MyButton Arduino Library

Release 1.2.0

Rad-hi

Mar 10, 2022

CONTENTS:

CHAPTER

ONE

GETTING STARTED WITH MYBUTTON LIBRARY

MyButton was originally created to make interacting with push-buttons easier for makers of interactive embedded projects. Down the line, it was extended with a second library **MyCountingButton**, which is dedicated for counting presses and interactions that are linked with counting presses in general.

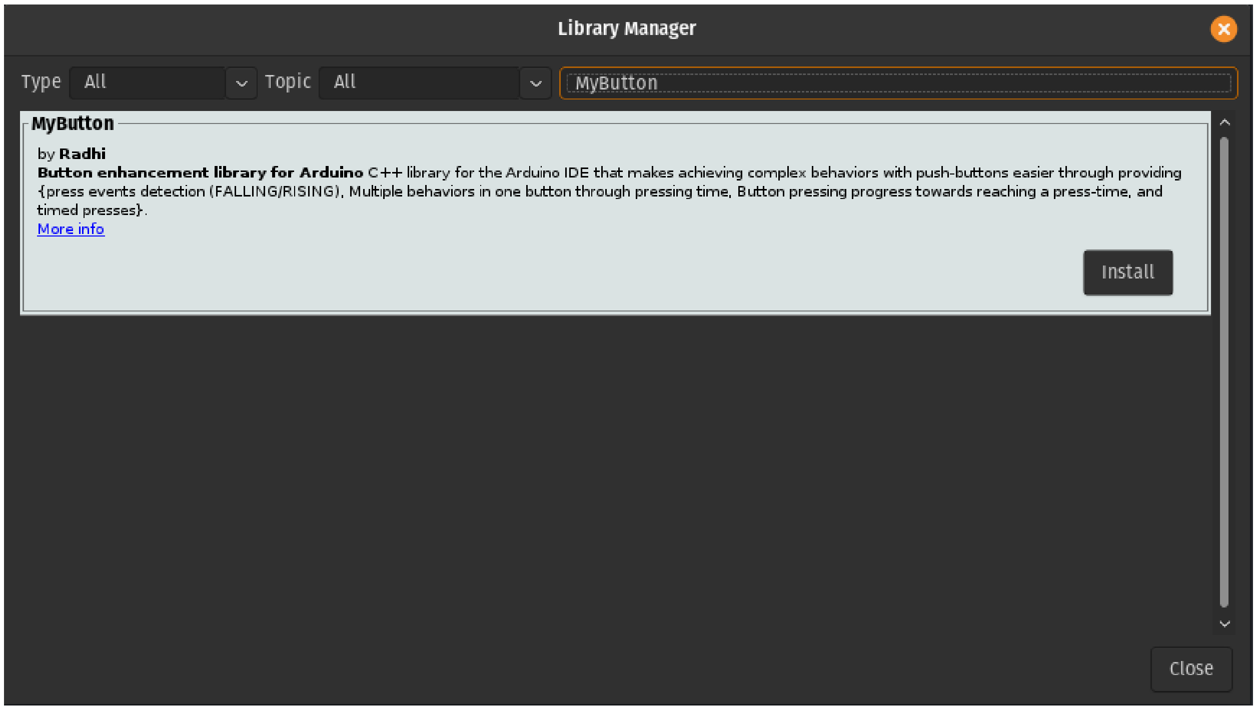
If you have a project that has push-botton(s) in it, or are thinking of one, and would like to make the interactions with this project a bit more polished and complex, whilst not having to deal with the problems that arise from working with push-buttons, such as bouncing, then MyButton was made for you!

Check the source code from here: https://github.com/Rad-hi/MyButton_Arduino

1.1 1. How to install ?

1.1.1 1.1. Through the Arduino Library manager:

Go to **tools -> Manage Libraries… ->** type **MyButton** and as shown in the picture below, you'll find the MyButton library.



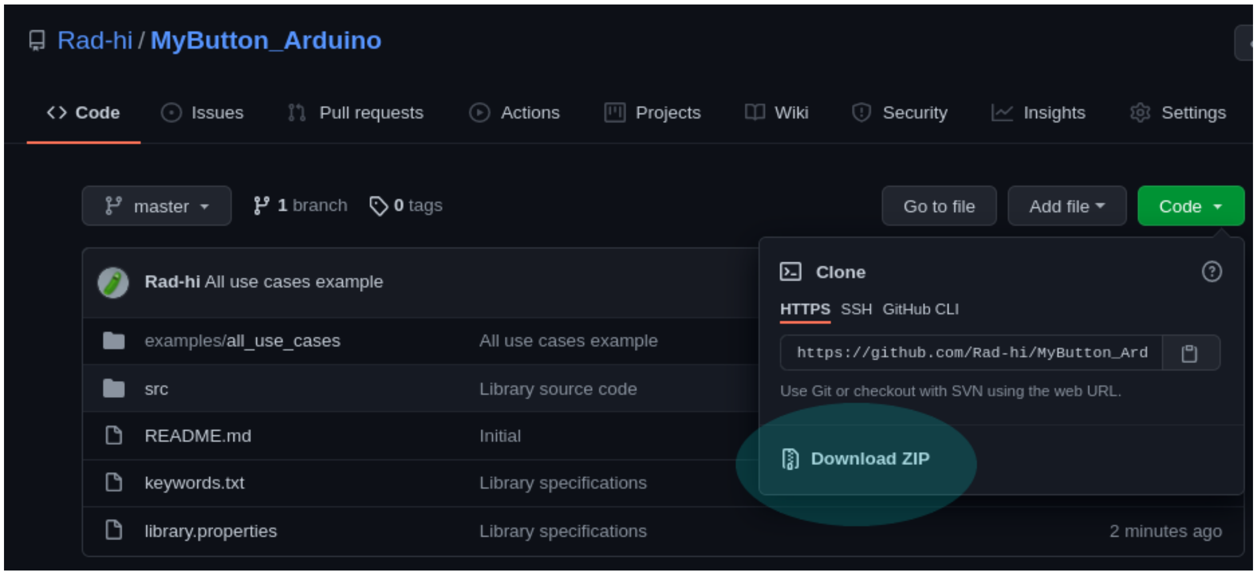
Type All Topic All MyButton

MyButton

by **Radhi**

Button enhancement library for Arduino C++ library for the Arduino IDE that makes achieving complex behaviors with push-buttons easier through providing {press events detection (FALLING/RISING), Multiple behaviors in one button through pressing time, Button pressing progress towards reaching a press-time, and timed presses}.

1.1.2 1.2. Direct download/import:



Rad-hi All use cases example

examples/all_use_cases	All use cases example	
src	Library source code	
README.md	Initial	
keywords.txt	Library specifications	
library.properties	Library specifications	2 minutes ago

`https://github.com/Rad-hi/MyButton_Ard`

Use Git or checkout with SVN using the web URL.

Click on **Download ZIP** to download the library, place the unzipped folder into your libraries folder in your **arduinos-ketchfolder/libraries/PUT_HERE**(on Windows, this is likely to be under **Documents/Arduino/libraries**, on linux this is under **home/Arduino/libraries**). You may need to create the libraries subfolder if its your first library. Now, restart the IDE.

1.2 2. What's next?

For individual documentations for each of the libraries (methods, how to use, objects, code examples, . . .), Go to the MyButton/MyCountingButton pages.

CHAPTER

TWO

MYBUTTON.H, A COMPREHENSIVE GUIDE

2.1 1. What does the library offer?

- Detection of pressing events on both the **RISING/FALLING** edges with debouncing.
- Association of multiple behaviors for a single button by creating multiple "checkpoints" for the button to report reaching **upon release**.
- Getting the progress of the pushed button **towards a target period** in a number of **specified steps**.
- Getting the time the button have been clicked for in 3 units (seconds, milliseconds, microseconds).

All happening in a non-blocking manner.

2.2 2. How to use?

The workflow is basic, you first instanciate an object `MyButton my_button` with one of the customization options that are availabe, and then you call of the methods associated with the object `my_button.__method_name__()` in the `loop(){ }`.

2.2.1 2.1. Instanciation of a button

- Default debouncing time of **5ms**:
  - `MyButton my_button(MY_BUTTON_PIN, NORMAL_UP);`
  - `MyButton my_button(MY_BUTTON_PIN, NORMAL_DOWN);`
- Custom debouncing time:
  - `MyButton my_button(MY_BUTTON_PIN, NORMAL_UP, MY_CUSTOM_DEBOUNCING_TIME_IN_MILLISECONDS);`

The `NORMAL_UP`, and `NORMAL_DOWN` keywords refer to whether the push button is normally UP or DOWN (pulled UP or DOWN).

2.2.2 2.2. Available methods

- **`bool readRawClick();`**
    - Returns whether the button was pressed or not **NON-DEBOUNCED**
- `bool readRisingClick();`
    - Returns a boolean value corresponding to the occurrence of a `rising` edge on the button pin.
- `bool readFallingClick();`
    - Returns a boolean value corresponding to the occurrence of a `falling` edge on the button pin.
- `uint32_t readTimedPress(uint8_t unit);`
    - Returns the time the button has been clicked for in one of 3 `units` (micros, millis, seconds) **NON-DEBOUNCED**
- `uint8_t readInSteps(uint32_t period, uint8_t num_steps);`
    - This function takes in a `period` in **milliseconds** and a `number of steps`, and on each step of that period `step == period/num_steps`, returns the `index` of the step, **0 INDEXED**, else returns NON_CLICKED (==255).
- `uint8_t readInProvidedSteps(uint32_t * periods, uint8_t num_steps);`
    - This function takes in an **incrementally sorted list of ``periods`` in milliseconds**, and reports the `index` of the reached period in the list, else it returns NON_CLICKED (==255). **ZERO-INDEXED**
- `uint8_t readMultiple(uint32_t * periods, uint8_t len);`
    - This function takes in an **incrementally sorted list of ``periods`` in milliseconds**, and if the button have been pressed for more than one of the periods (CHECKED ON RELEASE), it'd return the `index` of the period in the list, else it returns NON_CLICKED (==255). **ZERO-INDEXED**

2.2.3 2.3. Notes

- In case you choose to do a NORMAL_DOWN button, make sure to externally pull down the push-button, otherwise, an internal pull-up resistor is used by default to the NORMAL_UP mode.
- For saving on resources, and since there's no apparent use-case where someone would configure one button to exert more than one of the behaviors possible through the functions, the time tracking and the state variables are **shared** between functions, so calling _ as an example _ `my_button.readRisingClick()`, and `my_button.readFallingClick` back to back in the same loop would make the code behave unpredictibly.

CHAPTER

THREE

MYCOUNTINGBUTTON.H, A COMPREHENSIVE GUIDE

3.1 1. What does the library offer?

- Couting of clicks on the RISING/FALLING/CHANGING edges
- Couting on hardware interruption enabled pins (expl: for encodes), or any "normal" pin
- Triggering of a custom callback function when a set value is reached
- Set the counting direction (++/--), and set a custom value to count from

All happening in a non-blocking manner.

3.2 2. How to use?

There's a number of options availabe when it comes to creating a counting button. The subject that we'll discuss in the next section.

3.2.1 2.1. Instanciation of a counting button

The instanciation is simple, `MyCountingButton my_counting_btn` and now you have a counting button, but that's not enough to start using the methods, since this objects requires beginning, and here is where the options arise:

- Interruption based counting:
  - If you're new to interruptions, you can visit this link and discover them in details: https://create.arduino.cc/projecthub/rafitc/interrupts-basics-f475d5
  - This counting method could be used for `encoders`, since usually, that's where such detection speed (the one offered by using an interruption) would be required.
  - we can **begin** the interruption based counting through the call of one of these:
    * `void beginCountingInterrupter(uint8_t irq_pin, void (*_ISR_callback)(void));`
    * `void beginCountingInterrupter(uint8_t irq_pin, void (*_ISR_callback)(void), uint8_t dir_);`
    * `void beginCountingInterrupter(uint8_t irq_pin, void (*_ISR_callback)(void), uint8_t dir_, uint8_t trigger_on);`
  - Where each non-provided option falls to the default ones, the default direction is `ASCENDING (++)`, and default trigger_on is `FALLING`.
  - A call to this begin function would look like this:

      - `my_counting_btn.beginCountingInterrupter(ISR_BTN_PIN, GET_ISR(isr_btn, countingInterruption));`

- Normal events counting:
  - We can **begin** the noraml counting button through the call of one of these:
    - `void begin(uint8_t pin);`
    - `void begin(uint8_t pin, uint8_t off_state);`
    - `void begin(uint8_t pin, uint8_t off_state, uint8_t dir);`
    - `void begin(uint8_t pin, uint8_t off_state, uint8_t dir, uint8_t debounce_t);`
  - Defaults:
    - **off_state**: `NORMAL_UP`
    - **dir**: `ASCENDING`
    - **debounce_t**: `5 [milliseconds]`
  - A call to this begin function would look like this:
    - `my_counting_btn.begin(BTN_PIN, NORMAL_UP, ASCENDING, 25);`

3.2.2 2.2. Available methods

2.2.1. Settings

We have a number of settings possible that we can perform on our counting button.

- Configure a custom function to be called whenever a certain count is reached:
  - `void setupTriggerOnCount(long count, void (*callback)(void));`
  - Example:

```
#define BUTTON_PIN          5
void callback(){
    Serial.println("10 clicks!");
    my_counting_btn.resetCount();
}
void setup(){
    Serial.begin(9600);
    my_counting_btn.begin(BUTTON_PIN, NORMAL_UP, ASCENDING, 25);
    my_counting_btn.setupTriggerOnCount(10, callback);
}
```

- **And we can change the value to be triggered at dynamically through the call to:**
    - `void setTriggerCount(long count);`
- **Configure whether to count UP or DOWN:**
    - `void setDirection(int8_t direction);`
- **Reset the count to 0:**
    - `void resetCount();`
- **Sets the current value of the count(passed in value):**

  - `void setCount(long count);`
- **Configures on which edge the conting happens `profile = {ON_RISING, ON_FALLING, ON_CHANGE}`:**
  - `void setCountingProfile(uint8_t profile);`

2.2.2. Functionalities

- Returns the current count value: `long getCount();`
- Keep the listening for the counting events happening: `void loopCounter();`. In fact, this function must be called in the `loop()` of your Arduino sketch in order to not miss any pressing events.

3.2.3 2.3. Notes

- A button could only be began as one of the two options, either interruption based, or normal, not both (it just won't work).
- The interruption based counting buttons must be wired on hardware-interrupt enabled pins, otherwise it won't work.
- In case you choose to do a NORMAL_DOWN button, make sure to externally pull down the push-button, otherwise, an internal pull-up resistor is used by default to the NORMAL_UP mode.

CHAPTER FOUR

LINKS

- Source code: https://github.com/Rad-hi/MyButton_Arduino
- Author: https://github.com/Rad-hi