Arduino

Switch Library User Guide

A Library Supporting the Reading of Multiple

Mixed-type Simple Switches & Circuits

Warranties & Exceptions

This document and its content are in the public domain and may be used without restriction and without warranty.

Contents

Warranties & Exceptions 2
Introduction 3
Overview 3
Design Objectives 4
Constraints & Limitations 4
Switch Types Supported 5
Common Switch Wiring Schemes 5
Using the <switch_lib> Library 6
Location of the switch_lib Library 6
Steps to Successful Use 7
Specifications 10
Specifications – Switch Control Structure (SCS) 10
Specifications - Reserved Macro Definitions 11
Specifications - Switch Control Functions 12
Example Sketches 15
“switch_lib_example_1” - Turning LED On/Off With a Button Switch 16
“switch_lib_example_2” - Turning LED On/Off With a Toggle Switch 17
“switch_lib_example_3” - Turning Multiple LEDs On/Off With Multiple Button & Toggle Switches 18
“switch_lib_example_4” – Processing More Button & Toggle Switches 21

Introduction

Implementing switches, of any type, can be troublesome as not all switches are equal! Some are ‘fleeting’ or momentary, like button switches, and some are simply either on or off until they are ‘flipped’ at their next actuation. Button switches are fairly standard in their design, but toggle type switches are many varied – simple toggle, slide, tilt, rotary, etc. If you are incorporating switches into your projects then issues such as switch design, transition ‘noise’ and wiring schemes will all come into play at some point in a project’s design.

The good news is that both types of switch can be brought to heel by the <switch_lib> library which provides a simple to use, no frills, software solution for connecting a mix of switch types wired in a variety of circuit schemes. The end result is that, by using the <switch_lib>, the only components required are switches, connecting wires and, *if wished* , 10k ohm pull down resistors. However, even the 10k ohm resistors can be left out by choice of the right circuit (see below, Common Switch Wiring Schemes).

This User Guide (UG) describes the <switch_lib> library for Arduino, detailing the functions and definitions available to the end user for implementing switches of either style and in a choice of wiring schemes - any number of switches of any style and of varying common wiring designs may be configured, the only limitation being the number of digital pins available.

However, before continuing, if you would like to understand a little about the issues associated with switches have a look at the tutorial Understanding & Using Button Switches. Although it is centred on the simple button switch, the basics are also common to toggle switches.

Overview

This UG provides information that will prove helpful in understanding the capabilities of the switch library, switch_lib, in designing and implementing projects using switches, single or multiple of varying types.

The UG gives information and explanations of:

- Design objectives – what was sought.
- Constraints & limitations - it is vitally important to understand any constraints and limitations that the library imposes/suffers, as these may play a part in the way in which you utilise the library.
- Types of switch supported – there are many types of switch available. Those suitable for use with the library are highlighted.
- Common wiring schemes – again there are many ways in which a switch may be wired. The library has been designed to support the two of the most common wiring schemes to be used for any switch type. The approach is to minimise any additional hardware components used.
- Using the library – describes how the library should be (can be) incorporated into your projects.
- Declarations & definitions – provides a list of all of the library’s switch macro definitions and control struct(ure) available to the end user to incorporate into their sketches.

- Function specifications – each of the library's functions is detailed with an example in its use. Any specific points of note are also provided.
- Finally, example sketches are provided, building from single button and toggle switches to multiple switch types using both circuit schemes.

Hopefully, the UG will become a 'one-stop-shop' to help the end user supplement understanding in the application of switches and the library's capabilities.

Design Objectives

At the outset a number of key objectives were established for the design of the switch library, these being a library that provided/supported:

- a simple, logical and straight forward design
- ease of end user project switch configuration, irrespective of type and number of switches or how connected (wired)
- different switch types - the ubiquitous button switch and a variety of different types of toggle switch
- support for common wiring schemes – simply connected with or without a 10k ohm pull down resistor
- mixed switch/circuit implementations – support for a mix of switch types and wiring schemes
- software auto-debounce of noisy switch transitions – removing from end user design consideration issues relating to noisy switch transition by incorporating transparent debounce features
- one switch read function irrespective of switch type or wiring scheme – providing a simple to use function to read any and all switches
- developing a user guide that is informative and such that it is easy to 'dip' into.

Constraints & Limitations

Nothing in this world is perfect and <switch_lib> is far from that. Whilst it does provide a set of useful capabilities to aid and assist Arduino projects involving switches there are several constraints and limitations in its design and use to be aware of:

1. Every switch to be configured requires its own digital pin. Whilst this is not an issue for say, a mega 2560 microcontroller, lesser boards are more constraining in the number of digital pins they support. Certainly for UNO microcontrollers and better, there should not be a practical issue in mapping switches to digital pins for most switch hungry projects.
2. Development of the library was limited to six switches (see switch_lib Example 4) –

   - two x button switches wired <u>with</u> a 10k ohm pull down resistor
   - one x button switches wired <u>without</u> a 10k ohm pull down resistor
   - two x toggle switches wired <u>without</u> a 10k ohm pull down resistor
   - one x toggle switches wired <u>with</u> a 10k ohm pull down resistor

   but, there is no reason to believe that more could not be configured within the limits of the chosen microcontroller.

3. For every switch configured, 10 bytes of free memory will be allocated at run time when the <switch_lib> class is initiated. This memory requirement is in addition to the size of the compiled sketch.
4. The period of time defined for switch noise debounce is global and applicable to all switches, irrespective of type. It is preset ‘out of the box’ (OOTB) to 10 milliseconds but it may be programmatically adjusted by the end user code as required (see function `set_debounce`, below).
5. The library supports two simple and commonly seen switch wiring schemes (see Common Switch Wiring Schemes), these being without the use of any hardware components other than 10k ohm resistors. Even these can be dispensed with! Having said that, the library should also support (but not tested) switches connected with hardware debounce circuits. If this is the case then set the software debounce period to 0 milliseconds (see function `set_debounce`, below).
6. For switches to be responsive in something like real-time, they need to be tested frequently and, for button switches particularly, processed when a switch cycle is detected. However, toggle switches may have their current status examined at any point and any time. A software design based on a switch polling loop should be an ideal harness to ensure continuous switch testing and processing.

Switch Types Supported

There are so, so many switches available, many for specific purposes but most of a general nature and suitable for the majority of needs.

The switch library was developed to support two types of common and general use switches – button, or momentary switches, and toggle switches. Of course this latter type of switch, toggle, itself comes in all kinds of designs, for example, simple single lever, pop-on pop-off, rotary, slide, tilt, etc.

Button / momentary

Slide

Toggle

Tilt

Examples of common types of switch

The principal distinction between button (momentary) and toggle type switches is that button switches have a switch cycle of OFF-ON-OFF which signifies switch activation, whereas toggle switches go through either OFF-ON or ON-OFF representing two switch transitions. The status of toggle switches therefore persists after being physically switched – they stay ON or OFF. Switch_lib automatically handles this feature.

Common Switch Wiring Schemes

If you now appreciate the differences between switch types, it is necessary to understand how they should be connected to the microcontroller.

The two commonly seen switch wiring schemes *without* hardware debounce are shown below, as 'circuit_C1' and 'circuit_C2':

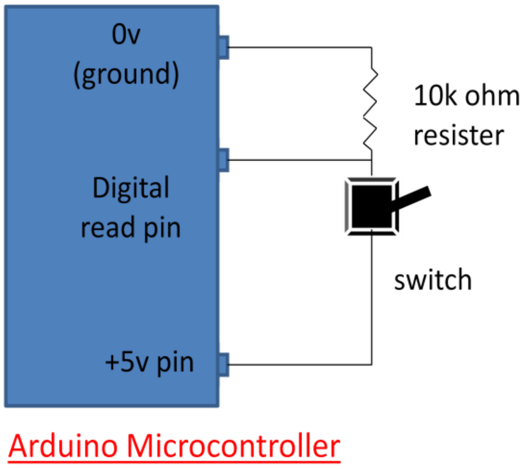


Figure 1 – Circuit C1

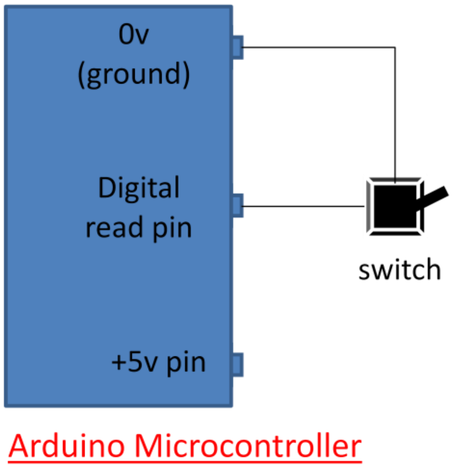


Figure 2 – Circuit C2

Either circuit can be used for *either* type of switch, but the key to configuring correctly lies in the way they are software configured via the `pinMode` function, as follows.

- The `pinMode` setting for initialising circuit_C1 is `pinMode(<pin>, INPUT)`. This has the effect of setting the digital pin <pin> to 0v, representing 'off'. The 10k ohm pull down resistor ensures that the pin stays at 0v until switched, otherwise the input pin will be susceptible to spurious firing from extraneous fields. When the switch is actuated the input rises to +5v which will be detected as the switch transitioning to 'on'.
- For circuit_C2 the `pinMode` setting is `pinMode(<pin>, INPUT_PULLUP)`. This brings into play an internal microcontroller pull up resistor resulting in the digital pin floating at 5v, representing 'off'. No external resistor is required and when the switch is actuated the pin will be brought to 0v which will be detected as the switch transitioning to 'on'.

Note the reversed conditions for 'off' and 'on' between the two circuit schemes. The <switch_lib> will account for this automatically.

Using the <switch_lib> Library

Location of the switch_lib Library

The switch library files should be installed within a directory called "switch_lib" under the Arduino libraries directory - ...\Arduino\libraries\.

The switch_lib directory will comprise four files:

1. switch_lib.h	... header file
2. switch_lib.cpp	... C++ functions
3. keywords.txt	... keyword file to highlight switch_lib keywords
4. switch_lib_user_guide.pdf	... this document, or file elsewhere if required

Steps to Successful Use

Before 'flighting to task' it is recommended to think carefully about what it is you wish to achieve, how switches are incorporated into your project and how switch_lib can be utilised.

The following steps are recommended:

1. Decide how many switches and of what type these will be
2. For each switch decide –
   a. which digital pin will be used?
   b. how will the switch be wired, circuit_C1 or circuit_C2?
3. What will happen when each switch is activated? This step is beyond this UG and is the purpose of your project.

If you are implementing many switches then it may be helpful to make a note of their configurations as once you start wiring and coding things can get a bit muddled up! The following template may be helpful to fill out at the start of your planning and for you to refer to into the development stage (it is also a useful documentation aid post implementation):

Project Name:					Date:
Digital	Switch Type		Digital Pin		Comments
Pin	Button	Toggle	C1	C2	

(add more rows as needed)

For example, switch_lib example 4 configures the following switches, pins and circuits:

Project Name:	switch_lib example 4				Date: 4 March 2021
Digital	Switch Type		Digital Pin		Comments
Pin	Button	Toggle	C1	C2	
2	X		X		2 terminal button
3	X			X	4 terminal button
4	X		X		2 terminal button
5		X		X	3 terminal slide
6		X	X		3 terminal toggle
7		X		X	2 terminal tilt

Having got to grips with what switches, pins and circuit schemes your project will be designed around it is necessary to understand how switch_lib can be used. As with all libraries there are a number of points to consider:

1. We need to ensure our sketch references the library
2. We need to create an instance of the library class, and
3. We need to understand how to correctly use the library's capabilities (e.g. functions and data).

There are a number of steps to be followed –

Step 1

To start, we need to declare the switch_lib library. At the top level of your sketch include the follows statements:

```
#include <Arduino.h>
#include <switch_lib.h>
```

Step 2

Then prior to `setup()` declare how many switches your sketch will be configured for (e.g. `#define num_switches 6`, or `byte num_switches = 6`; etc) together with your switch configuration data (as per the above template?). How you wish to declare this data is very much up to your personal preference. The example sketches, below, show several approaches that you may find instructive.

Step 3

Again, prior to your sketch setup() function and after your switch data declarations, add the following class initiation statement:

```
Switches my_switches(num_switches);
```

Where “`my_switches`” is the name you wish to use for the class you have initiated – this can be anything, but ‘`my_switches`’ is a pretty good name.

Step 4

Okay, we’re off and running? Not quite, before we can plough on and start reading switches we need to declare them to the library with their attributes. We do this by using the function `add.switch`. This function will add a specified switch to the library’s table of active switches such that when it is read (tested) by the read function it will know how it is to be handled. Therefore for *each* switch you wish to configure you will need to add it to the library’s active switch table, for this we use the `add.switch` function;

```
byte switch_id;
switch_id = my_switches.add.switch(button_switch,4,circuit_C2);
```

Things to notice:

- the `add.switch` function call is preceded with the name we have given to the **`Switches`** class, in this example ‘`my_switches`’. This is required to access any resource within the class
- ‘`button_switch’` and ‘`circuit_C2`’ are reserved keywords and are highlighted in red. There are a number of reserved words you may use throughout your sketch, see Specifications - Reserved Macro Definitions
- The function provides a return value. If the addition is successful, this value is the reference you should use whenever you use any of the library resources where switch reference is a required parameter. How you retain this is very much up to your design, but see the example sketches below which should prove helpful. See Specifications - Switch Control Functions `add.switch` to understand the possible return values/conditions.

The best place to ‘install’ your switches is in the `setup()` function, but it can be done anywhere so long as it is only done once and is in scope of the library class statement.

Step 5

Now that is done we can start to read the switches.

The simplest way to read a switch is to use the function `read_switch`. This function is agnostic to switch type and has a single parameter - the id of the switch we wish to read. It will return either ‘`switched`’ or ‘`!switched`’ (again reserved library macros), the meaning being obvious. For example:

```
if(my_switches.read_switch(switch_id) == switched)
{
...do something;
}
```

And that is it. There are other resources available from the library and these are described below.

To recap the steps (five) in order of application/use are:

Step	Example

1	#include <Arduino.h>
	#include <switch_lib.h>
	// plus any other libraries

2	#define num_switches 1 // number of switches to be configured, 1 in this example
	
	// define your switch data
	byte switch_id;
	byte button_pin = 4;
	...

3	**Switches** my_switches(num_switches); // create switch class instance

4	void setup(){
	  ...
	  // declare your switches to the library, for example:
	  switch_id = my_switches.add.switch(button_switch, button_pin,circuit_C2);
	  ...
	}

5	void loop(){
	do{
	  if(my_switches.read_switch(switch_id) == switched)
	  {
	    ...do something;
	  } while (true);
	}

If you need to reference any of the library’s class resources then you must prefix them with the name you have given to the class when you created/initiated it. For example:

```
my_switches.switches[switch_id].switch_type,
my_switches.switches[2].switch_status,
my_switches.add_switch(button_switch,12,circuit_C2),
my_switches.num_free_switch_slots(),
my_switches.set_debounce(25),
my_switches.read_switch(my_switch_data[sw]),
```

Etc.

Specifications

Specifications – Switch Control Structure (SCS)

At the heart of the <switch_lib> library lies a struct(ure) ‘table’ - the switch control structure (SCS), that is used to hold the data attributes for all declared/defined switches.

At initiation of the class, the SCS is created from free memory using a malloc call of sufficient size to match the number of switches the class is being defined for. Thereafter, it may be populated with switches by use of the `add_switch` function (see below) up to the maximum number of switches declared for the class.

The SCS has the following construction and layout:

```
struct switch_control {

  byte switch_type;          // type of switch connected
  byte switch_pin;           // digital input pin assigned to the switch
  byte switch_circuit_type;  // the type of circuit wired to the switch
  bool switch_on_value;      // used for BUTTON SWITCHES only –
                             // defines what "on" means
  bool switch_pending;       // records if switch in transition or not
  long unsigned int switch_db_start;// records debounce start time
                             // when associated switch starts transition
  bool switch_status;        // used for TOGGLE SWITCHES only – current
                             // state of toggle switch.
}
```

Elements of the SCS may be directly accessed from the end user sketch, as required, see above.

Specifications - Reserved Macro Definitions

The table below documents the library's reserved macro definitions. These are available for use by a sketch simply by referencing their name (column 1 and no prefix required), see example sketches. When used they will be coloured in red to show that they are reserved words.

Macro definitions	Values	Significance
`#define button_switch`	1	differentiates switch type, this being of type 'button'
`#define toggle_switch`	2	differentiates switch type, this being of type 'toggle'
`#define circuit_C1`	`INPUT`	switch circuit configured with an external pull down 10k ohm resistor
`#define circuit_C2`	`INPUT_PULLUP`	switch circuit configured without an external pull down resistor
`#define switched`	`true`	signifies switch has been pressed and switch cycle complete, otherwise `!switched`
`#define on`	`true`	used for toggle switch status. Off is `!on`
`#define not_used`	`true`	'not used' indicator – marks if a field in the switch control structure is used or not
`#define bad_params`	-2	invalid `add_switch` parameters
`#define add_failure`	-1	`add_switch` could not insert a given switch, i.e. no slots left

Specifications - Switch Control Functions

Type	int	Name	add_switch
Parameters	byte sw_type, byte sw_pin, byte circ_type parameter choices are: sw_type - is either ‘button_switch’ or ‘toggle_switch’, sw_pin - is the digital pin assigned to the switch, circ_type - is either ‘circuit_C1’ or ‘circuit_C2’.		
Purpose / functionality	This function will add (create) the specified switch (parameters) to the switch control structure, if possible. There are three possible outcomes from an add_switch call: 1. Successful addition of switch. In this case the return value is >= 0 and represents the physical slot (location ‘switch_id/token’) of the switch in the switch control structure. This should be retained by the calling code/design. 2. No further slots available in the switch control structure, all are used. 3. The supplied parameters are ‘bad’. The results of an add_switch call are as below.		
Return values	Return values are: >= 0 success, switch added to switch control struct(ure) - the switch control structure entry number is returned (switch_id/token) for the switch added, -1 add_failure - no slots available in the switch control structure, -2 bad_params - given parameter(s) for switch are not valid.		

Example

```
void create_my_switches() {
  for (int sw = 0; sw < num_switches; sw++) {
    int switch_id =
      my_switches.add_switch(my_switch_data[sw][0], // switch type
                             my_switch_data[sw][1], // digital pin number
                             my_switch_data[sw][2]);// circuit type
    if (switch_id < 0)
    { // There is a data compatibility mismatch (-2),
      // or no room left to add switch (-1).
      Serial.print("Failure to add a switch:\nswitch entry:");
      Serial.print(switch_id);
      Serial.print(", data line = ");
      Serial.print(my_switch_data[sw][0]);
      Serial.print(", ");
      Serial.print(my_switch_data[sw][1]);
      Serial.print(", ");
      Serial.println(my_switch_data[sw][2]);
      Serial.println("!! PROGRAMME TERMINATED !!");
      Serial.flush();
      exit(1);
    } else {
      // 'switch_id' is the switch control slot entry for this switch (sw),
      // so we can use this, if required, to know where our switches are
      // in the control structure by keeping a note of them against their
      // my_switch_data config settings.
      my_switch_data[sw][3] = switch_id;
    }
  }
}  // End create_my_switches
```

Type	int	Name	num_free_switch_slots
Parameters	none		
Purpose / functionality	Returns the number of free slots available in the switch control structure.		
Return values	0 to the maximum number of switches defined		

Example

```
Serial.print("\nNumber of free switch slots in the SCS = ");
Serial.println(my_switches.num_free_switch_slots());
```

Type	bool	Name	read_switch
Parameters	byte switch_id		
Purpose / functionality	Function will read the given switch returning a result as below. Note that the switch_id parameter is the switch entry number in the switch control structure of the switch to be read. This is the returned value from the add_switch function call. If an invalid switch_id is given the read function exits with a return value of !switched. See add_switch() for further information.		
Return values	switched or !switched		

Example

```
  do {
    if (my_switches.read_switch(switch_id) == switched) {
      led_level = HIGH - led_level;  // flip between HIGH and LOW each cycle
      digitalWrite(LED_BUILTIN, led_level);
    }
  } while (true);
```

Type	bool	Name	read_button_switch
Parameters	byte switch_id		
Purpose / functionality	This is used by the read_switch function and deals specifically with reading momentary button style switches. *The function can be used by end user code, but remember that the switch_id parameter is the switch entry number in the switch control structure of the switch to be read.*		
Return values	switched or !switched		

Example

```
if (my_switches.read_button_switch(switch_id) == switched){
  // button switch pressed
  ...
}
```

Type	bool	Name	read_toggle_switch
Parameters	byte switch_id		
Purpose / functionality	This is used by the read_switch function and deals specifically with reading toggle style switches. *The function can be used by end user code, but remember that the* switch_id *parameter is the switch entry number in the switch control structure of the switch to be read.*		
Return values	switched or !switched		

Example

```
if (my_switches.read_toggle_switch(switch_id) == switched){
  // toggle switch switched
  ...
}
```

Type	void	Name	print_switch
Parameters	byte switch_id		
Purpose / functionality	The function prints the switch parameters of the switch defined at slot switch_id in the switch control structure to the serial monitor. It can be helpful in the debugging phase and removed thereafter.		
Return values	none		

Example

```
my_switches.print_switch(3);

Example output a toggle switch, configured as circuit_C2 and occupying slot 3 (switch_id
= 3)in the switch control structure:

slot: 3 sw_type= 2 sw_pin= 5 circ_type= 2 pending= 0 db_start= 0 on_value= 0 sw_status= 0
```

Type	void	Name	print_switches
Parameters	none		
Purpose / functionality	The function prints the switch parameters of ALL switches held in the switch control structure to the serial monitor. It can be helpful in the debugging phase and removed thereafter.		
Return values	none		

Example

```
my_switches.print_switches();

Example output for 6 defined switches - 3 x button & 3 x toggle, configured as either
circuit_C1 or circuit_C2:

slot: 0 sw_type= 1 sw_pin= 2 circ_type= 0 pending= 0 db_start= 0 on_value= 1 sw_status= 1
slot: 1 sw_type= 1 sw_pin= 3 circ_type= 2 pending= 0 db_start= 0 on_value= 0 sw_status= 1
slot: 2 sw_type= 1 sw_pin= 4 circ_type= 0 pending= 0 db_start= 0 on_value= 1 sw_status= 1
slot: 3 sw_type= 2 sw_pin= 5 circ_type= 2 pending= 0 db_start= 0 on_value= 0 sw_status= 0
slot: 4 sw_type= 2 sw_pin= 6 circ_type= 0 pending= 0 db_start= 0 on_value= 1 sw_status= 0
slot: 5 sw_type= 2 sw_pin= 7 circ_type= 2 pending= 0 db_start= 0 on_value= 0 sw_status= 0
```

Type	`void`	Name	`set_debounce`
Parameters	`int period`		
Purpose / functionality	The function may be used to set the debounce period, in milliseconds, for switch reading functions. Note that: 1. the debounce value is set to 10 milliseconds, by default 2. the debounce setting is global and applies to ALL defined switches 3. the parameter value must be >= 0. Negative values are ignored.		
Return values	none		
Example			
`my_switches.set_debounce(20);  // set debounce for 20 msecs`			

Example Sketches

What follows are a number of examples in the use of the switch_lib library. These are provided to aid understanding in how the switch_lib can be applied to your projects.

Each example sketch may be copied and pasted directly into the Arduino IDE, compiled and uploaded without any further coding – just ensure that you have downloaded the switch_lib library files first.

To simplify the example demonstrations, switches are linked to suitably configured LEDs to show switch operations. What a switch will do is clearly something for the designer whether switching low power components or high power ones via low power relays etc.

Any additional components beyond an Arduino microcontroller, connecting wires and a breadboard are indicated for each sketch.

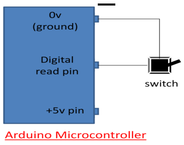
The example switch sketches are:

1. Turning on and off the in-built LED of the Arduino microcontroller (normally on pin 13) using a button switch.
2. Turning on and off the in-built LED of the Arduino microcontroller (normally on pin 13) using a toggle switch.
3. Four switches, two button and two toggle, wired in different schemes, with each switch turning on and off an associated LED.
4. Six switches, three button and three toggle, wired in different schemes, with each switch being processed by its own switch-case statement. In this example button 1 is used to display the switch states of all toggle switches.

"switch_lib_example_1" - Turning LED On/Off With a Button Switch

This example sketch uses a button switch simply connected to turn the Arduino in-built led on and off with each press.

Note that a led state change will only occur when the button switch is released, that is after the completion of the switching cycle.

Components required	Circuit schemes
1 x button switch	circuit_C2  Figure 2 – Circuit C2

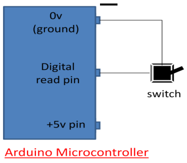
```
/*
   Ron D Bentley, Stafford, UK
   Mar 2021

   -      Example of use of the switch_lib library        -
   Reading single button switch to turn built in led on/off
   ''''''''''''''''''''''''''''''''''''''''''''''''''''''''

   This example and code is in the public domain and
   may be used without restriction and without warranty.

*/
#include <Arduino.h>
#include <switch_lib.h>  // switch_lib .h & .cpp files are stored under
...\Arduino\libraries\switch_lib\

int  switch_id;
bool led_level = LOW;

#define num_switches 1  // only a single switch in this sketch example

// Declare/define the switch instance of given size
Switches my_switches(num_switches);

void setup() {
  // attach a button switch to digital pin 3, no pull down resistor
  // and store the switch's id for later use.
  switch_id = my_switches.add_switch(button_switch, 3, circuit_C2);

  // validate the return
  if (switch_id < 0) {
    // error returned - there is a data compatibilty mismatch (-2),
    // or no room left to add switch (-1).
    Serial.begin(9600);
    Serial.println(F("Failure to add a switch."));
    if (switch_id == -1) {
      Serial.println(F("add_switch - no room to create given switch."));
    } else {
      // can only be that data for switch is invalid
      Serial.println(F("add_switch - one or more parameters is/are invalid."));
    }
    Serial.println(F("!! PROGRAMME TERMINATED !!"));
    Serial.flush();
    exit(1);
  }
  // inititialise built in led and turn to off
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
}
```

```
void loop() {
  // keep reading the switch we have created and toggle the built in
  // led on/off for each press.
  do {
    if (my_switches.read_switch(switch_id) == switched) {
      led_level = HIGH - led_level;  // flip between HIGH and LOW each cycle
      digitalWrite(LED_BUILTIN, led_level);
    }
  } while (true);
}
```

“switch_lib_example_2” - Turning LED On/Off With a Toggle Switch

This example sketch uses a toggle switch, simply connected, to turn the Arduino in-built led on and off. It is essentially the same sketch as in example 1, above, the only difference being that a toggle switch is used instead of a button switch. Compare this sketch with the example 1 sketch and note the *one and only coding* difference – the switch type declared in the `add.switch` call in the `setup()` process, this being ‘`toggle_switch`’ instead of ‘`button_switch`’!

Note that a led state change occurs at each position of the toggle switch.

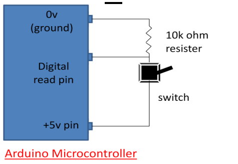
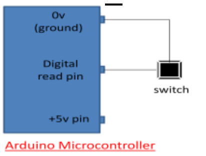
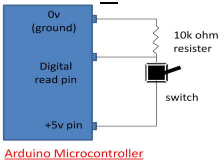
Components required	Circuit schemes
1 x toggle switch	circuit_C2  Figure 2 – Circuit C2

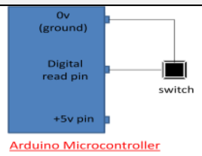
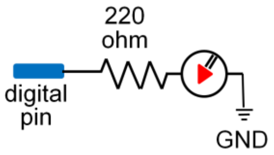
```
/*
   Ron D Bentley, Stafford, UK
   Mar 2021

   -      Example of use of the switch_lib library        -
   Reading single toggle switch to turn built in led on/off
   ''''''''''''''''''''''''''''''''''''''''''''''''''''''''

   This example and code is in the public domain and
   may be used without restriction and without warranty.

*/
#include <Arduino.h>
#include <switch_lib.h>  // switch_lib .h & .cpp files are stored under
...\Arduino\libraries\switch_lib\

int  switch_id;
bool led_level = LOW;

#define num_switches 1  // only a single switch in this sketch example

// Declare/define the switch instance of given size
Switches my_switches(num_switches);

void setup() {
  // attach a button switch to digital pin 3, no pull down resistor
  // and store the switch's id for later use.
  switch_id = my_switches.add_switch(toggle_switch, 3, circuit_C2);

  // validate the return
  if (switch_id < 0) {
    // error returned - there is a data compatibilty mismatch (-2),
    // or no room left to add switch (-1).
    Serial.begin(9600);
    Serial.println(F("Failure to add a switch."));
    if (switch_id == -1) {
      Serial.println(F("add_switch - no room to create given switch."));
    } else {
      // can only be that data for switch is invalid
      Serial.println(F("add_switch - one or more parameters is/are invalid."));
    }
    Serial.println(F("!! PROGRAMME TERMINATED !!"));
    Serial.flush();
    exit(1);
  }
  // inititialise built in led and turn to off
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
}

void loop() {
  // keep reading the switch we have created and toggle the built in
  // led on/off for each press.
  do {
    if (my_switches.read_switch(switch_id) == switched) {
      led_level = HIGH - led_level;  // flip between HIGH and LOW each cycle
      digitalWrite(LED_BUILTIN, led_level);
    }
  } while (true);
}
```

“switch_lib_example_3” - Turning Multiple LEDs On/Off With Multiple Button & Toggle Switches

In this example we see how we can implement and manage a number of button and toggle switches with ease by defining our switch and LED parameters in an orderly way – we shall use a struct(ure) data type to keep everything we need together.

For the purposes of this example we shall connect two button and two toggle switches, each connected with each type of circuit.

Components required	Circuit schemes
1 x button switch	circuit_C1  0v (ground) Digital read pin +5v pin 10k ohm resister switch Arduino Microcontroller Figure 1 – Circuit C1
1 x button switch	circuit_C2  0v (ground) Digital read pin +5v pin switch Arduino Microcontroller Figure 2 – Circuit C2
1 x toggle switch	circuit_C1  0v (ground) Digital read pin +5v pin 10k ohm resister switch Arduino Microcontroller Figure 1 – Circuit C1
1 x toggle switch	circuit_C2

Components required	Circuit schemes
	 0v (ground) Digital read pin +5v pin switch Arduino Microcontroller Figure 2 – Circuit C2
2 x 10k ohm resistors	1 each for each circuit_C1
4 x LEDs	Standard wiring scheme 
4 x 220 ohm resistors	

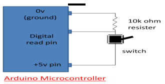
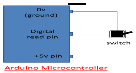
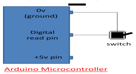
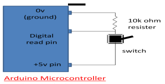
```
/*
   Ron D Bentley, Stafford, UK
   Mar 2021

   -      Example of use of the switch_lib library        -
       Reading button & toggle switches to turn on/off LEDs
   ''''''''''''''''''''''''''''''''''''''''''''''''''''''''

   This example and code is in the public domain and
   may be used without restriction and without warranty.

*/
#include <Arduino.h>
#include <switch_lib.h>  // switch_lib .h & .cpp files are stored under
...\Arduino\libraries\switch_lib\

int  switch_id;

#define not_configured     255  // used to indicate if a switch_control data entry
has be configured

#define num_switches 4

// we will use a struct(ure) data type to keep our switch/LED
// data tidy and readily accessible
struct switch_control {
  byte  sw_type;          // type of switch connected
  byte  sw_pin;           // digital input pin assigned to the switch
  byte  sw_circuit_type;  // the type of circuit wired to the switch
  byte  sw_id;            // holds the switch id given by the add.switch function
for this switch
  byte  sw_led_pin;       // digital pin connecting the LED for this switch
  bool  sw_led_status;    // current status LOW/HIGH of the LED connected to this
switch
} btl[num_switches] = {  // 'btl' = buttons, toggles & LEDs

  button_switch, 6, circuit_C1, not_configured, 2, LOW,// start with LED status LOW
  button_switch, 7, circuit_C2, not_configured, 3, LOW,
  toggle_switch, 8, circuit_C1, not_configured, 4, LOW,
  toggle_switch, 9, circuit_C2, not_configured, 5, LOW

};

// Declare/define the switch instance of given size
Switches my_switches(num_switches);

void setup() {
  // attach each switches to its defined digital pin/circuit type
  // and store the switch's id back in its struct entry for later use.
  for (byte sw = 0; sw < num_switches; sw++) {
    switch_id  = my_switches.add_switch(btl[sw].sw_type,
                                        btl[sw].sw_pin,
                                        btl[sw].sw_circuit_type);
    // validate the return
    if (switch_id < 0) {
      // error returned - there is a data compatibility mismatch (-2),
      // or no room left to add switch (-1).
      Serial.begin(9600);
      Serial.println(F("Failure to add a switch."));
      if (switch_id == -1) {
        Serial.println(F("add_switch - no room to create given switch."));
      } else {
        // can only be that data for switch is invalid
        Serial.println(F("add_switch - one or more parameters is/are invalid."));
      }
      Serial.println(F("!! PROGRAMME TERMINATED !!"));
      Serial.flush();
      exit(1);
    }
    btl[sw].sw_id = switch_id; // store given switch id for this sw for use later
    // now initialise the switch's associated LED and turn off
    pinMode(btl[sw].sw_led_pin, OUTPUT);
    digitalWrite(btl[sw].sw_led_pin, btl[sw].sw_led_status);
  }
}

void loop() {
  // keep reading the switches we have created and toggle their
  // associated LEDs on/off
  do {
    for (byte sw = 0; sw < num_switches; sw++) {
      if (my_switches.read_switch(btl[sw].sw_id) == switched) {
        btl[sw].sw_led_status = HIGH - btl[sw].sw_led_status;  // flip between HIGH
and LOW each cycle
        digitalWrite(btl[sw].sw_led_pin, btl[sw].sw_led_status);
      }
    }
  } while (true);
}
```

"switch_lib_example_4" – Processing More Button & Toggle Switches

In this final example we shall build on the previous examples by implementing six switches – three button and three toggle, to show how we are able to keep adding switches of different wiring schemes. This time, however, we shall not use LEDs to show switch activation, but the serial monitor. We shall also see how we are able to refer to the status of toggle switches outside of them being read by the `read_switch` function and show their status by using a button switch.

To note is that in this example we use multidimensional array to hold our switch data, rather than a struct(ure) as in example 3. You will also see that a switch-case series of statements are used to process the switches once triggered.

Make sure to open the serial monitor once the sketch is compiled and uploaded.

Components required	Circuit schemes
2 x button switch	circuit_C1  0v (ground) Digital read pin +5v pin 10k ohm resister switch Arduino Microcontroller Figure 1 – Circuit C1
1 x button switch	circuit_C2  0v (ground) Digital read pin +5v pin switch Arduino Microcontroller Figure 2 – Circuit C2
2 x toggle switch	circuit_C2  0v (ground) Digital read pin +5v pin switch Arduino Microcontroller Figure 2 – Circuit C2
1 x toggle switch	circuit_C1  0v (ground) Digital read pin +5v pin 10k ohm resister switch Arduino Microcontroller Figure 1 – Circuit C1
3 x 10k ohm resistors	1 each for each circuit_C1

```
/*
   Ron D Bentley, Stafford, UK
   Feb 2021

   -    Example of use of the switch_lib library     -
   Reading Multiple Switch Types, using simple polling
   '''''''''''''''''''''''''''''''''''''''''''''''''''

   This example and code is in the public domain and
   may be used without restriction and without warranty.

*/
#include <Arduino.h>
#include <switch_lib.h>  // switch_lib .h & .cpp files are stored under
                         // ...\Arduino\libraries\switch_lib\

// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// Declare/define specific 'my_data' for 'my_project'
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#define num_switches 6

// Switch to Pin Macro Definition List:
#define my_button_pin_1       2  // digital pin number
#define my_button_pin_2       3  // etc
#define my_button_pin_3       4

#define my_toggle_pin_1       5
#define my_toggle_pin_2       6
```

```
#define my_toggle_pin_3      7

#define not_configured     255  // used to indicate if the my_switch_data entry has
be configured

// Establish type of switch, assigned digital pin and circuit type
// for each switch we are connecting. Until we present each
// switch entry to the add.switch function it will not be
// recorded as configured, hence the use of the final column.
// We start with all defined switches as being 'not_configured'
// Note that:
// 'button_switch', 'toggle_switch', 'circuit_C1' and 'circuit_C2'
// are library defined macros.

byte my_switch_data[num_switches][4] =
{
  button_switch,  my_button_pin_1, circuit_C1, not_configured,
  button_switch,  my_button_pin_2, circuit_C2, not_configured,
  button_switch,  my_button_pin_3, circuit_C1, not_configured,
  toggle_switch,  my_toggle_pin_1, circuit_C2, not_configured,
  toggle_switch,  my_toggle_pin_2, circuit_C1, not_configured,
  toggle_switch,  my_toggle_pin_3, circuit_C2, not_configured
};

// Declare/define the switch instance of given size
Switches my_switches(num_switches);

//
// Set up connected switches as per 'my_switch_data' configs
//

void setup()
{
  Serial.begin(9600);
  // Create/install the defined switches...
  create_my_switches();
  my_switches.set_debounce(20);  // set debounce for 20 msecs
  // Report on my_data set up
  Serial.println(F("\nDeclared & configured switches:"));
  my_switches.print_switches();
  Serial.print(F("\nNumber of free switch slots = "));
  Serial.println(my_switches.num_free_switch_slots());
  Serial.flush();
}

void loop()
{
  do {
    // Poll all switches - examine each connected switch in turn and, if switched,
    // process its associated purpose.
    for (int switch_id = 0; switch_id < num_switches; switch_id++) {
      if (my_switches.read_switch(switch_id) == switched) {
        // This switch ('sw') has been pressed, so process via its switch-case code
        if (my_switches.switches[switch_id].switch_type == button_switch) {
          Serial.print(F("\nbutton switch on digital pin "));
        } else {
          Serial.print(F("\ntoggle switch on digital pin "));
        }
        byte my_switch_pin = my_switches.switches[switch_id].switch_pin;
        Serial.print(my_switch_pin);
        Serial.println(F(" triggered"));
        // Move to switch's associated code section
        switch (my_switch_pin)
        {
          case my_button_pin_1:
            // button switch 1 used to reveal the status of toggle switches
            // as their status is maintained every time they are actuated.
            Serial.println(F("case statement 1 entered"));
            print_toggle_status();
            Serial.flush();
            break;
          case my_button_pin_2:
            Serial.println(F("case statement 2 entered"));
            break;
          case my_button_pin_3:
            Serial.println(F("case statement 3 entered"));
            break;
          case my_toggle_pin_1:
            Serial.print(F("case statement 4 entered, switch is "));
            Serial.println(my_switches.switches[switch_id].switch_status);
            break;
          case my_toggle_pin_2:
            Serial.print(F("case statement 5 entered, switch is "));
            Serial.println(my_switches.switches[switch_id].switch_status);
            break;
          case my_toggle_pin_3:
            Serial.print(F("case statement 6 entered, switch is "));
            Serial.println(my_switches.switches[switch_id].switch_status);
            break;
          default:
            // Spurious switch index!  Should never arise as this is controlled
            // by the for loop within defined upper bound
            break;
        }
        Serial.flush();  // flush out the output buffer
      }
    }
  }
  while (true);
}

//
// Print the current status/setting of each toggle switch configured.
// We scan down my_switch_data to pick out toggle switches and if they
// configured access theit status.
//

void print_toggle_status() {
  Serial.println(F("toggle switches setting: "));
  for (byte sw = 0; sw < num_switches; sw++) {
    if (my_switch_data[sw][0] == toggle_switch &&
        my_switch_data[sw][3] != not_configured) {
      Serial.print(F("toggle switch on digital pin "));
      Serial.print(my_switch_data[sw][1]);
      Serial.print(F(" is "));
      byte switch_id = my_switch_data[sw][3]; // this is the position in the switch
control struct for this switch
      if (my_switches.switches[switch_id].switch_status == on) {
        Serial.println(F("ON"));
      } else {
        Serial.println(F("OFF"));
      }
    }
  }
}

//
// Create a switch entry for each wired up switch, in accordance
// with 'my' declared switch data.
// add_switch params are - switch_type, digital pin number and circuit type.
// Return values from add_switch are:
//    >= 0 the switch control structure entry number ('switch_id') for the switch
//    added,
//      -1 no slots available in the switch control structure,
//      -2 given parameter(s) for switch are not valid.
```

```
void create_my_switches() {
  for (int sw = 0; sw < num_switches; sw++) {
    int switch_id = my_switches.add_switch(my_switch_data[sw][0], // switch type
                                           my_switch_data[sw][1], // digital pin no
                                           my_switch_data[sw][2]);// circuit type
    if (switch_id < 0)
    { // There is a data compatibilty mismatch (-2),
      // or no room left to add switch (-1).
      Serial.print(F("Failure to add a switch:\nswitch entry:"));
      Serial.print(sw);
      Serial.print(F(", data line = "));
      Serial.print(my_switch_data[sw][0]);
      Serial.print(F(", "));
      Serial.print(my_switch_data[sw][1]);
      Serial.print(F(", "));
      Serial.println(my_switch_data[sw][2]);
      Serial.println(F("!! PROGRAMME TERMINATED !!"));
      Serial.flush();
      exit(1);
    } else {
      // 'switch_id' is the switch control slot entry for this switch (sw),
      // so we can use this, if required, to know where our switches are
      // in the control structure by keeping a note of them against their
      // my_switch_data config settings.
      my_switch_data[sw][3] = switch_id;
    }
  }
}  // End create_my_switches
```