gpiod_line_bulk *bulk,
1168     const char *consumer,
1169     const int *default_vals) GPIOD_API;
1170
1177 int gpiod_line_request_bulk_rising_edge_events(struct gpiod_line_bulk *bulk,
1178     const char *consumer) GPIOD_API;
1179
1186 int gpiod_line_request_bulk_falling_edge_events(struct gpiod_line_bulk *bulk,
1187     const char *consumer) GPIOD_API;
1188
1195 int gpiod_line_request_bulk_both_edges_events(struct gpiod_line_bulk *bulk,
1196     const char *consumer) GPIOD_API;
1197
1205 int gpiod_line_request_bulk_input_flags(struct gpiod_line_bulk *bulk,
1206     const char *consumer,
1207     int flags) GPIOD_API;
1208
1217 int gpiod_line_request_bulk_output_flags(struct gpiod_line_bulk *bulk,
1218     const char *consumer, int flags,
1219     const int *default_vals) GPIOD_API;
1220
1228 int gpiod_line_request_bulk_rising_edge_events_flags(
1229     struct gpiod_line_bulk *bulk,
1230     const char *consumer,
1231     int flags) GPIOD_API;
1232
1240 int gpiod_line_request_bulk_falling_edge_events_flags(
1241     struct gpiod_line_bulk *bulk,
1242     const char *consumer,
1243     int flags) GPIOD_API;
1244
1252 int gpiod_line_request_bulk_both_edges_events_flags(
1253     struct gpiod_line_bulk *bulk,
1254     const char *consumer,
1255     int flags) GPIOD_API;
1256
1261 void gpiod_line_release(struct gpiod_line *line) GPIOD_API;
1262
1270 void gpiod_line_release_bulk(struct gpiod_line_bulk *bulk) GPIOD_API;
```

```
1271
1277 bool gpiod_line_is_requested(struct gpiod_line *line) GPIOD_API;
1278
1285 bool gpiod_line_is_free(struct gpiod_line *line) GPIOD_API;
1286
1303 int gpiod_line_get_value(struct gpiod_line *line) GPIOD_API;
1304
1316 int gpiod_line_get_value_bulk(struct gpiod_line_bulk *bulk,
1317                               int *values) GPIOD_API;
1318
1326 int gpiod_line_set_value(struct gpiod_line *line, int value) GPIOD_API;
1327
1339 int gpiod_line_set_value_bulk(struct gpiod_line_bulk *bulk,
1340                               const int *values) GPIOD_API;
1341
1365 int gpiod_line_set_config(struct gpiod_line *line, int direction,
1366                           int flags, int value) GPIOD_API;
1367
1386 int gpiod_line_set_config_bulk(struct gpiod_line_bulk *bulk,
1387                                int direction, int flags,
1388                                const int *values) GPIOD_API;
1389
1390
1398 int gpiod_line_set_flags(struct gpiod_line *line, int flags) GPIOD_API;
1399
1410 int gpiod_line_set_flags_bulk(struct gpiod_line_bulk *bulk,
1411                               int flags) GPIOD_API;
1412
1419 int gpiod_line_set_direction_input(struct gpiod_line *line) GPIOD_API;
1420
1430 int
1431 gpiod_line_set_direction_input_bulk(struct gpiod_line_bulk *bulk) GPIOD_API;
1432
1440 int gpiod_line_set_direction_output(struct gpiod_line *line,
1441                                     int value) GPIOD_API;
1442
1455 int gpiod_line_set_direction_output_bulk(struct gpiod_line_bulk *bulk,
1456                                           const int *values) GPIOD_API;
1457
1473 enum {
1474     GPIOD_LINE_EVENT_RISING_EDGE = 1,
1476     GPIOD_LINE_EVENT_FALLING_EDGE,
1478 };
1479
1483 struct gpiod_line_event {
1484     struct timespec ts;
1486     int event_type;
1488 };
1489
1497 int gpiod_line_event_wait(struct gpiod_line *line,
1498                           const struct timespec *timeout) GPIOD_API;
1499
1509 int gpiod_line_event_wait_bulk(struct gpiod_line_bulk *bulk,
1510                                const struct timespec *timeout,
1511                                struct gpiod_line_bulk *event_bulk) GPIOD_API;
1512
1520 int gpiod_line_event_read(struct gpiod_line *line,
1521                           struct gpiod_line_event *event) GPIOD_API;
1522
1532 int gpiod_line_event_read_multiple(struct gpiod_line *line,
1533                                    struct gpiod_line_event *events,
1534                                    unsigned int num_events) GPIOD_API;
1535
1546 int gpiod_line_event_get_fd(struct gpiod_line *line) GPIOD_API;
1547
1558 int gpiod_line_event_read_fd(int fd, struct gpiod_line_event *event) GPIOD_API;
1559
1569 int gpiod_line_event_read_fd_multiple(int fd, struct gpiod_line_event *events,
1570                                       unsigned int num_events) GPIOD_API;
1571
1593 struct gpiod_line *
1594 gpiod_line_get(const char *device, unsigned int offset) GPIOD_API;
1595
1609 struct gpiod_line *gpiod_line_find(const char *name) GPIOD_API;
1610
1617 void gpiod_line_close_chip(struct gpiod_line *line) GPIOD_API;
1618
1624 struct gpiod_chip *gpiod_line_get_chip(struct gpiod_line *line) GPIOD_API;
1625
1646 struct gpiod_chip_iter *gpiod_chip_iter_new(void) GPIOD_API;
1647
1653 void gpiod_chip_iter_free(struct gpiod_chip_iter *iter) GPIOD_API;
1654
1664 void gpiod_chip_iter_free_noclose(struct gpiod_chip_iter *iter) GPIOD_API;
1665
1673 struct gpiod_chip *
1674 gpiod_chip_iter_next(struct gpiod_chip_iter *iter) GPIOD_API;
```

```
1675
1684 struct gpiod_chip *
1685 gpiod_chip_iter_next_noclose(struct gpiod_chip_iter *iter) GPIOD_API;
1686
1697 #define gpiod_foreach_chip(iter, chip) \
1698 for ((chip) = gpiod_chip_iter_next(iter); \
1699 (chip); \
1700 (chip) = gpiod_chip_iter_next(iter))
1701
1712 #define gpiod_foreach_chip_noclose(iter, chip) \
1713 for ((chip) = gpiod_chip_iter_next_noclose(iter); \
1714 (chip); \
1715 (chip) = gpiod_chip_iter_next_noclose(iter))
1716
1723 struct gpiod_line_iter *
1724 gpiod_line_iter_new(struct gpiod_chip *chip) GPIOD_API;
1725
1730 void gpiod_line_iter_free(struct gpiod_line_iter *iter) GPIOD_API;
1731
1738 struct gpiod_line *
1739 gpiod_line_iter_next(struct gpiod_line_iter *iter) GPIOD_API;
1740
1747 #define gpiod_foreach_line(iter, line) \
1748 for ((line) = gpiod_line_iter_next(iter); \
1749 (line); \
1750 (line) = gpiod_line_iter_next(iter))
1751
1765 const char *gpiod_version_string(void) GPIOD_API;
1766
1771 #ifndef __cplusplus
1772 } /* extern "C" */
1773 #endif
1774
1775 #endif /* __LIBGPIOD_GPIOD_H__ */
```

Index

- Common helper macros, [11](#)
 - GPIOD_BIT, [11](#)
- consumer
 - gpiod_line_request_config, [78](#)
- event
 - gpiod_ctxless_event_poll_fd, [75](#)
- event_type
 - gpiod_line_event, [77](#)
- fd
 - gpiod_ctxless_event_poll_fd, [75](#)
- flags
 - gpiod_line_request_config, [78](#)
- GPIO chip operations, [26](#)
 - gpiod_chip_close, [27](#)
 - gpiod_chip_find_line, [27](#)
 - gpiod_chip_find_lines, [28](#)
 - gpiod_chip_get_all_lines, [28](#)
 - gpiod_chip_get_line, [29](#)
 - gpiod_chip_get_lines, [29](#)
 - gpiod_chip_label, [30](#)
 - gpiod_chip_name, [30](#)
 - gpiod_chip_num_lines, [30](#)
 - gpiod_chip_open, [31](#)
 - gpiod_chip_open_by_label, [31](#)
 - gpiod_chip_open_by_name, [32](#)
 - gpiod_chip_open_by_number, [32](#)
 - gpiod_chip_open_lookup, [32](#)
- GPIO line operations, [33](#)
- gpiod.h, [79](#)
- GPIOD_BIT
 - Common helper macros, [11](#)
- gpiod_chip_close
 - GPIO chip operations, [27](#)
- gpiod_chip_find_line
 - GPIO chip operations, [27](#)
- gpiod_chip_find_lines
 - GPIO chip operations, [28](#)
- gpiod_chip_get_all_lines
 - GPIO chip operations, [28](#)
- gpiod_chip_get_line
 - GPIO chip operations, [29](#)
- gpiod_chip_get_lines
 - GPIO chip operations, [29](#)
- gpiod_chip_iter_free
 - Iterators for GPIO chips and lines, [70](#)
- gpiod_chip_iter_free_noclose
 - Iterators for GPIO chips and lines, [70](#)
- gpiod_chip_iter_new
 - Iterators for GPIO chips and lines, [71](#)
- gpiod_chip_iter_next
 - Iterators for GPIO chips and lines, [71](#)
- gpiod_chip_iter_next_noclose
 - Iterators for GPIO chips and lines, [71](#)
- gpiod_chip_label
 - GPIO chip operations, [30](#)
- gpiod_chip_name
 - GPIO chip operations, [30](#)
- gpiod_chip_num_lines
 - GPIO chip operations, [30](#)
- gpiod_chip_open
 - GPIO chip operations, [31](#)
- gpiod_chip_open_by_label
 - GPIO chip operations, [31](#)
- gpiod_chip_open_by_name
 - GPIO chip operations, [32](#)
- gpiod_chip_open_by_number
 - GPIO chip operations, [32](#)
- gpiod_chip_open_lookup
 - GPIO chip operations, [32](#)
- GPIOD_CTXLESS_EVENT_CB_FALLING_EDGE
 - High-level API, [15](#)
- GPIOD_CTXLESS_EVENT_CB_RET_ERR
 - High-level API, [15](#)
- GPIOD_CTXLESS_EVENT_CB_RET_OK
 - High-level API, [15](#)
- GPIOD_CTXLESS_EVENT_CB_RET_STOP
 - High-level API, [15](#)
- GPIOD_CTXLESS_EVENT_CB_RISING_EDGE
 - High-level API, [15](#)
- GPIOD_CTXLESS_EVENT_CB_TIMEOUT
 - High-level API, [15](#)
- GPIOD_CTXLESS_EVENT_FALLING_EDGE
 - High-level API, [15](#)
- gpiod_ctxless_event_handle_cb
 - High-level API, [14](#)
- gpiod_ctxless_event_loop
 - High-level API, [16](#)
- gpiod_ctxless_event_loop_multiple
 - High-level API, [17](#)
- gpiod_ctxless_event_monitor
 - High-level API, [18](#)
- gpiod_ctxless_event_monitor_ext
 - High-level API, [18](#)
- gpiod_ctxless_event_monitor_multiple
 - High-level API, [19](#)
- gpiod_ctxless_event_monitor_multiple_ext
 - High-level API, [19](#)

- High-level API, [20](#)
- `gpiod_ctxless_event_poll_cb`
 - High-level API, [14](#)
- `gpiod_ctxless_event_poll_fd`, [75](#)
 - event, [75](#)
 - fd, [75](#)
- `GPIOD_CTXLESS_EVENT_POLL_RET_ERR`
 - High-level API, [16](#)
- `GPIOD_CTXLESS_EVENT_POLL_RET_STOP`
 - High-level API, [16](#)
- `GPIOD_CTXLESS_EVENT_POLL_RET_TIMEOUT`
 - High-level API, [16](#)
- `GPIOD_CTXLESS_EVENT_RISING_EDGE`
 - High-level API, [15](#)
- `gpiod_ctxless_find_line`
 - High-level API, [21](#)
- `GPIOD_CTXLESS_FLAG_BIAS_DISABLE`
 - High-level API, [15](#)
- `GPIOD_CTXLESS_FLAG_BIAS_PULL_DOWN`
 - High-level API, [15](#)
- `GPIOD_CTXLESS_FLAG_BIAS_PULL_UP`
 - High-level API, [15](#)
- `GPIOD_CTXLESS_FLAG_OPEN_DRAIN`
 - High-level API, [15](#)
- `GPIOD_CTXLESS_FLAG_OPEN_SOURCE`
 - High-level API, [15](#)
- `gpiod_ctxless_get_value`
 - High-level API, [22](#)
- `gpiod_ctxless_get_value_ext`
 - High-level API, [22](#)
- `gpiod_ctxless_get_value_multiple`
 - High-level API, [23](#)
- `gpiod_ctxless_get_value_multiple_ext`
 - High-level API, [23](#)
- `gpiod_ctxless_set_value`
 - High-level API, [24](#)
- `gpiod_ctxless_set_value_ext`
 - High-level API, [24](#)
- `gpiod_ctxless_set_value_multiple`
 - High-level API, [25](#)
- `gpiod_ctxless_set_value_multiple_ext`
 - High-level API, [25](#)
- `gpiod_foreach_chip`
 - Iterators for GPIO chips and lines, [69](#)
- `gpiod_foreach_chip_noclose`
 - Iterators for GPIO chips and lines, [69](#)
- `gpiod_foreach_line`
 - Iterators for GPIO chips and lines, [70](#)
- `gpiod_line_active_state`
 - Line info, [37](#)
- `GPIOD_LINE_ACTIVE_STATE_HIGH`
 - Line info, [37](#)
- `GPIOD_LINE_ACTIVE_STATE_LOW`
 - Line info, [37](#)
- `gpiod_line_bias`
 - Line info, [38](#)
- `GPIOD_LINE_BIAS_AS_IS`
 - Line info, [37](#)
- `GPIOD_LINE_BIAS_DISABLE`
 - Line info, [37](#)
- `GPIOD_LINE_BIAS_PULL_DOWN`
 - Line info, [37](#)
- `GPIOD_LINE_BIAS_PULL_UP`
 - Line info, [37](#)
- `gpiod_line_bulk`, [76](#)
 - lines, [76](#)
 - num_lines, [76](#)
- `gpiod_line_bulk_foreach_line`
 - Operating on multiple lines, [34](#)
- `gpiod_line_bulk_foreach_line_off`
 - Operating on multiple lines, [35](#)
- `GPIOD_LINE_BULK_INITIALIZER`
 - Operating on multiple lines, [35](#)
- `gpiod_line_close_chip`
 - Misc line functions, [67](#)
- `gpiod_line_consumer`
 - Line info, [38](#)
- `gpiod_line_direction`
 - Line info, [38](#)
- `GPIOD_LINE_DIRECTION_INPUT`
 - Line info, [37](#)
- `GPIOD_LINE_DIRECTION_OUTPUT`
 - Line info, [37](#)
- `gpiod_line_event`, [76](#)
 - event_type, [77](#)
 - ts, [77](#)
- `GPIOD_LINE_EVENT_FALLING_EDGE`
 - Line events handling, [63](#)
- `gpiod_line_event_get_fd`
 - Line events handling, [63](#)
- `gpiod_line_event_read`
 - Line events handling, [63](#)
- `gpiod_line_event_read_fd`
 - Line events handling, [64](#)
- `gpiod_line_event_read_fd_multiple`
 - Line events handling, [64](#)
- `gpiod_line_event_read_multiple`
 - Line events handling, [65](#)
- `GPIOD_LINE_EVENT_RISING_EDGE`
 - Line events handling, [63](#)
- `gpiod_line_event_wait`
 - Line events handling, [65](#)
- `gpiod_line_event_wait_bulk`
 - Line events handling, [65](#)
- `gpiod_line_find`
 - Misc line functions, [67](#)
- `gpiod_line_get`
 - Misc line functions, [67](#)
- `gpiod_line_get_chip`
 - Misc line functions, [68](#)
- `gpiod_line_get_value`
 - Reading & setting line values, [56](#)
- `gpiod_line_get_value_bulk`
 - Reading & setting line values, [56](#)
- `gpiod_line_is_free`
 - Line requests, [44](#)

- gpiod_line_is_open_drain
 - Line info, [39](#)
- gpiod_line_is_open_source
 - Line info, [39](#)
- gpiod_line_is_requested
 - Line requests, [45](#)
- gpiod_line_is_used
 - Line info, [40](#)
- gpiod_line_iter_free
 - Iterators for GPIO chips and lines, [72](#)
- gpiod_line_iter_new
 - Iterators for GPIO chips and lines, [72](#)
- gpiod_line_iter_next
 - Iterators for GPIO chips and lines, [73](#)
- gpiod_line_name
 - Line info, [40](#)
- gpiod_line_needs_update
 - Line info, [40](#)
- gpiod_line_offset
 - Line info, [41](#)
- gpiod_line_release
 - Line requests, [45](#)
- gpiod_line_release_bulk
 - Line requests, [45](#)
- gpiod_line_request
 - Line requests, [46](#)
- gpiod_line_request_both_edges_events
 - Line requests, [46](#)
- gpiod_line_request_both_edges_events_flags
 - Line requests, [47](#)
- gpiod_line_request_bulk
 - Line requests, [47](#)
- gpiod_line_request_bulk_both_edges_events
 - Line requests, [47](#)
- gpiod_line_request_bulk_both_edges_events_flags
 - Line requests, [48](#)
- gpiod_line_request_bulk_falling_edge_events
 - Line requests, [48](#)
- gpiod_line_request_bulk_falling_edge_events_flags
 - Line requests, [49](#)
- gpiod_line_request_bulk_input
 - Line requests, [49](#)
- gpiod_line_request_bulk_input_flags
 - Line requests, [50](#)
- gpiod_line_request_bulk_output
 - Line requests, [50](#)
- gpiod_line_request_bulk_output_flags
 - Line requests, [50](#)
- gpiod_line_request_bulk_rising_edge_events
 - Line requests, [51](#)
- gpiod_line_request_bulk_rising_edge_events_flags
 - Line requests, [51](#)
- gpiod_line_request_config, [77](#)
 - consumer, [78](#)
 - flags, [78](#)
 - request_type, [78](#)
- GPIOD_LINE_REQUEST_DIRECTION_AS_IS
 - Line requests, [44](#)
- GPIOD_LINE_REQUEST_DIRECTION_INPUT
 - Line requests, [44](#)
- GPIOD_LINE_REQUEST_DIRECTION_OUTPUT
 - Line requests, [44](#)
- GPIOD_LINE_REQUEST_EVENT_BOTH_EDGES
 - Line requests, [44](#)
- GPIOD_LINE_REQUEST_EVENT_FALLING_EDGE
 - Line requests, [44](#)
- GPIOD_LINE_REQUEST_EVENT_RISING_EDGE
 - Line requests, [44](#)
- gpiod_line_request_falling_edge_events
 - Line requests, [52](#)
- gpiod_line_request_falling_edge_events_flags
 - Line requests, [52](#)
- GPIOD_LINE_REQUEST_FLAG_ACTIVE_LOW
 - Line requests, [44](#)
- GPIOD_LINE_REQUEST_FLAG_BIAS_DISABLE
 - Line requests, [44](#)
- GPIOD_LINE_REQUEST_FLAG_BIAS_PULL_DOWN
 - Line requests, [44](#)
- GPIOD_LINE_REQUEST_FLAG_BIAS_PULL_UP
 - Line requests, [44](#)
- GPIOD_LINE_REQUEST_FLAG_OPEN_DRAIN
 - Line requests, [44](#)
- GPIOD_LINE_REQUEST_FLAG_OPEN_SOURCE
 - Line requests, [44](#)
- gpiod_line_request_input
 - Line requests, [52](#)
- gpiod_line_request_input_flags
 - Line requests, [53](#)
- gpiod_line_request_output
 - Line requests, [53](#)
- gpiod_line_request_output_flags
 - Line requests, [54](#)
- gpiod_line_request_rising_edge_events
 - Line requests, [54](#)
- gpiod_line_request_rising_edge_events_flags
 - Line requests, [54](#)
- gpiod_line_set_config
 - Setting line configuration, [58](#)
- gpiod_line_set_config_bulk
 - Setting line configuration, [59](#)
- gpiod_line_set_direction_input
 - Setting line configuration, [59](#)
- gpiod_line_set_direction_input_bulk
 - Setting line configuration, [59](#)
- gpiod_line_set_direction_output
 - Setting line configuration, [60](#)
- gpiod_line_set_direction_output_bulk
 - Setting line configuration, [60](#)
- gpiod_line_set_flags
 - Setting line configuration, [61](#)
- gpiod_line_set_flags_bulk
 - Setting line configuration, [61](#)
- gpiod_line_set_value
 - Reading & setting line values, [56](#)
- gpiod_line_set_value_bulk
 - Reading & setting line values, [57](#)

- gpiod_line_update
 - Line info, 41
- gpiod_version_string
 - Stuff that didn't fit anywhere else, 73
- High-level API, 12
 - GPIOD_CTXLESS_EVENT_CB_FALLING_EDGE, 15
 - GPIOD_CTXLESS_EVENT_CB_RET_ERR, 15
 - GPIOD_CTXLESS_EVENT_CB_RET_OK, 15
 - GPIOD_CTXLESS_EVENT_CB_RET_STOP, 15
 - GPIOD_CTXLESS_EVENT_CB_RISING_EDGE, 15
 - GPIOD_CTXLESS_EVENT_CB_TIMEOUT, 15
 - GPIOD_CTXLESS_EVENT_FALLING_EDGE, 15
 - gpiod_ctxless_event_handle_cb, 14
 - gpiod_ctxless_event_loop, 16
 - gpiod_ctxless_event_loop_multiple, 17
 - gpiod_ctxless_event_monitor, 18
 - gpiod_ctxless_event_monitor_ext, 18
 - gpiod_ctxless_event_monitor_multiple, 19
 - gpiod_ctxless_event_monitor_multiple_ext, 20
 - gpiod_ctxless_event_poll_cb, 14
 - GPIOD_CTXLESS_EVENT_POLL_RET_ERR, 16
 - GPIOD_CTXLESS_EVENT_POLL_RET_STOP, 16
 - GPIOD_CTXLESS_EVENT_POLL_RET_TIMEOUT, 16
 - GPIOD_CTXLESS_EVENT_RISING_EDGE, 15
 - gpiod_ctxless_find_line, 21
 - GPIOD_CTXLESS_FLAG_BIAS_DISABLE, 15
 - GPIOD_CTXLESS_FLAG_BIAS_PULL_DOWN, 15
 - GPIOD_CTXLESS_FLAG_BIAS_PULL_UP, 15
 - GPIOD_CTXLESS_FLAG_OPEN_DRAIN, 15
 - GPIOD_CTXLESS_FLAG_OPEN_SOURCE, 15
 - gpiod_ctxless_get_value, 22
 - gpiod_ctxless_get_value_ext, 22
 - gpiod_ctxless_get_value_multiple, 23
 - gpiod_ctxless_get_value_multiple_ext, 23
 - gpiod_ctxless_set_value, 24
 - gpiod_ctxless_set_value_ext, 24
 - gpiod_ctxless_set_value_multiple, 25
 - gpiod_ctxless_set_value_multiple_ext, 25
- Iterators for GPIO chips and lines, 68
 - gpiod_chip_iter_free, 70
 - gpiod_chip_iter_free_noclose, 70
 - gpiod_chip_iter_new, 71
 - gpiod_chip_iter_next, 71
 - gpiod_chip_iter_next_noclose, 71
 - gpiod_foreach_chip, 69
 - gpiod_foreach_chip_noclose, 69
 - gpiod_foreach_line, 70
 - gpiod_line_iter_free, 72
 - gpiod_line_iter_new, 72
 - gpiod_line_iter_next, 73
- Line events handling, 62
 - GPIOD_LINE_EVENT_FALLING_EDGE, 63
 - gpiod_line_event_get_fd, 63
 - gpiod_line_event_read, 63
 - gpiod_line_event_read_fd, 64
 - gpiod_line_event_read_fd_multiple, 64
 - gpiod_line_event_read_multiple, 65
 - GPIOD_LINE_EVENT_RISING_EDGE, 63
 - gpiod_line_event_wait, 65
 - gpiod_line_event_wait_bulk, 65
- Line info, 35
 - gpiod_line_active_state, 37
 - GPIOD_LINE_ACTIVE_STATE_HIGH, 37
 - GPIOD_LINE_ACTIVE_STATE_LOW, 37
 - gpiod_line_bias, 38
 - GPIOD_LINE_BIAS_AS_IS, 37
 - GPIOD_LINE_BIAS_DISABLE, 37
 - GPIOD_LINE_BIAS_PULL_DOWN, 37
 - GPIOD_LINE_BIAS_PULL_UP, 37
 - gpiod_line_consumer, 38
 - gpiod_line_direction, 38
 - GPIOD_LINE_DIRECTION_INPUT, 37
 - GPIOD_LINE_DIRECTION_OUTPUT, 37
 - gpiod_line_is_open_drain, 39
 - gpiod_line_is_open_source, 39
 - gpiod_line_is_used, 40
 - gpiod_line_name, 40
 - gpiod_line_needs_update, 40
 - gpiod_line_offset, 41
 - gpiod_line_update, 41
- Line requests, 42
 - gpiod_line_is_free, 44
 - gpiod_line_is_requested, 45
 - gpiod_line_release, 45
 - gpiod_line_release_bulk, 45
 - gpiod_line_request, 46
 - gpiod_line_request_both_edges_events, 46
 - gpiod_line_request_both_edges_events_flags, 47
 - gpiod_line_request_bulk, 47
 - gpiod_line_request_bulk_both_edges_events, 47
 - gpiod_line_request_bulk_both_edges_events_flags, 48
 - gpiod_line_request_bulk_falling_edge_events, 48
 - gpiod_line_request_bulk_falling_edge_events_flags, 49
 - gpiod_line_request_bulk_input, 49
 - gpiod_line_request_bulk_input_flags, 50
 - gpiod_line_request_bulk_output, 50
 - gpiod_line_request_bulk_output_flags, 50
 - gpiod_line_request_bulk_rising_edge_events, 51
 - gpiod_line_request_bulk_rising_edge_events_flags, 51
 - GPIOD_LINE_REQUEST_DIRECTION_AS_IS, 44
 - GPIOD_LINE_REQUEST_DIRECTION_INPUT, 44
 - GPIOD_LINE_REQUEST_DIRECTION_OUTPUT, 44
 - GPIOD_LINE_REQUEST_EVENT_BOTH_EDGES, 44

- [GPIO_LINE_REQUEST_EVENT_FALLING_EDGE](#), [44](#)
 - [GPIO_LINE_REQUEST_EVENT_RISING_EDGE](#), [44](#)
 - [gpiod_line_request_falling_edge_events](#), [52](#)
 - [gpiod_line_request_falling_edge_events_flags](#), [52](#)
 - [GPIO_LINE_REQUEST_FLAG_ACTIVE_LOW](#), [44](#)
 - [GPIO_LINE_REQUEST_FLAG_BIAS_DISABLE](#), [44](#)
 - [GPIO_LINE_REQUEST_FLAG_BIAS_PULL_DOWN](#), [44](#)
 - [GPIO_LINE_REQUEST_FLAG_BIAS_PULL_UP](#), [44](#)
 - [GPIO_LINE_REQUEST_FLAG_OPEN_DRAIN](#), [44](#)
 - [GPIO_LINE_REQUEST_FLAG_OPEN_SOURCE](#), [44](#)
 - [gpiod_line_request_input](#), [52](#)
 - [gpiod_line_request_input_flags](#), [53](#)
 - [gpiod_line_request_output](#), [53](#)
 - [gpiod_line_request_output_flags](#), [54](#)
 - [gpiod_line_request_rising_edge_events](#), [54](#)
 - [gpiod_line_request_rising_edge_events_flags](#), [54](#)
- [lines](#)
 - [gpiod_line_bulk](#), [76](#)
- [Misc line functions](#), [66](#)
 - [gpiod_line_close_chip](#), [67](#)
 - [gpiod_line_find](#), [67](#)
 - [gpiod_line_get](#), [67](#)
 - [gpiod_line_get_chip](#), [68](#)
- [num_lines](#)
 - [gpiod_line_bulk](#), [76](#)
- [Operating on multiple lines](#), [34](#)
 - [gpiod_line_bulk_foreach_line](#), [34](#)
 - [gpiod_line_bulk_foreach_line_off](#), [35](#)
 - [GPIO_LINE_BULK_INITIALIZER](#), [35](#)
- [Reading & setting line values](#), [55](#)
 - [gpiod_line_get_value](#), [56](#)
 - [gpiod_line_get_value_bulk](#), [56](#)
 - [gpiod_line_set_value](#), [56](#)
 - [gpiod_line_set_value_bulk](#), [57](#)
- [request_type](#)
 - [gpiod_line_request_config](#), [78](#)
- [Setting line configuration](#), [57](#)
 - [gpiod_line_set_config](#), [58](#)
 - [gpiod_line_set_config_bulk](#), [59](#)
 - [gpiod_line_set_direction_input](#), [59](#)
 - [gpiod_line_set_direction_input_bulk](#), [59](#)
 - [gpiod_line_set_direction_output](#), [60](#)
 - [gpiod_line_set_direction_output_bulk](#), [60](#)
 - [gpiod_line_set_flags](#), [61](#)
 - [gpiod_line_set_flags_bulk](#), [61](#)
- [Stuff that didn't fit anywhere else](#), [73](#)