SWITCHBUTTON

1.0.0

Generated by Doxygen 1.10.0

Chapter 1

SWITCHBUTTON

An Arduino Uno/Nano library to use momentary switch buttons with debouncing, short and long press detection, supports sleep modes and works without blocking or delay()

Figure 1.1 Switch buttons

Examples how to use the library

- examples/simple/simple.ino
- examples/buttonStates/buttonStates.ino
- examples/sleepmode/sleepmode.ino

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 SWITCHBUTTON Class Reference

```
#include <SWITCHBUTTON.h>
```

Public Types

- enum buttonLogicalStates {
  IDLE , SHORTPRESSED , LONGPRESSED , LONGPRESSEDRELEASED ,
  INPROGRESS , MISSED , DEBOUNCING }
- enum buttonPhysicalStates { RELEASED , PRESSED , UNKNOWN }

Public Member Functions

- SWITCHBUTTON (byte sw_pin, bool inputPulledUp=true)

  Constructor of a the momentary switch object.
- byte checkButton ()

  Returns the current logical state from stored button state.
- byte getButton ()

  Get current logical state for momentary switch.
- byte getState ()

  Get last stored physical button state for the momentary switch.
- bool readyForSleep ()

  Checks, if it save to go to sleep deeper than SLEEP_MODE_IDLE.
- void setDebounceTimeMS (unsigned int debounceTimeMS)

  Change debounce time for a rising or falling signal edge.
- void setState (byte state)

  Store physical button state for the momentary switch.

4.1.1 Detailed Description

Class for a momentary switch button.

Definition at line 34 of file SWITCHBUTTON.h.

4.1.2 Member Enumeration Documentation

4.1.2.1 buttonLogicalStates

```
enum SWITCHBUTTON::buttonLogicalStates
```

Button logical states

Enumerator

IDLE	Button is not pressed and is idle
SHORTPRESSED	Button was short pressed
LONGPRESSED	Button was long pressed
LONGPRESSEDRELEASED	Button was released after a long press
INPROGRESS	Button press is in progress
MISSED	Incomplete long press was detected
DEBOUNCING	Button is blocked for debouncing

Definition at line 37 of file SWITCHBUTTON.h.

```
    {
      IDLE,
      SHORTPRESSED,
      LONGPRESSED,
      LONGPRESSEDRELEASED,
      INPROGRESS,
      MISSED,
      DEBOUNCING
    };
```

4.1.2.2 buttonPhysicalStates

`enum SWITCHBUTTON::buttonPhysicalStates`

Button physical states

Enumerator

RELEASED	Button not pressed
PRESSED	Button pressed
UNKNOWN	Button never checked or set

Definition at line 48 of file SWITCHBUTTON.h.

```
    {
      RELEASED,
      PRESSED,
      UNKNOWN
    };
```

4.1.3 Constructor & Destructor Documentation

4.1.3.1 SWITCHBUTTON()

```
SWITCHBUTTON::SWITCHBUTTON (
              byte sw_pin,
              bool inputPulledUp = true )  [inline]
```

Constructor of a the momentary switch object.

Parameters

in	*sw_pin*	Digital input pin connected to the momentary switch
in	*inputPulledUp*	Set to true (=default), if momentary switch is pulled up by a resistor or pinMode(,INPUT_PULLUP) otherwise set parameter to false

Definition at line 61 of file SWITCHBUTTON.h.

```
{
  m_sw_pin = sw_pin; // Digital input pin for the button
  m_inputPulledUp = inputPulledUp; // Has button a pullup resistor (LOW=pressed,HIGH=released)?
  m_debounceTimeMS = DEBOUNCEMS;
  m_state = UNKNOWN;
  m_lastState = UNKNOWN;
  m_lastLongPressedMS = 0;
  m_lastButtonStartMS = 0;
  m_lastButtonChangeMS=0;
  m_waitingRelease = false;
  m_pendingLongPressed = false;
}
```

4.1.4 Member Function Documentation

4.1.4.1 checkButton()

```
byte SWITCHBUTTON::checkButton ( )  [inline]
```

Returns the current logical state from stored button state.

If you do not use interrupts, you have to start setState() and checkButton() or a function using these (for example getButton()) very frequently in your loop to prevent missing button presses

Return values

SWITCHBUTTON::IDLE	Button is not pressed and is idle
SWITCHBUTTON::SHORTPRESSED	Button was short pressed
SWITCHBUTTON::LONGPRESSED	Button was long pressed
SWITCHBUTTON::LONGPRESSEDRELEASED	Button was released after a long pressed
SWITCHBUTTON::INPROGRESS	Button press is in progress
SWITCHBUTTON::MISSED	Incomplete long press was detected
SWITCHBUTTON::DEBOUNCING	Button is blocked for debouncing

Definition at line 89 of file SWITCHBUTTON.h.

```
{
  unsigned long currentMillis = millis();
  // When first check or after debounce time
  if ((m_lastState == UNKNOWN)
    || (currentMillis-m_lastButtonChangeMS > m_debounceTimeMS)) {
    m_lastButtonChangeMS = currentMillis - m_debounceTimeMS - 1; // Prevent overrun
    if ((m_state==RELEASED) && (m_lastState == UNKNOWN)) { // Init
      m_lastState = m_state;
      return IDLE;
    }
    if (m_state != m_lastState) { // Button state has changed
      if (m_state == PRESSED) { // Rising edge
        m_waitingRelease = true;
        m_lastButtonStartMS = currentMillis;
      }
      m_lastButtonChangeMS = currentMillis;
      m_lastState = m_state;
    }
    if (m_state == PRESSED) { // Button is pressed
      if (m_waitingRelease) {
        if ((currentMillis - m_lastButtonStartMS > LONGPRESSEDMS)
          && (currentMillis-m_lastLongPressedMS > LONGPRESSEDDEADTIMEMS)) {
          m_lastButtonStartMS = currentMillis - LONGPRESSEDMS - 1; // Prevent overrun
          m_lastLongPressedMS = currentMillis;
          m_pendingLongPressed = true;
          return LONGPRESSED;
        } else return INPROGRESS;
      } else return INPROGRESS;
    } else { // Button is released
      if (m_waitingRelease) {
        m_waitingRelease = false;
```

```
            if (currentMillis - m_lastButtonStartMS <= LONGPRESSEDMS){
              return SHORTPRESSED;
            } else {
              if (m_pendingLongPressed) {
                m_pendingLongPressed = false;
                return LONGPRESSEDRELEASED;
              }
              return MISSED; // Too long gap between rising edge and button release
            }
          } else return IDLE;
        }
      } else { // In debounce time
        return DEBOUNCING;
      }
    }
```

4.1.4.2 getButton()

`byte SWITCHBUTTON::getButton ( ) [inline]`

Get current logical state for momentary switch.

Reads physical button state with DigitalRead() and checks current logical state by calling checkButton()

Return values

SWITCHBUTTON::IDLE	Button is not pressed and is idle
SWITCHBUTTON::SHORTPRESSED	Button was short pressed
SWITCHBUTTON::LONGPRESSED	Button was long pressed
SWITCHBUTTON::LONGPRESSEDRELEASED	Button was released after a long pressed
SWITCHBUTTON::INPROGRESS	Button press is in progress
SWITCHBUTTON::MISSED	Incomplete long press was detected
SWITCHBUTTON::DEBOUNCING	Button is blocked for debouncing

Definition at line 150 of file SWITCHBUTTON.h.

```
    {
      if (m_inputPulledUp) {
        setState((digitalRead(m_sw_pin)==LOW) ? PRESSED:RELEASED);
      } else {
        setState((digitalRead(m_sw_pin)==HIGH) ? PRESSED:RELEASED);
      }
      return checkButton();
    }
```

4.1.4.3 getState()

`byte SWITCHBUTTON::getState ( ) [inline]`

Get last stored physical button state for the momentary switch.

Return values

SWITCHBUTTON::IDLE	Button was not pressed and is idle
SWITCHBUTTON::PRESSED	Button was pressed
SWITCHBUTTON::UNKNOWN	Button was never checked or set

Definition at line 167 of file SWITCHBUTTON.h.

```
{
  return m_state;
}
```

4.1.4.4 readyForSleep()

```
bool SWITCHBUTTON::readyForSleep ( )  [inline]
```

Checks, if it save to go to sleep deeper than SLEEP_MODE_IDLE.

Returns true, if device has no pending button press. Sleep mode SLEEP_MODE_IDLE is always possible but deeper sleep modes, for example SLEEP_MODE_PWR_SAVE, are only save after readyForSleep() returns true

Return values

true	Yes, it is save to go to sleep deeper then SLEEP_MODE_IDLE
false	No, only SLEEP_MODE_IDLE is possible

Definition at line 183 of file SWITCHBUTTON.h.

```
{
  return (checkButton() == IDLE);
}
```

4.1.4.5 setDebounceTimeMS()

```
void SWITCHBUTTON::setDebounceTimeMS (
            unsigned int debounceTimeMS )  [inline]
```

Change debounce time for a rising or falling signal edge.

Parameters

in	*debounceTimeMS*	Debounce time in milliseconds

Definition at line 193 of file SWITCHBUTTON.h.

```
{
  m_debounceTimeMS = debounceTimeMS;
}
```

4.1.4.6 setState()

```
void SWITCHBUTTON::setState (
            byte state )  [inline]
```

Store physical button state for the momentary switch.

Parameters

in	*state*	Button state: SWITCHBUTTON::PRESSED(=pressed) or SWITCHBUTTON::RELEASED(=released).

Definition at line 203 of file SWITCHBUTTON.h.

```
00204     {
00205       if (state == PRESSED ) m_state = PRESSED;
00206       if (state == RELEASED ) m_state = RELEASED;
00207     }
```

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

- SWITCHBUTTON.h

Chapter 5

File Documentation

5.1 buttonStates.ino

```
/*
 * Example to get all button states
 */

#include <SWITCHBUTTON.h>

// Button pin
#define SW_PIN 6
// Button object
SWITCHBUTTON switchButton(SW_PIN);

void setup() {
  Serial.begin(9600);
  // If you have no external pullup resistor, set pin to INPUT_PULLUP
  pinMode(SW_PIN,INPUT_PULLUP);
}

void loop() {
  static byte lastButtonState = SWITCHBUTTON::UNKNOWN;
  byte buttonState;

  // Start getButton() frequently in your loop to avoid missing button presses
  buttonState = switchButton.getButton();
  if (buttonState != lastButtonState) {
    lastButtonState = buttonState;
    switch (buttonState) {
      case SWITCHBUTTON::SHORTPRESSED:
        Serial.println("Pressed");
        break;
      case SWITCHBUTTON::LONGPRESSED:
        Serial.println("Long pressed");
        break;
      case SWITCHBUTTON::MISSED:
        Serial.println("Missed");
        break;
      case SWITCHBUTTON::INPROGRESS:
        Serial.println("In progress");
        break;
      case SWITCHBUTTON::IDLE:
        Serial.println("Idle");
        break;
      case SWITCHBUTTON::DEBOUNCING:
        Serial.println("Debouncing");
        break;
      case SWITCHBUTTON::LONGPRESSEDRELEASED:
        Serial.println("Long pressed released");
        break;
    }
  }
}
```

5.2 simple.ino

```
/*
```

```
 * Example to get the final button press state
 */

#include <SWITCHBUTTON.h>

// Button pin
#define SW_PIN 6
// Button object
SWITCHBUTTON switchButton(SW_PIN);

void setup() {
  Serial.begin(9600);
  // If you have no external pullup resistor, set pin to INPUT_PULLUP
  pinMode(SW_PIN,INPUT_PULLUP);
}

void loop() {
  // Start getButton() frequently in your loop to avoid missing button presses
  switch (switchButton.getButton()) {
    case SWITCHBUTTON::SHORTPRESSED:
      Serial.println("Pressed");
      break;
    case SWITCHBUTTON::LONGPRESSED:
      Serial.println("Long pressed");
      break;
  }
}
```

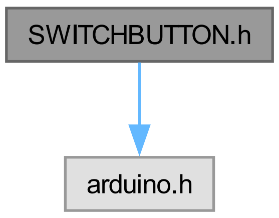
5.3 sleepmode.ino

```
/*
 * Example to get the final button press state while using sleep modes
 */

#include <avr/sleep.h>
#include <SWITCHBUTTON.h>

// Button pin
#define SW_PIN 6
SWITCHBUTTON switchButton(SW_PIN);

// Enable pin change interrupt
void pciSetup(byte pin) {
  *digitalPinToPCMSK(pin) |= bit (digitalPinToPCMSKbit(pin)); // enable pin
  PCIFR  |= bit (digitalPinToPCICRbit(pin)); // clear any outstanding interrupt
  PCICR  |= bit (digitalPinToPCICRbit(pin)); // enable interrupt for the group
}

// ISR to handle pin change interrupt for D0 to D7 here
ISR (PCINT2_vect) {
  // Empty, only as a wakeup trigger
}

void setup() {
  Serial.begin(9600);

  // If you have no external pullup resistor for the button, set pin to INPUT_PULLUP
  pinMode(SW_PIN,INPUT_PULLUP);

  // Set pin change interrupt for momentary switch
  pciSetup(SW_PIN);
}

void loop() {
  // Default sleep mode (SWITCHBUTTON library allows everytime SLEEP_MODE_IDLE)
  byte selectedSleepMode = SLEEP_MODE_IDLE;

  // Start getButton() frequently in your loop to avoid missing button presses
  switch (switchButton.getButton()) {
    case SWITCHBUTTON::SHORTPRESSED:
      Serial.println("Short pressed");
      break;
    case SWITCHBUTTON::LONGPRESSED:
      Serial.println("Long pressed");
      break;
    case SWITCHBUTTON::IDLE:
      // Set deeper sleep mode, because now we need no millis()&Co
      selectedSleepMode = SLEEP_MODE_PWR_DOWN;
      break;
  }
  Serial.flush();
  set_sleep_mode(selectedSleepMode);
  sleep_mode();
}
```

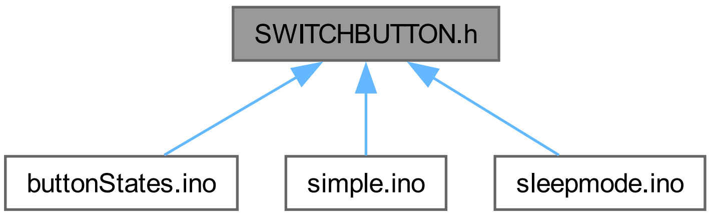
5.4 SWITCHBUTTON.h File Reference

```
#include <arduino.h>
```

Include dependency graph for SWITCHBUTTON.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SWITCHBUTTON

Macros

- #define SWITCHBUTTON_VERSION "1.0.0"
- #define LONGPRESSEDMS 1000
- #define DEBOUNCEMS 100
- #define LONGPRESSEDDEADTIMEMS 500

5.4.1 Detailed Description

Class: SWITCHBUTTON

Arduino Library for a momentary switch button.

This library

- detects and differentiates between short and long button presses
- supports debouncing
- works nonblocking and without delay()
- supports sleep modes (see example "pinChangeInterruptPowerSave")

Home: `https://github.com/codingABI/SWITCHBUTTON`

Author

codingABI `https://github.com/codingABI/`

Copyright

CC0

Version

1.0.0

Definition in file SWITCHBUTTON.h.

5.4.2 Macro Definition Documentation

5.4.2.1 DEBOUNCEMS

```
#define DEBOUNCEMS 100
```

Default debounce time in milliseconds after a rising or falling signal edge

Definition at line 29 of file SWITCHBUTTON.h.

5.4.2.2 LONGPRESSEDDEADTIMEMS

```
#define LONGPRESSEDDEADTIMEMS 500
```

Splits a sustained long press in individuals long presses after these milliseconds

Definition at line 31 of file SWITCHBUTTON.h.

5.4.2.3 LONGPRESSEDMS

```
#define LONGPRESSEDMS 1000
```

Time duration in milliseconds to detect a button long press

Definition at line 27 of file SWITCHBUTTON.h.

5.4.2.4 SWITCHBUTTON_VERSION

```
#define SWITCHBUTTON_VERSION "1.0.0"
```

Library version

Definition at line 22 of file SWITCHBUTTON.h.

5.5 SWITCHBUTTON.h

Go to the documentation of this file.

```
00001
00019 #pragma once
00020
00022 #define SWITCHBUTTON_VERSION "1.0.0"
00023
00024 #include <arduino.h>
00025
00027 #define LONGPRESSEDMS 1000
00029 #define DEBOUNCEMS 100
00031 #define LONGPRESSEDDEADTIMEMS 500
00032
00034 class SWITCHBUTTON {
00035   public:
00037     enum buttonLogicalStates
00038     {
00039       IDLE,
00040       SHORTPRESSED,
00041       LONGPRESSED,
00042       LONGPRESSEDRELEASED,
00043       INPROGRESS,
00044       MISSED,
00045       DEBOUNCING
00046     };
00048     enum buttonPhysicalStates
00049     {
00050       RELEASED,
00051       PRESSED,
00052       UNKNOWN
00053     };
00054
00061     SWITCHBUTTON(byte sw_pin, bool inputPulledUp=true)
00062     {
00063       m_sw_pin = sw_pin; // Digital input pin for the button
00064       m_inputPulledUp = inputPulledUp; // Has button a pullup resistor (LOW=pressed,HIGH=released)?
00065       m_debounceTimeMS = DEBOUNCEMS;
00066       m_state = UNKNOWN;
00067       m_lastState = UNKNOWN;
00068       m_lastLongPressedMS = 0;
00069       m_lastButtonStartMS = 0;
00070       m_lastButtonChangeMS=0;
00071       m_waitingRelease = false;
00072       m_pendingLongPressed = false;
00073     }
00074
00089     byte checkButton()
00090     {
00091       unsigned long currentMillis = millis();
00092       // When first check or after debounce time
00093       if ((m_lastState == UNKNOWN)
00094         || (currentMillis-m_lastButtonChangeMS > m_debounceTimeMS)) {
00095         m_lastButtonChangeMS = currentMillis - m_debounceTimeMS - 1; // Prevent overrun
00096         if ((m_state==RELEASED) && (m_lastState == UNKNOWN)) { // Init
```

```
          m_lastState = m_state;
          return IDLE;
        }
        if (m_state != m_lastState) { // Button state has changed
          if (m_state == PRESSED) { // Rising edge
            m_waitingRelease = true;
            m_lastButtonStartMS = currentMillis;
          }
          m_lastButtonChangeMS = currentMillis;
          m_lastState = m_state;
        }
        if (m_state == PRESSED) { // Button is pressed
          if (m_waitingRelease) {
            if ((currentMillis - m_lastButtonStartMS > LONGPRESSEDMS)
              && (currentMillis-m_lastLongPressedMS > LONGPRESSEDDEADTIMEMS)) {
              m_lastButtonStartMS = currentMillis - LONGPRESSEDMS - 1; // Prevent overrun
              m_lastLongPressedMS = currentMillis;
              m_pendingLongPressed = true;
              return LONGPRESSED;
            } else return INPROGRESS;
          } else return INPROGRESS;
        } else { // Button is released
          if (m_waitingRelease) {
            m_waitingRelease = false;
            if (currentMillis - m_lastButtonStartMS <= LONGPRESSEDMS){
              return SHORTPRESSED;
            } else {
              if (m_pendingLongPressed) {
                m_pendingLongPressed = false;
                return LONGPRESSEDRELEASED;
              }
              return MISSED; // Too long gap between rising edge and button release
            }
          } else return IDLE;
        }
      } else { // In debounce time
        return DEBOUNCING;
      }
    }

    byte getButton()
    {
      if (m_inputPulledUp) {
        setState((digitalRead(m_sw_pin)==LOW) ? PRESSED:RELEASED);
      } else {
        setState((digitalRead(m_sw_pin)==HIGH) ? PRESSED:RELEASED);
      }
      return checkButton();
    }

    byte getState()
    {
      return m_state;
    }

    bool readyForSleep()
    {
      return (checkButton() == IDLE);
    }

    void setDebounceTimeMS(unsigned int debounceTimeMS)
    {
      m_debounceTimeMS = debounceTimeMS;
    }

    void setState(byte state)
    {
      if (state == PRESSED ) m_state = PRESSED;
      if (state == RELEASED ) m_state = RELEASED;
    }
  private:
    byte m_sw_pin;
    bool m_inputPulledUp;
    unsigned int m_debounceTimeMS;
    unsigned long m_lastLongPressedMS;
    unsigned long m_lastButtonStartMS;
    unsigned long m_lastButtonChangeMS;
    byte m_state;
    byte m_lastState;
    bool m_waitingRelease;
    bool m_pendingLongPressed;
};
```

Index