Content – Button-Master Documentation

Summary

General

The whole library is used for digital inputs (HIGH or LOW), especially buttons. The whole library is able to read debounced button states, read single, double and long click and do some specific clicks. More about that in the “Functions”-topic.

Every type of button is supported like pulldown, pullup and intern pullup (not available on every Arduino board). Furthermore, the user can manually choose between inverted (normally used for pullup button) and non-inverted (normally used for pulldown button) mode.

Every functions works without delay() and won’t interrupt your loop, when functions are called. They can slow down your code with a minimum amount of time. Doing some measurement showed that the maximum required time is about 30 microsseconds. All time-depending codes are working with the function millis() and are based on the timer1.

Besides there are two functionalities: real-time and update.
Real-time works like the standard digitalRead(). The button events are processed by the time the function is called and returns a value.
If you use update functions, you always process all events in one function and then get the values which are saved during that one function.
More about that in the “Functionalities”-topic.

Debounce Class

Button debounce is done in this class and their outcomes can be used for different activities. This class can be used alone by “#include <Debounce.h>” into your code.
Functions to read/get current state, risen or fallen edge and changed state are available.

ButtonEvent Class

This class uses the Debounce class by inheritance. To use the functions of this class just include it into your code with “#include <ButtonEvent.h>” without including Debounce.h.

Functions

Debounce

- **Debounce(*button pin*, *debounce time*)**:
  Constructor 1
  Initialize a Debounce object with the button pin and debounce time. Pin mode won't be set, so it must be defined in your setup. Then all library functions (Debounce) can be used.
- **Debounce(*button pin*, *debounce time*, *button mode*, *invert mode*)**:
  Constructor 2
  Initialize a Debounce object with button pin and debounce time. Furthermore, the button and invert mode will be set. Library functions (Debounce) can be used without any further setups.
- **setDebounce(*debounce time*)**:
  Function (void)
  Set the debounce time to a new one. If debounce time is equal to 0 the functions works like standard digitalRead(…).
- **readState()**:
  Returns 0/1 (bool)
  Reads and returns the current state of the button.
- **readRise()**:
  Returns 0/1 (bool)
  Reads the button state and returns 1 if the state rose.
- **readFall()**:
  Returns 0/1 (bool)
  Reads the button state and returns 1 if the state fell.
- **readChange()**:
  Returns 0/1 (bool)
  Reads a change of the button state (HIGH to LOW, LOW to HIGH) and returns 1 if it changed.
- **updateBounce()**:
  Function (void)
  All get…()-functions (Debounce) are based on the update-function. It reads all changes of the button state and safes it in variables. These variables can be read with the get…()-functions.
- **getState()**:
  Returns 0/1 (bool)
  Returns the state of the button.
- **getRise()**:
  Returns 0/1 (bool)
  Returns 1 if the state rose.
- **getFall()**:
  Returns 0/1 (bool)
  Returns 1 if the state fell.
- **getChange()**:
  Returns 0/1 (bool)
  Returns 1 if the state changed.

ButtonEvent

- **Button(*button pin*, *debounce time*):**
  Constructor 1
  Initialize a ButtonEvent object with the button pin and debounce time. Pin mode won't be set, so it must be defined in your setup. Then all library functions (Debounce, ButtonEvent) can be used.
- **Button(*button pin*, *debounce time*, *button mode*, *invert mode*):**
  Constructor 2
  Initialize a ButtonEvent object with button pin and debounce time. Furthermore, the button and invert mode will be set. Library functions (Debounce, ButtonEvent) can be used without any further setups.
- **setParaStandard():**
  Function (void)
  Sets all timer to standard.
  (Further info on the time diagrams: …/ButtonEvent/schematics/Time Diagram – X)
- **setParaClick(*single click time*, *min between time*, *max between time*, *long click time*):**
  Function (void)
  Sets click timer. So that there won't be problems in the code, there are some requirements: The single click and min between time must be bigger than the debounce time. The max. between time must be bigger than the min between time and the long click must be longer than the single click time
  (Further info on the click time diagrams: …/ButtonEvent/schematics/Time Diagram – X Click)
- **setParaAdd(*change add time*, *slow add time*, *fast add time*):**
  Function (void)
  Sets add timer. There are also some requirements like the change add must be bigger than the slow add time (slow add must run once). Furthermore, the slow and fast add time must be positive. It would be possible to make it add fast, then change to slow.
  (Further info on the click time diagrams: …/ButtonEvent/schematics/Time Diagram – Add Click)
- **readSingle():**
  Returns 0/1 (bool)
  Reads a single click and returns 1 if one is done.
  (Further info: …/ButtonEvent/schematics/Time Diagram – Single Click)
- **readDouble():**
  Returns 0/1 (bool)
  Reads a double click and returns 1 if one is done.
  (Further info: …/ButtonEvent/schematics/Time Diagram – Double Click)
- **readLong():**
  Returns 0/1 (bool)
  Reads a long click and returns 1 if one is done.
  (Further info: …/ButtonEvent/schematics/Time Diagram – Long Click)
- **readAction():**
  Returns 0/1/2/3 (char)
  Reads all click events and returns a value. This values can be compared with constants.

- **readAdd():**
  Returns 0/1 (bool)
  Reads a add click and returns 1 if one is done.
  (Further info: …/ButtonEvent/schematics/Time Diagram – Add Click)
- **readAddCalc(*value*, *increment*):**
  Function (void), changes value
  Increments the value (with reference) with the committed increment value if a add click is done (to decrement use a negative value e.g. -5).
- **readAddCalc(*value*):**
  Function (void), changes value
  Increments the value (with reference) by +1 if a add click is done.
- **updateButton():**
  Function (void)
  All get…()-functions (ButtonEvent) are based on the update-function. It reads all changes of the button state and safes it in variables. These variables can be read with the get…()-functions.
- **getSingle():**
  Returns 0/1 (bool)
  Returns 1 if a single click was done.
  (Further info: …/ButtonEvent/schematics/Time Diagram – Single Click)
- **getDouble():**
  Returns 0/1 (bool)
  Returns 1 if a double click was done.
  (Further info: …/ButtonEvent/schematics/Time Diagram – Double Click)
- **getLong():**
  Returns 0/1 (bool)
  Returns 1 if a long click was done.
  (Further info: …/ButtonEvent/schematics/Time Diagram – Long Click)
- **getAction():**
  Returns 0/1/2/3 (char)
  Returns a value depending on which click was done. This values can be compared with constants.
- **getAdd():**
  Returns 0/1 (bool)
  Returns 1 if a add click was done.
  (Further info: …/ButtonEvent/schematics/Time Diagram – Add Click)
- **getAddCalc(*value*, *increment*):**
  Function (void), changes value
  Increments the value (with reference) with the committed increment value if a add click was done (to decrement use a negative value e.g. -5).
- **getAddCalc(*value*):**
  Function (void), changes value
  Increments the value (with reference) by +1 if a add click was done.

Modes and Constants

- Real-time mode/functionality:
  The state and events get returned once, as the functions get called. Using the same functions in one cycle is not recommended and will generate problems.
- Update mode/functionality:
  Once the update-function get called the state and all events get saved in variables. These can be called as often as required. When the update-function get called again all states get updated.
- Button mode:
  Sets the button as input and can be initialized through this constants:
  - **NoMode**: Button mode is undefined
  - **Pulldown**: Button is pulldown button.
  - **Pullup**: Button is pullup button.
  - **InternPullup**: Button is pullup button and works without extra pullup resistors on extern circuit (not available on every Arduino chip).

  (Electrical circuits can be found at ../ButtonEvent/schematics/ Electrical Circuit – X)
- Invert Mode:
  Toggles HIGH and LOW state of button. Mostly used on pullup buttons and can be initialized through this constants:
  - **NoInvert**: Button state won't be inverted automatically (pullup button pressed = HIGH, pulldown button pressed = LOW).
  - **Invert**: Button state will be inverted automatically (pullup button pressed = LOW, pulldown button pressed = HIGH).
- Compare constants:
  For read/getAction() comparison:
  - **SingleClick**
  - **DoubleClick**
  - **LongClick**

Hard- & Software used

Software

- Windows 10
- Arduino IDE* 1.6.8
- Sublime Text 3 (with Stino add-on)
- Fritzing 0.9.2b
- Excel & Word 2016
- Paint

Hardware

- Arduino board(s) (Uno, Mega, etc.) (with USB cable)*
- Breadboard*
- Jumper wires*
- Button(s)*
- Pulldown resistor(s) ($10\text{k}\Omega$)*

*Required for Library-User

Side Facts

How to Install

Follow this link: https://www.arduino.cc/en/Guide/Libraries

Project

- Version: v1.0.2-014
- Tested on*:
  - Arduino Uno Board
  - Arduino Mega Board
- Next Features:
  - Example to calibrate times
  - Define usage in header
  - Add optional serial monitor warnings, infos

** all examples worked correctly*

The Originator

- By Christoph Amon
- Born August 1999
- From Austria/Lower Austria
- Student of HTL 3 Rennweg, Wien/Vienna
- Side project since January 2016

Contact

For further information:

Mail: christoph.amon.htl@gmail.com