

SWITCHBUTTON

1.0.0

Generated by Doxygen 1.10.0

1 SWITCHBUTTON	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 SWITCHBUTTON Class Reference	7
4.1.1 Detailed Description	7
4.1.2 Member Enumeration Documentation	8
4.1.2.1 buttonLogicalStates	8
4.1.2.2 buttonPhysicalStates	9
4.1.3 Constructor & Destructor Documentation	9
4.1.3.1 SWITCHBUTTON()	9
4.1.4 Member Function Documentation	10
4.1.4.1 checkButton()	10
4.1.4.2 getButton()	11
4.1.4.3 getState()	11
4.1.4.4 readyForSleep()	12
4.1.4.5 setDebounceTimeMS()	12
4.1.4.6 setState()	12
5 File Documentation	15
5.1 buttonStates.ino	15
5.2 simple.ino	15
5.3 sleepmode.ino	16
5.4 SWITCHBUTTON.h File Reference	17
5.4.1 Detailed Description	18
5.4.2 Macro Definition Documentation	18
5.4.2.1 DEBOUNCEMS	18
5.4.2.2 LONGPRESSEDDEADTIMEMS	18
5.4.2.3 LONGPRESSEDMS	19
5.4.2.4 SWITCHBUTTON_VERSION	19
5.5 SWITCHBUTTON.h	19
Index	21

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:

SWITCHBUTTON	7
--	-------------------

Chapter 3

File Index

3.1 File List

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

buttonStates.ino	15
simple.ino	15
sleepmode.ino	16
SWITCHBUTTON.h	17

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](#).

```

00038     {
00039         IDLE,
00040         SHORTPRESSED,
00041         LONGPRESSED,
00042         LONGPRESSEDRELEASED,
00043         INPROGRESS,
00044         MISSED,
00045         DEBOUNCING
00046     };

```

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](#).

```

00049     {
00050         RELEASED,
00051         PRESSED,
00052         UNKNOWN
00053     };

```

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

Definition at line 61 of file SWITCHBUTTON.h.

```

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     }

```

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.

```

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
00097                 m_lastState = m_state;
00098                 return IDLE;
00099             }
00100             if (m_state != m_lastState) { // Button state has changed
00101                 if (m_state == PRESSED) { // Rising edge
00102                     m_waitingRelease = true;
00103                     m_lastButtonStartMS = currentMillis;
00104                 }
00105                 m_lastButtonChangeMS = currentMillis;
00106                 m_lastState = m_state;
00107             }
00108             if (m_state == PRESSED) { // Button is pressed
00109                 if (m_waitingRelease) {
00110                     if ((currentMillis - m_lastButtonStartMS > LONGPRESSEDMS)
00111                         && (currentMillis-m_lastLongPressedMS > LONGPRESSEDDEADTIMES)) {
00112                         m_lastButtonStartMS = currentMillis - LONGPRESSEDMS - 1; // Prevent overrun
00113                         m_lastLongPressedMS = currentMillis;
00114                         m_pendingLongPressed = true;
00115                         return LONGPRESSED;
00116                     } else return INPROGRESS;
00117                 } else return INPROGRESS;
00118             } else { // Button is released
00119                 if (m_waitingRelease) {
00120                     m_waitingRelease = false;

```

```

00121         if (currentMillis - m_lastButtonStartMS <= LONGPRESSEDMS){
00122             return SHORTPRESSED;
00123         } else {
00124             if (m_pendingLongPressed) {
00125                 m_pendingLongPressed = false;
00126                 return LONGPRESSEDRELEASED;
00127             }
00128             return MISSED; // Too long gap between rising edge and button release
00129         }
00130     } else return IDLE;
00131 }
00132 } else { // In debounce time
00133     return DEBOUNCING;
00134 }
00135 }

```

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](#).

```

00151 {
00152     if (m_inputPulledUp) {
00153         setState((digitalRead(m_sw_pin)==LOW) ? PRESSED:RELEASED);
00154     } else {
00155         setState((digitalRead(m_sw_pin)==HIGH) ? PRESSED:RELEASED);
00156     }
00157     return checkButton();
00158 }

```

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](#).

```

00168     {
00169         return m_state;
00170     }

```

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

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

Definition at line 183 of file [SWITCHBUTTON.h](#).

```

00184     {
00185         return (checkButton() == IDLE);
00186     }

```

4.1.4.5 setDebounceTimeMS()

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

Change debounce time for a rising or falling signal edge.

Parameters

in	<i>debounceTimeMS</i>	Debounce time in milliseconds
----	-----------------------	-------------------------------

Definition at line 193 of file [SWITCHBUTTON.h](#).

```

00194     {
00195         m_debounceTimeMS = debounceTimeMS;
00196     }

```

4.1.4.6 setState()

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

Store physical button state for the momentary switch.

Parameters

in	<i>state</i>	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

```
00001 /*
00002  * Example to get all button states
00003  */
00004
00005 #include <SWITCHBUTTON.h>
00006
00007 // Button pin
00008 #define SW_PIN 6
00009 // Button object
00010 SWITCHBUTTON switchButton(SW_PIN);
00011
00012 void setup() {
00013     Serial.begin(9600);
00014     // If you have no external pullup resistor, set pin to INPUT_PULLUP
00015     pinMode(SW_PIN, INPUT_PULLUP);
00016 }
00017
00018 void loop() {
00019     static byte lastButtonState = SWITCHBUTTON::UNKNOWN;
00020     byte buttonState;
00021
00022     // Start getButton() frequently in your loop to avoid missing button presses
00023     buttonState = switchButton.getButton();
00024     if (buttonState != lastButtonState) {
00025         lastButtonState = buttonState;
00026         switch (buttonState) {
00027             case SWITCHBUTTON::SHORTPRESSED:
00028                 Serial.println("Pressed");
00029                 break;
00030             case SWITCHBUTTON::LONGPRESSED:
00031                 Serial.println("Long pressed");
00032                 break;
00033             case SWITCHBUTTON::MISSED:
00034                 Serial.println("Missed");
00035                 break;
00036             case SWITCHBUTTON::INPROGRESS:
00037                 Serial.println("In progress");
00038                 break;
00039             case SWITCHBUTTON::IDLE:
00040                 Serial.println("Idle");
00041                 break;
00042             case SWITCHBUTTON::DEBOUNCING:
00043                 Serial.println("Debouncing");
00044                 break;
00045             case SWITCHBUTTON::LONGPRESSEDRELEASED:
00046                 Serial.println("Long pressed released");
00047                 break;
00048         }
00049     }
00050 }
```

5.2 simple.ino

```
00001 /*
```

```

00002  * Example to get the final button press state
00003  */
00004
00005  #include <SWITCHBUTTON.h>
00006
00007  // Button pin
00008  #define SW_PIN 6
00009  // Button object
00010  SWITCHBUTTON switchButton(SW_PIN);
00011
00012  void setup() {
00013      Serial.begin(9600);
00014      // If you have no external pullup resistor, set pin to INPUT_PULLUP
00015      pinMode(SW_PIN, INPUT_PULLUP);
00016  }
00017
00018  void loop() {
00019      // Start getButton() frequently in your loop to avoid missing button presses
00020      switch (switchButton.getButton()) {
00021          case SWITCHBUTTON::SHORTPRESSED:
00022              Serial.println("Pressed");
00023              break;
00024          case SWITCHBUTTON::LONGPRESSED:
00025              Serial.println("Long pressed");
00026              break;
00027      }
00028  }

```

5.3 sleepmode.ino

```

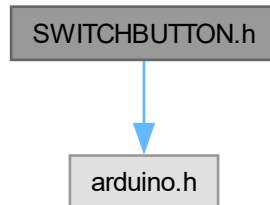
00001  /*
00002  * Example to get the final button press state while using sleep modes
00003  */
00004
00005  #include <avr/sleep.h>
00006  #include <SWITCHBUTTON.h>
00007
00008  // Button pin
00009  #define SW_PIN 6
00010  SWITCHBUTTON switchButton(SW_PIN);
00011
00012  // Enable pin change interrupt
00013  void pciSetup(byte pin) {
00014      *digitalPinToPCMSK(pin) |= bit (digitalPinToPCMSKbit(pin)); // enable pin
00015      PCIFR |= bit (digitalPinToPCICRbit(pin)); // clear any outstanding interrupt
00016      PCICR |= bit (digitalPinToPCICRbit(pin)); // enable interrupt for the group
00017  }
00018
00019  // ISR to handle pin change interrupt for D0 to D7 here
00020  ISR (PCINT2_vect) {
00021      // Empty, only as a wakeup trigger
00022  }
00023
00024  void setup() {
00025      Serial.begin(9600);
00026
00027      // If you have no external pullup resistor for the button, set pin to INPUT_PULLUP
00028      pinMode(SW_PIN, INPUT_PULLUP);
00029
00030      // Set pin change interrupt for momentary switch
00031      pciSetup(SW_PIN);
00032  }
00033
00034  void loop() {
00035      // Default sleep mode (SWITCHBUTTON library allows everytime SLEEP_MODE_IDLE)
00036      byte selectedSleepMode = SLEEP_MODE_IDLE;
00037
00038      // Start getButton() frequently in your loop to avoid missing button presses
00039      switch (switchButton.getButton()) {
00040          case SWITCHBUTTON::SHORTPRESSED:
00041              Serial.println("Short pressed");
00042              break;
00043          case SWITCHBUTTON::LONGPRESSED:
00044              Serial.println("Long pressed");
00045              break;
00046          case SWITCHBUTTON::IDLE:
00047              // Set deeper sleep mode, because now we need no millis()&Co
00048              selectedSleepMode = SLEEP_MODE_PWR_DOWN;
00049              break;
00050      }
00051      Serial.flush();
00052      set_sleep_mode(selectedSleepMode);
00053      sleep_mode();
00054  }
00055  }

```

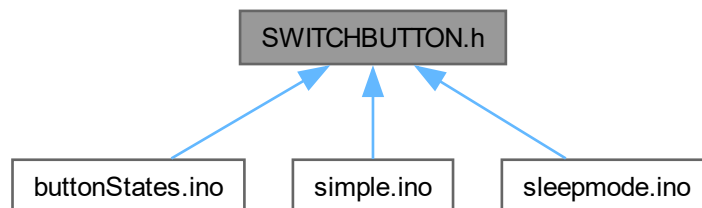
5.4 SWITCHBUTTON.h File Reference

```
#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
```

```

00097         m_lastState = m_state;
00098         return IDLE;
00099     }
00100     if (m_state != m_lastState) { // Button state has changed
00101         if (m_state == PRESSED) { // Rising edge
00102             m_waitingRelease = true;
00103             m_lastButtonStartMS = currentMillis;
00104         }
00105         m_lastButtonChangeMS = currentMillis;
00106         m_lastState = m_state;
00107     }
00108     if (m_state == PRESSED) { // Button is pressed
00109         if (m_waitingRelease) {
00110             if ((currentMillis - m_lastButtonStartMS > LONGPRESSEDMS)
00111                 && (currentMillis - m_lastLongPressedMS > LONGPRESSEDDEADTIMES)) {
00112                 m_lastButtonStartMS = currentMillis - LONGPRESSEDMS - 1; // Prevent overrun
00113                 m_lastLongPressedMS = currentMillis;
00114                 m_pendingLongPressed = true;
00115                 return LONGPRESSED;
00116             } else return INPROGRESS;
00117         } else return INPROGRESS;
00118     } else { // Button is released
00119         if (m_waitingRelease) {
00120             m_waitingRelease = false;
00121             if (currentMillis - m_lastButtonStartMS <= LONGPRESSEDMS) {
00122                 return SHORTPRESSED;
00123             } else {
00124                 if (m_pendingLongPressed) {
00125                     m_pendingLongPressed = false;
00126                     return LONGPRESSEDRELEASED;
00127                 }
00128                 return MISSED; // Too long gap between rising edge and button release
00129             }
00130         } else return IDLE;
00131     }
00132 } else { // In debounce time
00133     return DEBOUNCING;
00134 }
00135 }
00136
00150 byte getButton()
00151 {
00152     if (m_inputPulledUp) {
00153         setState((digitalRead(m_sw_pin) == LOW) ? PRESSED : RELEASED);
00154     } else {
00155         setState((digitalRead(m_sw_pin) == HIGH) ? PRESSED : RELEASED);
00156     }
00157     return checkButton();
00158 }
00159
00167 byte getState()
00168 {
00169     return m_state;
00170 }
00171
00183 bool readyForSleep()
00184 {
00185     return (checkButton() == IDLE);
00186 }
00187
00193 void setDebounceTimeMS(unsigned int debounceTimeMS)
00194 {
00195     m_debounceTimeMS = debounceTimeMS;
00196 }
00197
00203 void setState(byte state)
00204 {
00205     if (state == PRESSED) m_state = PRESSED;
00206     if (state == RELEASED) m_state = RELEASED;
00207 }
00208 private:
00209     byte m_sw_pin;
00210     bool m_inputPulledUp;
00211     unsigned int m_debounceTimeMS;
00212     unsigned long m_lastLongPressedMS;
00213     unsigned long m_lastButtonStartMS;
00214     unsigned long m_lastButtonChangeMS;
00215     byte m_state;
00216     byte m_lastState;
00217     bool m_waitingRelease;
00218     bool m_pendingLongPressed;
00219 };

```


Index

- buttonLogicalStates
 - SWITCHBUTTON, [8](#)
- buttonPhysicalStates
 - SWITCHBUTTON, [9](#)
- buttonStates.ino, [15](#)
- checkButton
 - SWITCHBUTTON, [10](#)
- DEBOUNCES
 - SWITCHBUTTON.h, [18](#)
- DEBOUNCING
 - SWITCHBUTTON, [9](#)
- getButton
 - SWITCHBUTTON, [11](#)
- getState
 - SWITCHBUTTON, [11](#)
- IDLE
 - SWITCHBUTTON, [9](#)
- INPROGRESS
 - SWITCHBUTTON, [9](#)
- LONGPRESSED
 - SWITCHBUTTON, [9](#)
- LONGPRESSEDDEADTIMEMS
 - SWITCHBUTTON.h, [18](#)
- LONGPRESSEDMS
 - SWITCHBUTTON.h, [18](#)
- LONGPRESSEDRELEASED
 - SWITCHBUTTON, [9](#)
- MISSED
 - SWITCHBUTTON, [9](#)
- PRESSED
 - SWITCHBUTTON, [9](#)
- readyForSleep
 - SWITCHBUTTON, [12](#)
- RELEASED
 - SWITCHBUTTON, [9](#)
- setDebounceTimeMS
 - SWITCHBUTTON, [12](#)
- setState
 - SWITCHBUTTON, [12](#)
- SHORTPRESSED
 - SWITCHBUTTON, [9](#)
- simple.ino, [15](#)
- sleepmode.ino, [16](#)
- SWITCHBUTTON, [1](#), [7](#)
 - buttonLogicalStates, [8](#)
 - buttonPhysicalStates, [9](#)
 - checkButton, [10](#)
 - DEBOUNCING, [9](#)
 - getButton, [11](#)
 - getState, [11](#)
 - IDLE, [9](#)
 - INPROGRESS, [9](#)
 - LONGPRESSED, [9](#)
 - LONGPRESSEDRELEASED, [9](#)
 - MISSED, [9](#)
 - PRESSED, [9](#)
 - readyForSleep, [12](#)
 - RELEASED, [9](#)
 - setDebounceTimeMS, [12](#)
 - setState, [12](#)
 - SHORTPRESSED, [9](#)
 - SWITCHBUTTON, [9](#)
 - UNKNOWN, [9](#)
- SWITCHBUTTON.h, [17](#), [19](#)
 - DEBOUNCES, [18](#)
 - LONGPRESSEDDEADTIMEMS, [18](#)
 - LONGPRESSEDMS, [18](#)
 - SWITCHBUTTON_VERSION, [19](#)
- SWITCHBUTTON_VERSION
 - SWITCHBUTTON.h, [19](#)
- UNKNOWN
 - SWITCHBUTTON, [9](#)