LwBTN

Tilen MAJERLE

Oct 31, 2022

CONTENTS

Welcome to the documentation for version latest-develop.

Download library Getting started Open Github Donate

CHAPTER

ONE

FEATURES

- Written in ANSI C99
- Platform independent, requires user to provide millisecond timing source
- No dynamic memory allocation
- Callback driven event management
- Easy to use and maintain
- User friendly MIT license

CHAPTER

TWO

REQUIREMENTS

- C compiler
- Few kB of non-volatile memory

CHAPTER

THREE

CONTRIBUTE

Fresh contributions are always welcome. Simple instructions to proceed:

1. Fork Github repository
2. Respect C style & coding rules used by the library
3. Create a pull request to `develop` branch with new features or bug fixes

Alternatively you may:

1. Report a bug
2. Ask for a feature request

CHAPTER

FOUR

LICENSE

```
MIT License

Copyright (c) 2022 Tilen MAJERLE

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER

FIVE

TABLE OF CONTENTS

5.1 Getting started

Getting started may be the most challenging part of every new library. This guide is describing how to start with the library quickly and effectively

5.1.1 Download library

Library is primarly hosted on Github.

You can get it by:

- Downloading latest release from releases area on Github
- Cloning `main` branch for latest stable version
- Cloning `develop` branch for latest development

Download from releases

All releases are available on Github releases area.

Clone from Github

First-time clone

This is used when you do not have yet local copy on your machine.

- Make sure `git` is installed.
- Open console and navigate to path in the system to clone repository to. Use command `cd your_path`
- Clone repository with one of available options below
  - Run `git clone --recurse-submodules https://github.com/MaJerle/lwbtn` command to clone entire repository, including submodules
  - Run `git clone --recurse-submodules --branch develop https://github.com/MaJerle/lwbtn` to clone *development* branch, including submodules
  - Run `git clone --recurse-submodules --branch main https://github.com/MaJerle/lwbtn` to clone *latest stable* branch, including submodules
- Navigate to `examples` directory and run favourite example

Update cloned to latest version

- Open console and navigate to path in the system where your repository is located. Use command `cd your_path`
- Run `git pull origin main` command to get latest changes on `main` branch
- Run `git pull origin develop` command to get latest changes on `develop` branch
- Run `git submodule update --init --remote` to update submodules to latest version

Note: This is preferred option to use when you want to evaluate library and run prepared examples. Repository consists of multiple submodules which can be automatically downloaded when cloning and pulling changes from root repository.

5.1.2 Add library to project

At this point it is assumed that you have successfully download library, either cloned it or from releases page. Next step is to add the library to the project, by means of source files to compiler inputs and header files in search path

- Copy `lwbtn` folder to your project, it contains library files
- Add `lwbtn/src/include` folder to *include path* of your toolchain. This is where *C/C++* compiler can find the files during compilation process. Usually using `-I` flag
- Add source files from `lwbtn/src/` folder to toolchain build. These files are built by *C/C++* compiler
- Copy `lwbtn/src/include/lwbtn/lwbtn_opts_template.h` to project folder and rename it to `lwbtn_opts.h`
- Copy `lwbtn/src/include/lwbtn/lwbtn_types_template.h` to project folder and rename it to `lwbtn_types.h`
- Build the project

5.1.3 Configuration file

Configuration file is used to overwrite default settings defined for the essential use case. Library comes with template config file, which can be modified according to needs. and it should be copied (or simply renamed in-place) and named `lwbtn_opts.h`

Note: Default configuration template file location: `lwbtn/src/include/lwbtn/lwbtn_opts_template.h`. File must be renamed to `lwbtn_opts.h` first and then copied to the project directory where compiler include paths have access to it by using `#include "lwbtn_opts.h"`.

List of configuration options are available in the *Configuration* section. If any option is about to be modified, it should be done in configuration file

Listing 1: Template configuration file

```
1 /**
2  * \file            lwbtn_opts_template.h
3  * \brief           LwBTN configuration file
4  */
```

(continues on next page)

(continued from previous page)

```

/*
 * Copyright (c) 2022 Tilen MAJERLE
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge,
 * publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * This file is part of LwBTN - Lightweight button manager.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v0.0.1
 */
#ifndef LWBTN_HDR_OPTS_H
#define LWBTN_HDR_OPTS_H

/* Rename this file to "lwbtn_opts.h" for your application */

/*
 * Open "include/lwbtn/lwbtn_opt.h" and
 * copy & replace here settings you want to change values
 */

#endif /* LWBTN_HDR_OPTS_H */
```

Note: If you prefer to avoid using configuration file, application must define a global symbol `LWBTN_IGNORE_USER_OPTS`, visible across entire application. This can be achieved with `-D` compiler option.

5.2 User manual

LwBTN is simple button manager library, with great focus on embedded systems. Motivation behind start of development was linked to several on-going projects including some input reading (button handling), each of them demanding little differences in process.

LwBTN is therefore relatively simple and lightweight, yet it can provide pretty comprehensive processing of your application buttons.

5.2.1 How it works

User must define buttons array and pass it to the library. Next to that, 2 more functions are required:

- Function to read the architecture button state
- Function to receive various button events

User shall later periodically call processing function with current system time as simple parameter and get ready to receive various events.

A simple example for win32 is below:

Listing 2: Win32 example code

```c
#include "lwbtn/lwbtn.h"
#include "windows.h"
#include <stdio.h>
#include <stdlib.h>

static LARGE_INTEGER freq, sys_start_time;
static uint32_t get_tick(void);

/* User defined settings */
const int keys[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};
uint32_t last_time_keys[sizeof(keys) / sizeof(keys[0])] = {0};

/* List of buttons to process with assigned custom arguments for callback functions */
static lwbtn_btn_t btns[] = {{.arg = (void*)&keys[0]}, {.arg = (void*)&keys[1]}, {.arg = (void*)&keys[2]},
                             {.arg = (void*)&keys[3]}, {.arg = (void*)&keys[4]}, {.arg = (void*)&keys[5]},
                             {.arg = (void*)&keys[6]}, {.arg = (void*)&keys[7]}, {.arg = (void*)&keys[8]},
                             {.arg = (void*)&keys[9]}};

/**
 * \brief           Get input state callback
 * \param           lw: LwBTN instance
 * \param           btn: Button instance
 * \return          `1` if button active, `0` otherwise
 */
uint8_t
prv_btn_get_state(struct lwbtn* lw, struct lwbtn_btn* btn) {
    (void)lw;
```

(continues on next page)

(continued from previous page)

```
    /*
     * Function will return negative number if button is pressed,
     * or zero if button is releases
     */
    return GetAsyncKeyState(*(int*)btn->arg) < 0;
}

/**
 * \brief           Button event
 *
 * \param           lw: LwBTN instance
 * \param           btn: Button instance
 * \param           evt: Button event
 */
void
prv_btn_event(struct lwbtn* lw, struct lwbtn_btn* btn, lwbtn_evt_t evt) {
    const char* s;
    uint32_t color;
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    uint32_t* diff_time_ptr = &last_time_keys[(*(int*)btn->arg) - '0'];
    uint32_t diff_time = get_tick() - *diff_time_ptr;

    /* This is for purpose of test and timing validation */
    if (diff_time > 2000) {
        diff_time = 0;
    }
    *diff_time_ptr = get_tick(); /* Set current date as last one */

    /* Get event string */
    if (evt == LWBTN_EVT_KEEPALIVE) {
        s = "KEEPALIVE";
        color = FOREGROUND_RED;
    } else if (evt == LWBTN_EVT_ONPRESS) {
        s = "  ONPRESS";
        color = FOREGROUND_GREEN;
    } else if (evt == LWBTN_EVT_ONRELEASE) {
        s = "ONRELEASE";
        color = FOREGROUND_BLUE;
    } else if (evt == LWBTN_EVT_ONCLICK) {
        s = "  ONCLICK";
        color = FOREGROUND_RED | FOREGROUND_GREEN;
    } else {
        s = "  UNKNOWN";
        color = FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE;
    }
    SetConsoleTextAttribute(hConsole, color);
    printf("[%7u][%6u] CH: %c, evt: %s, keep-alive cnt: %3u, click cnt: %3u\r\n",␣
↪(unsigned)get_tick(),
           (unsigned)diff_time, *(int*)btn->arg, s, (unsigned)btn->keepalive.cnt,␣
↪(unsigned)btn->click.cnt);
    SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_
↪BLUE);
```

(continues on next page)

(continued from previous page)

```
    (void)lw;
}

/**
 * \brief           Example function
 */
int
example_win32(void) {
    printf("Application running\r\n");
    QueryPerformanceFrequency(&freq);
    QueryPerformanceCounter(&sys_start_time);

    /* Define buttons */
    lwbtn_init_ex(NULL, btns, sizeof(btns) / sizeof(btns[0]), prv_btn_get_state, prv_btn_
↪event);

    while (1) {
        /* Process forever */
        lwbtn_process_ex(NULL, get_tick());

        /* Artificial sleep to offload win process */
        Sleep(5);
    }
    return 0;
}

/**
 * \brief           Get current tick in ms from start of program
 * \return          uint32_t: Tick in ms
 */
static uint32_t
get_tick(void) {
    LONGLONG ret;
    LARGE_INTEGER now;

    QueryPerformanceFrequency(&freq);
    QueryPerformanceCounter(&now);
    ret = now.QuadPart - sys_start_time.QuadPart;
    return (uint32_t)((ret * 1000) / freq.QuadPart);
}
```

5.2.2 Input events

During button (or input if you will) lifetime, application can expect some of these events (but not limited to):

- `LWBTN_EVT_ONPRESS` event is sent to application whenever input goes from inactive to active state and minimum debounce time passes by
- `LWBTN_EVT_ONRELEASE` event is sent to application whenever input sent **onpress** event prior to that and when input goes from active to inactive state
- `LWBTN_EVT_KEEPALIVE` event is periodically sent between **onpress** and **onrelease** events

- `LWBTN_EVT_ONCLICK` event is sent after **onrelease** and only if active button state was within allowed window for valid click event.

5.2.3 On-Press event

Onpress event is the first in a row when input is detected active. With nature of embedded systems and various buttons connected to devices, it is necessary to filter out potential noise to ignore unintential multiple presses. This is done by checking line to be at stable level for at least some minimum time, normally called *debounce time*, usually it takes around `20ms`.

Fig. 1: On-Press event trigger after minimum debounce time

5.2.4 On-Release event

Onrelease event is triggered immediately when input goes from active to inactive state, and only if onpress event has been detected prior to that.

Fig. 2: On-Release event trigger

5.2.5 On-Click event

Onclick event is triggered after a combination of multiple events:

- **Onpress** event shall be detected properly, indicating button has been pressed
- **Onrelease** event shall be detected, indicating button has been released
- Time between **onpress** and **onrelease** events has to be within time window

When conditions are met, **onclick** event is sent, either immediately after **onrelease** or after certain timeout after **onrelease** event.

Fig. 3: Sequence for valid click event

A windows-test program demonstration of events is visible below.

Second number for each line is a **milliseconds** difference between events. OnClick is reported approximately (windows real-time issue) `400` ms after on-release event.

Tip: Timeout window between last **onrelease** event and **onclick** event is configurable

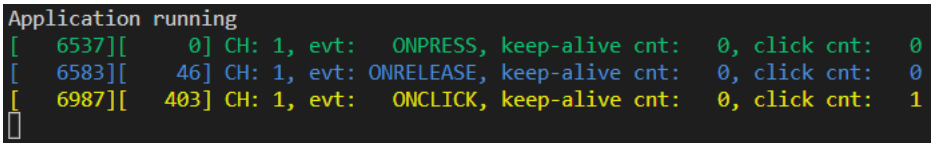

Fig. 4: Click event test program

5.2.6 Multi-click events

Multi-click feature is where **timeout** for **onclick** event comes into play. Idea behind timeout feature is to allow multiple presses and to only send **onclick** once for all presses, including the number of detected presses during that time. This let's the application to react only once with known number of presses. This eliminates the problem where in case of **double** click trigger, you also receive **single-click** event, while you do not know yet, if second (or third) event will be triggered after.

Note: Imagine having a button that toggles one light on single click and turns off all lights in a room on double click. With timeout feature and single **onclick** notification, user will only receive the **onclick** once and will, based on the consecutive presses number value, perform appropriate action if it was single or multi click.

Simplified diagram for multi-click, ignoring debounce time indicators, is below. **cp** indicates number of detected **consecutive onclick press** events, to be reported in the final **onclick** event

Fig. 5: Multi-click event example - with 3 consecutive presses

A windows-test program demonstration of events is visible below.

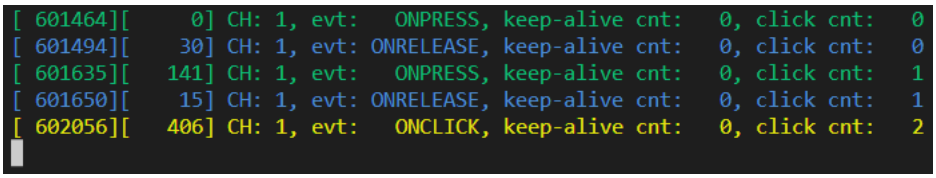

Fig. 6: Multi-click event test program

Multi-click event with **onclick** event reported only after second press after minimum timeout of `400ms`.

Note: Number of consecutive clicks can be upper-limited to the desired value.

When user makes more (or equal) consecutive clicks than maximum, an **onclick** event is sent immediately after **onrelease** event for last detected click.

There is no need to wait timeout expiration since upper clicks limit has been reached.

Tip: It is possible to control the behavior of **onclick** event (when consecutive number reaches maximum set value) timing using *LWBTN_CFG_CLICK_MAX_CONSECUTIVE_SEND_IMMEDIATELY* configuration. When enabled, behavior is as illustrated above. When disabled, **onclick** event it sent in timeout (or in case of new onpress), even if max allowed clicks has been reached.

Illustration below shows what happens during multiple clicks

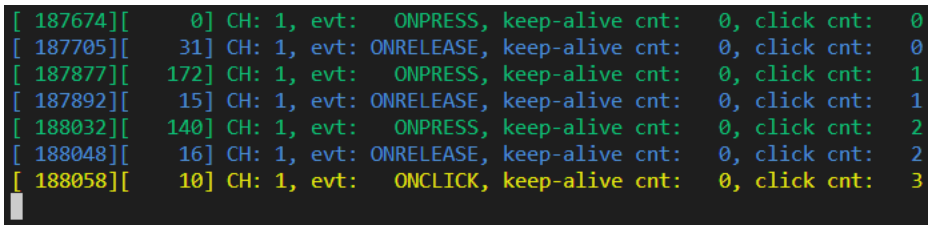

Fig. 7: Max number of onclick events, onclick is sent immediately after onrelease

- Max number of consecutive clicks is 3
- User makes 4 consecutive clicks

Fig. 8: Multi-click events with too many clicks - consecutive send immediately is enabled - it is sent after 3rd onrelease

Fig. 9: Multi-click events with too many clicks - consecutive send immediately is disabled

Image below illustrates when send immediately is enabled. It is visible how first **onclick** is sent just after **onrelease** event (when max consecutive is set to 3).

When **multi-click** feature is disabled, **onclick** event is sent after every valid sequence of **onpress** and **onrelease** events.

Tip: If you do not want multi-click feature, set max number of consecutive clicks to 1. This will eliminate timeout feature since every click event will trigger **maximum clicks detected** and therefore send the event immediately after **onrelease**

Demo log text, with fast pressing of button, and events reported after every **onrelease**

5.2.7 Keep alive event

Keep-alive event is sent periodically between **onpress** and **onrelease** events. It can be used to detect application is still alive and provides counter how many keep-alive events have been sent up to the point of event.

Feature can be used to make a trigger at specific time if button is in active state (a hold event).

5.3 API reference

List of all the modules:

```
[ 242253][     0] CH: 1, evt:   ONPRESS, keep-alive cnt:   0, click cnt:   0
[ 242284][    31] CH: 1, evt: ONRELEASE, keep-alive cnt:   0, click cnt:   0
[ 242455][   170] CH: 1, evt:   ONPRESS, keep-alive cnt:   0, click cnt:   1
[ 242485][    31] CH: 1, evt: ONRELEASE, keep-alive cnt:   0, click cnt:   1
[ 242673][   188] CH: 1, evt:   ONPRESS, keep-alive cnt:   0, click cnt:   2
[ 242704][    31] CH: 1, evt: ONRELEASE, keep-alive cnt:   0, click cnt:   2
[ 242709][     5] CH: 1, evt:   ONCLICK, keep-alive cnt:   0, click cnt:   3
[ 243125][   200] CH: 1, evt:   ONPRESS, keep-alive cnt:   0, click cnt:   1
[ 243171][    46] CH: 1, evt: ONRELEASE, keep-alive cnt:   0, click cnt:   1
[ 243578][   407] CH: 1, evt:   ONCLICK, keep-alive cnt:   0, click cnt:   2
```

Fig. 10: 5 presses detected with 3 set as maximum. First on-click is sent immediately, while second is sent after timeout

Fig. 11: Multi-click events disabled with cp == 1

```
[  15823][   152] CH: 1, evt:   ONPRESS, keep-alive cnt:   0, click cnt:   0
[  15839][    15] CH: 1, evt: ONRELEASE, keep-alive cnt:   0, click cnt:   0
[  15853][    15] CH: 1, evt:   ONCLICK, keep-alive cnt:   0, click cnt:   1
[  16007][   154] CH: 1, evt:   ONPRESS, keep-alive cnt:   0, click cnt:   0
[  16023][    16] CH: 1, evt: ONRELEASE, keep-alive cnt:   0, click cnt:   0
[  16028][     5] CH: 1, evt:   ONCLICK, keep-alive cnt:   0, click cnt:   1
[  16180][   151] CH: 1, evt:   ONPRESS, keep-alive cnt:   0, click cnt:   0
[  16211][    32] CH: 1, evt: ONRELEASE, keep-alive cnt:   0, click cnt:   0
[  16216][     5] CH: 1, evt:   ONCLICK, keep-alive cnt:   0, click cnt:   1
```

Fig. 12: Multi-click events disabled with cp == 1

Fig. 13: Keep alive events with 2 successful click events

```
[ 280771][     0] CH: 1, evt:   ONPRESS, keep-alive cnt:   0, click cnt:   0
[ 280880][   109] CH: 1, evt: KEEPALIVE, keep-alive cnt:   1, click cnt:   0
[ 280975][    95] CH: 1, evt: KEEPALIVE, keep-alive cnt:   2, click cnt:   0
[ 281084][   109] CH: 1, evt: KEEPALIVE, keep-alive cnt:   3, click cnt:   0
[ 281177][    93] CH: 1, evt: KEEPALIVE, keep-alive cnt:   4, click cnt:   0
[ 281271][    94] CH: 1, evt: KEEPALIVE, keep-alive cnt:   5, click cnt:   0
[ 281382][   111] CH: 1, evt: KEEPALIVE, keep-alive cnt:   6, click cnt:   0
[ 281475][    93] CH: 1, evt: KEEPALIVE, keep-alive cnt:   7, click cnt:   0
[ 281585][   110] CH: 1, evt: KEEPALIVE, keep-alive cnt:   8, click cnt:   0
[ 281678][    93] CH: 1, evt: KEEPALIVE, keep-alive cnt:   9, click cnt:   0
[ 281772][    94] CH: 1, evt: KEEPALIVE, keep-alive cnt:  10, click cnt:   0
[ 281849][    77] CH: 1, evt: ONRELEASE, keep-alive cnt:  10, click cnt:   0
```

Fig. 14: Keep alive events when button is kept pressed

5.3.1 LwBTN

group **LWBTN**

Lightweight button manager.

Defines

lwbtn_init(btns, btns_cnt, get_state_fn, evt_fn)

Initialize LwBTN library with buttons on default button group.

See also:

lwbtn_init_ex

Parameters

- **btns** – **[in]** Array of buttons to process
- **btns_cnt** – **[in]** Number of buttons to process
- **get_state_fn** – **[in]** Pointer to function providing button state on demand
- **evt_fn** – **[in]** Button event function callback

lwbtn_process(mstime)

Periodically read button states and take appropriate actions.

See also:

lwbtn_process_ex

Parameters

- **mstime** – **[in]** Current system time in milliseconds

Typedefs

typedef void (***lwbtn_evt_fn**)(struct lwbtn *lw, struct lwbtn_btn *btn, *lwbtn_evt_t* evt)

Button event function callback prototype.

Param lw **[in]** LwBTN instance

Param btn **[in]** Button instance from array for which event occured

Param evt **[in]** Event type

typedef uint8_t (***lwbtn_get_state_fn**)(struct lwbtn *lw, struct lwbtn_btn *btn)

Get button/input state callback function.

Param lw **[in]** LwBTN instance

Param btn **[in]** Button instance from array to read state

Return 1 when button is considered `active`, `0` otherwise

Enums

enum **lwbtn_evt_t**

List of button events.

Values:

enumerator **LWBTN_EVT_ONPRESS** = 0x00

On press event - sent when valid press is detected (after debounce if enabled)

enumerator **LWBTN_EVT_ONRELEASE**

On release event - sent when valid release event is detected (from active to inactive)

enumerator **LWBTN_EVT_ONCLICK**

On Click event - sent when valid sequence of on-press and on-release events occurs

enumerator **LWBTN_EVT_KEEPALIVE**

Keep alive event - sent periodically when button is active

Functions

uint8_t **lwbtn_init_ex**(*lwbtn_t* *lw, *lwbtn_btn_t* *btns, uint16_t btns_cnt, *lwbtn_get_state_fn* get_state_fn, *lwbtn_evt_fn* evt_fn)

Initialize button manager.

Parameters

- **lw** – **[in]** LwBTN instance. Set to `NULL` to use default one
- **btns** – **[in]** Array of buttons to process
- **btns_cnt** – **[in]** Number of buttons to process
- **get_state_fn** – **[in]** Pointer to function providing button state on demand
- **evt_fn** – **[in]** Button event function callback

Returns 1 on success, `0` otherwise

uint8_t **lwbtn_process_ex**(*lwbtn_t* *lw, uint32_t mstime)

Button processing function, that reads the inputs and makes actions accordingly.

Parameters **mstime** – **[in]** Current time in milliseconds

Returns 1 on success, `0` otherwise

struct **lwbtn_argdata_port_pin_state_t**

#include <lwbtn.h> Custom user argument data structure.

This is a simple pre-defined structure, that can be used by user to define most commonly required feature in embedded systems, that being GPIO port, GPIO pin and state when button is considered active.

User can later attach this structure as argument to button structure

Public Members

void *port

User defined GPIO port information

void *pin

User defined GPIO pin information

uint8_t state

User defined GPIO state level when considered active

struct lwbtn_btn_t

#include <*lwbtn.h*> Button/input structure.

Public Members

uint16_t flags

Private button flags management

uint8_t old_state

Old button state - 1 means active, 0 means inactive

uint32_t time_change

Time in ms when button state got changed last time

uint32_t last_time

Time in ms of last send keep alive event

Time in ms of last successfully detected (not sent!) click event

uint16_t cnt

Number of keep alive events sent after successful on-press detection. Value is reset after on-release

struct *lwbtn_btn_t*::[anonymous] keepalive

Keep alive structure

uint8_t cnt

Number of consecutive clicks detected, respecting maximum timeout between clicks

struct *lwbtn_btn_t*::[anonymous] click

Click event structure

void *arg

User defined custom argument for callback function purpose

struct **lwbtn_t**

#include <*lwbtn.h*> LwBTN group structure.

Public Members

lwbtn_btn_t ***btns**

Pointer to buttons array

uint16_t **btns_cnt**

Number of buttons in array

lwbtn_evt_fn **evt_fn**

Pointer to event function

lwbtn_get_state_fn **get_state_fn**

Pointer to get state function

5.3.2 Configuration

This is the default configuration of the middleware. When any of the settings shall be modified, it shall be done in dedicated application config `lwbtn_opts.h` file.

Note: Check *Getting started* for guidelines on how to create and use configuration file.

group **LWBTN_OPT**

Default configuration setup.

Defines

LWBTN_CFG_TIME_DEBOUNCE

Minimum debounce time in units of milliseconds.

This is the time input shall have stable level to detect valid *onpress* event

LWBTN_CFG_TIME_CLICK_MIN

Minimum active input time for valid click event, in milliseconds.

Input shall be pressed at least this amount of time to even consider the potential valid click event. Set the value to `0` to disable this feature

LWBTN_CFG_TIME_CLICK_MAX

Maximum active input time for valid click event, in milliseconds.

Input shall be pressed at most this amount of time to still trigger valid click. Set to `-1` to allow any time triggering click event.

LWBTN_CFG_TIME_CLICK_MULTI_MAX

Maximum allowed time between last on-release and next valid on-press, to still allow multi-click events, in milliseconds.

This value is also used as a timeout length. It sends *onclick* event if there is no further presses by the application.

LWBTN_CFG_CLICK_MAX_CONSECUTIVE

Maximum number of allowed consecutive click events, before structure gets reset to default value.

LWBTN_CFG_TIME_KEEPALIVE_PERIOD

Keep-alive event period, in milliseconds.

LWBTN_CFG_CLICK_MAX_CONSECUTIVE_SEND_IMMEDIATELY

Enables `1` or disables `0` immediate onclick event after on-release event, if number of consecutive clicks reaches max value.

When this mode is disabled, onclick is sent in one of 2 cases:

- An on-click timeout occurred
- Next on-press event occurred before timeout expired

5.4 Changelog

```
# Changelog

## Develop

- First commit
```

INDEX

L