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

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 and guidance 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.

- Example sketches

  example sketches are provided, building from single button and toggle switches to multiple switch types using both circuit schemes and using different programming techniques to declare and use switch data.

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
- allowing the optional linking of switches to a digital output such that auto-switching of the output can occur without end user programmatic coding
- 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 project developers 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, 12 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, optionally, 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.

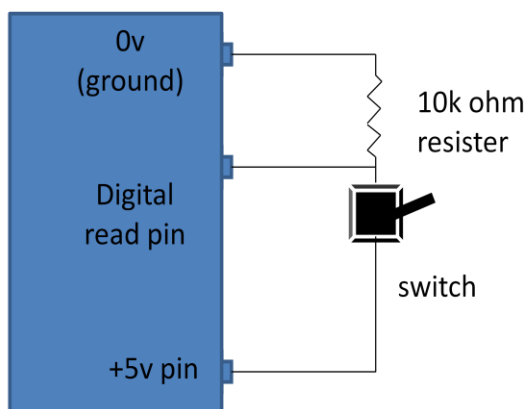| Button / momentary | Slide | Toggle | Tilt |

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 distinct and separate switch transitions. The status of toggle switches therefore persists after being

physically switched – they stay ON or OFF. `<switch_lib>` automatically handles these physical characteristics.
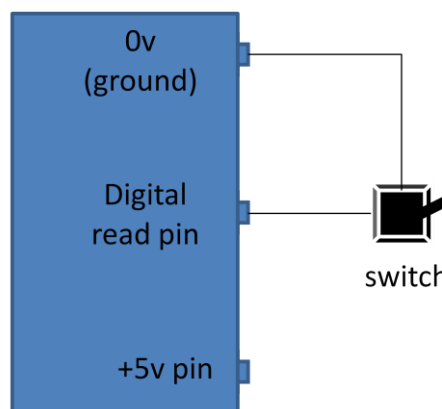
Common Switch Wiring Schemes

If you now appreciate the differences between switch types, it is necessary to understand how they should be connected to the microcontroller and the differences between the two commonly seen switch wiring schemes *without* hardware debounce. These 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 is essential and 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 principal considerations are:

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?
   c. do we wish to create a link from the switch to a digital output pin and, if so, what state should the output pin be initialised to at creation time?
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:	
Pin	Switch Configs				Linked Outputs			Notes	
	Switch Type		Circuit Type		Pin	Initial Value			
	Button	Toggle	C1	C2		LOW	HIGH		

(add more rows as needed)

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

Project Name:	Leds & Relays							Date:	4 March 2021
Pin	Switch Configs				Linked Outputs			Notes	
	Switch Type		Circuit Type		Pin	Initial Value			
	Button	Toggle	C1	C2		LOW	HIGH		
2		X	X		8		X	Relay 1 - no switching coding	
3		X		X	9	X		Led 1 - no switching coding	
4		X	X					Relay 2 - needs switching coding	
5	X			X				Led 2 - needs switching coding	
6	X		X		10		X	Relay 3 - no switching coding	
7	X			X	11	X		Led 3 - no switching coding	

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 along 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(s) 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. They each define the switch and circuit type, respectively. There are a number of reserved words you may use throughout your sketch, see Specifications - Reserved Macro Definitions and Switch Control Functions add.switch to understand the possible parameters and return values/conditions
- 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 add/create 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.

Example - 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 added, 1 in this example```
	
	```// define your switch data```
	```byte switch_id;```
	```byte button_pin = 4;```
	```...```

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

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

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

Additionally to note:

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.
  byte switch_out_pin;      // the digital pin mapped to this switch,
                            // if any
  bool switch_out_pin_status;// the status of the mapped pin
```

Members 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 / Comments
#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 <u>with</u> an external pull down 10k ohm resistor
#define circuit_C2	INPUT_PULLUP	switch circuit configured <u>without</u> an external pull down resistor
#define switched	true	A value returned by read_switch, read_button_switch & read_toggle_switch functions. 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	A value returned by add_switch function - invalid parameters
#define add_failure	-1	A value returned by add_switch function - could not insert a given switch, i.e. no slots left
#define link_success	0	A value returned by link_switch_to_output function - output pin successfully linked to a switch
#define link_failure	-1	A value returned by link_switch_to_output function - output pin could not be linked to a switch

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	**`link_switch_to_output`**
Parameters	`byte switch_id, byte output_pin, bool HorL`		

Purpose / functionality	This function will link the given digital pin (`output_pin`) to an already created/defined switch (switch_id) and initialise it according to the specified parameter, as follows:
	
	`pinMode(output_pin, HorL)`, where `HorL` is either `LOW` or `HIGH`
	
	Once linked, the output pin will be flipped between LOW/HIGH or HIGH/LOW each time the associated switch is read (tested) by the `read_switch` function and found to have been actuated (switched).
	
	Note that:
	1. This feature allows simple digital output pin switching without any requirement for end user coding.
	2. The output can be initialised at either LOW or HIGH level.
	3. Automatic output pin flipping only occurs via use of the `read_switch` function. If switches have a linked output and they need to be read *without* automatic output pin flipping then use `read_button_switch` / `read_toggle_switch`, instead. These functions will not flip associated switch output levels.
	4. *Redefining a linked switch output* - existing defined switch linked outputs can be redefined, as required, by further calls to the switch link function.
	5. *Removing a linked switch output* - if a switch has a linked output defined and it is necessary to remove it then this can be done by a call to `link_switch_to_output` with an output pin value of 0. The output level will be set according to the `HorL` parameter. For example:
	
	`link_result = my_switches.link_switch_to_output(`
	`        switch_id,`
	`        0,`
	`        LOW);// set output level to low`
	
	However, if the output level is to remain unaltered then set the `HorL` parameter to '`my_switches`'.`switches[switch_id]`.`switch_out_pin_status`, where '`my_switches`' is the name of your class for the **Switches** class.
	
	For example:
	
	`link_result = my_switches.link_switch_to_output(`
	`        switch_id,`
	`        0,`
	`        my_switches.switches[switch_id].`
	`                switch_out_pin_status);//don't change`		

Return values	`0, link_success` – linkage was successful `-1, link_failure` – linkage failed, switch_id not in range of defined switches		
Example			

Type	int	Name	link_switch_to_output

```
   int link_result = my_switches.link_switch_to_output(
                      switch_id,
                      LED_BUILTIN,
                      LOW);
   if (link_result == link_failure ) {
     // linking failed, invalid switch id
     Serial.begin(9600);
     Serial.println(F("Failure to link an output to a switch"));
     Serial.println(F("!!PROGRAM TERMINATED!!"));
     Serial.flush();
     exit(2);
   }

Example 2:
   // Link/associate this switch to the relay output pin
   // with the switch we have just installed/created so that every
   // time the switch is actuated the relay will be automatically
   // flipped.  Start with relay at HIGH setting.
   int link_result = my_switches.link_switch_to_output(
                      switch_id,
                      relay_1,
                      HIGH);
   if (link_result == link_failure ) {
     // linking failed, invalid switch id
     Serial.begin(9600);
     Serial.println(F("Failure to link an output to a switch"));
     Serial.println(F("!!PROGRAM TERMINATED!!"));
     Serial.flush();
     exit(2);
   }
```

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: 1. 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. 2. If an invalid switch_id is given the read function exits with a return value of !switched. 3. If a switch has a linked output pin associated with it then this function will flip the current output pin level, i.e. from LOW to HIGH, or from HIGH to LOW. See add_switch and link_switch_to_pin for further information.		
Return values	switched or !switched		

Type	bool	Name	**read_switch**

<table>
<tr><td colspan="4" align="center">Example</td></tr>
</table>

```
Example 1:
  // the switch does not have and output pin linked to it, so we
  // need to handle the flipping of the LED
  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);

Example 2:
  // the switch has been defined with a linked/associated output pin
  // connected to a LED.
  // We therefore have nothing to do but keep reading the switch and the
  // LED will be automatically flipped for us.
  do {

    my_switches.read_switch(switch_id);

  } 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 note: 1. Remember that the switch_id parameter is the switch entry number in the switch control structure of the switch to be read. 2. If the switch has a linked/associated output pin then it will *not* be processed.		
Return values	switched or !switched		

<table>
<tr><td colspan="4" align="center">Example</td></tr>
</table>

```
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 note: 1. Remember that the switch_id parameter is the switch entry number in the switch control structure of the switch to be read. 2. If the switch has a linked/associated output pin then it will *not* be processed.		
Return values	switched or !switched		

<table>
<tr><td colspan="4" align="center">Example</td></tr>
</table>

Type	bool	Name	read_toggle_switch

```
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
                op_pin= 0 op_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
                op_pin= 8 op_status= 0
slot: 1 sw_type= 1 sw_pin= 3 circ_type= 2 pending= 0 db_start= 0 on_value= 0 sw_status= 1
                op_pin= 9 op_status= 1
slot: 2 sw_type= 1 sw_pin= 4 circ_type= 0 pending= 0 db_start= 0 on_value= 1 sw_status= 1
                op_pin= 0 op_status= 0
slot: 3 sw_type= 2 sw_pin= 5 circ_type= 2 pending= 0 db_start= 0 on_value= 0 sw_status= 0
                op_pin= 0 op_status= 0
slot: 4 sw_type= 2 sw_pin= 6 circ_type= 0 pending= 0 db_start= 0 on_value= 1 sw_status= 0
                op_pin= 0 op_status= 0
slot: 5 sw_type= 2 sw_pin= 7 circ_type= 2 pending= 0 db_start= 0 on_value= 0 sw_status= 0
                op_pin= 0 op_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 from the github links provided for each example sketch, compiled and uploaded without any further coding – just ensure that you have downloaded the `<switch_lib>` library files first.

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

The example sketches are:

1. Example 1.1 - turning on and off the in-built LED of the Arduino microcontroller (normally on pin 13) using a <u>button</u> switch using direct coding.
2. Example 1.2 – as for example 1.1 but using the function `link_switch_to_output` to flip the in-built LED using indirect coding.
3. Example 2,1 - turning on and off the in-built LED of the Arduino microcontroller (normally on pin 13) using a <u>toggle</u> switch using direct coding.
4. Example 2.2 – as for example 2.1 but using the function `link_switch_to_output` to flip the in-built LED using indirect coding.
5. Example 3.1 - four switches, two button and two toggle, wired in different schemes, with each switch turning on and off an associated LED using direct coding.
6. Example 3.2 - as for example 3.1 but using the function `link_switch_to_output` to flip each of the associated LEDs using indirect coding.
7. Example 4 - six switches, three button and three toggle, wired in different schemes, with each switch being processed by its own switch-case statement. The sketch incorporates a mix of direct coding and use of the function `link_switch_to_output` to control the effects of the switches, using the serial monitor, LEDs and relays.
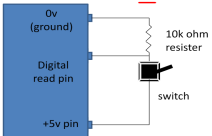
All of the sketches can be accessed from github - follow the specific github link with each example. Alternatively, the main `<switch_lib>` github page with the `<switch_lib>` files (.h, .cpp, .txt and .pdf files) can be found at this link.

Example 1.1 - Turning LED On/Off With a Button Switch, Directly Coded

This example sketch uses a button switch and will turn the Arduino in-built led on and off with each press. The switch is wired for `circuit_C1`.

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

The sketch can be also accessed from github, here.

Components required	Circuit schemes
1 x button switch	circuit_C1 Arduino Microcontroller Figure 1 – Circuit C1

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

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   -          Example of use of the switch_lib library         -
   Example 1.1
   Reading single button switch to turn built in led on/off.
   When the button switch is actuated, the in-built led
   (LED_BUILTIN), will be flipped ON/OFF etc by using suitable
   coding in the sketch's main loop.
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   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;   // start with led off

#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 2, with
  // an external pull down resistor, circuit_C1,
  // and store the switch's id for later use.
  switch_id = my_switches.add_switch(
               button_switch,
               2,
               circuit_C1);
  // validate the return
  if (switch_id < 0) {
    // Error returned - there is a data compatibility mismatch (-2, bad_params),
    // or no room left to add switch (-1, add_failure).
    Serial.begin(9600);
    Serial.println(F("Failure to add a switch"));
    if (switch_id == add_failure) {
      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("!!PROGRAM TERMINATED!!"));
    Serial.flush();
    exit(1);
  }
  // Initialise 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) {
      // Flip between HIGH and LOW each cycle
      led_level = HIGH - led_level;
      digitalWrite(LED_BUILTIN, led_level);
    }
  } while (true);
}
```
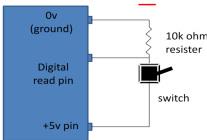
Example 1.2 - Turning LED On/Off With a Button Switch, Indirectly Coded

This example sketch uses a button switch to turn the Arduino in-built led on and off with each press, indirectly, by using the `link_switch_to_output` function. Compare this sketch with example 1.1 sketch and note the differences – the button switch is linked to an output pin and no code exists in the sketch to flip the output pin (in-built LED). The switch is wired for `circuit_C1`.

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

The sketch can be accessed from github here.

Components required	Circuit schemes
1 x button switch	circuit_C1 Figure 1 – Circuit C1

```
/*
   Ron D Bentley, Stafford, UK
   Mar 2021
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   -           Example of use of the switch_lib library        -
   Example 1.2
   Reading single button switch to turn built in led on/off, but
   in this example we shall link the switch to an output pin
   (LED_BUILTIN) using a switch_lib function, so that when
   actuated, the output pin will be automatically flipped
   HIGH-LOW etc each time the button switch is pressed WITHOUT
   any further coding.
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   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 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 2, with
  // an external pull down resistor, circuit_C1,
  // and store the switch's id for later use.
  switch_id = my_switches.add_switch(
              button_switch,
              2,
              circuit_C1);
  // validate the return
  if (switch_id < 0) {
    // Error returned - there is a data compatibilty mismatch (-2, bad_params),
    // or no room left to add switch (-1, add_failure).
```

```cpp
    Serial.begin(9600);
    Serial.println(F("Failure to add a switch"));
    if (switch_id == add_failure) {
      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("!PROGRAM TERMINATED!!"));
    Serial.flush();
    exit(1);
  }

  // Link/associate this switch to the in-built led (normally pin 13)
  // with the switch we have just installed/created so that every
  // time the switch is actuated the built in LED will be automatically
  // flipped.  Start with LED at low setting.
  int link_result = my_switches.link_switch_to_output(
                       switch_id,
                       LED_BUILTIN,
                       LOW);
  if (link_result == link_failure ) {
    // linking failed, invalid switch id
    Serial.begin(9600);
    Serial.println(F("Failure to link an output to a switch"));
    Serial.println(F("!!PROGRAM TERMINATED!!"));
    Serial.flush();
    exit(2);
  }
}

void loop() {
  do {
    // just keep reading, LED_BUILTIN will automatically be flipped for us
    // so we dont need to do anything else
    my_switches.read_switch(switch_id);
  }
  while (true);
}
```
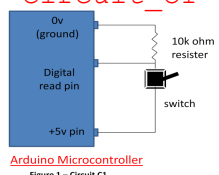
Example 2.1 - Turning LED On/Off With a Toggle Switch, Directly Coded

This example sketch uses a toggle switch and will turn the Arduino in-built led on and off with each actuation. The sketch is essentially the same as example 1.1, the difference being that a toggle switch type is declared instead of a button type. The switch is wired for circuit_C1.

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

The sketch can be accessed from github here.

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

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

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   -       Example of use of the switch_lib library       -
   Example 2.1
   Reading single toggle switch to turn built in led on/off.
   When the toggle switch is activated, the in-built led
   (LED_BUILTIN), will be flipped ON/OFF etc by using suitable
   coding in the sketch's main loop.
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   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 toggle switch to digital pin 2, with
  // an external pull down resistor, circuit_C1,
  // and store the switch's id for later use.
  switch_id = my_switches.add_switch(
                toggle_switch,
                2,
                circuit_C1);

  // Validate the return
  if (switch_id < 0) {
    // Error returned - there is a data compatibilty mismatch (-2, bad_params),
    // or no room left to add switch (-1, add_failure).
    Serial.begin(9600);
    Serial.println(F("Failure to add a switch"));
    if (switch_id == add_failure) {
```

```
      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("!!PROGRAM TERMINATED!!"));
    Serial.flush();
    exit(1);
  }
  // Initialise 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) {
      // flip between HIGH and LOW each cycle
      led_level = HIGH - led_level;
      digitalWrite(LED_BUILTIN, led_level);
    }
  } while (true);
}
```
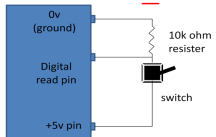
Example 2.2 - Turning LED On/Off With a Toggle Switch, Indirectly Coded

This example sketch uses a toggle switch to turn the Arduino in-built led on and off, indirectly, by using the `link_switch_to_output` function. It is essentially the same sketch as in example 2.1, above. Compare this sketch with example 2.1 sketch and note the differences – the toggle switch is linked to an output pin and no code exists in the sketch to flip the output pin (in-built LED).

Note that a led state change occurs at <u>each</u> position of the toggle switch.

The sketch can be accessed from github <u>here</u>.

Components required	Circuit schemes
1 x toggle switch	circuit_C1 Figure 1 – Circuit C1

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

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   -       Example of use of the switch_lib library       -
   Example 2.2
   Reading single toggle switch to turn built in led on/off.
   Toggle switch is associated with an output pin (LED_BUILTIN)
   using a switch_lib function, so that when activated, the
   output pin will be automatically flipped, HIGH-LOW etc  each
   time the toggle switch is actuated WITHOUT any further coding.
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   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 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 toggle switch to digital pin 2, with
  // an external pull down resistor, circuit_C1,
  // and store the switch's id for later use.
  switch_id = my_switches.add_switch(
              toggle_switch,
              2,
              circuit_C1);
  // Validate the return
  if (switch_id < 0) {
    // Error returned - there is a data compatibilty mismatch (-2, bad_params),
    // or no room left to add switch (-1, add_failure).
    Serial.begin(9600);
    Serial.println(F("Failure to add a switch"));
```

```
    if (switch_id == add_failure) {
      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("!!PROGRAM TERMINATED!!"));
    Serial.flush();
    exit(1);
  }

  // Link/associate the LED_BUILT digital pin (normally pin 13)
  // with the switch we have just installed/created
  // so that every time the switch is activated the built in
  // LED will be automatically be flipped.  Start with LED at HIGH setting.
  int link_result = my_switches.link_switch_to_output(
                        switch_id,
                        LED_BUILTIN,
                        HIGH);
  if (link_result == link_failure) {
    // Linking failed, invalid switch id
    Serial.begin(9600);
    Serial.println(F("Failure to link an output to a switch"));
    Serial.println(F("!!PROGRAM TERMINATED!!"));
    Serial.flush();
    exit(2);
  }
}

void loop() {
  do {
    if (my_switches.read_switch(switch_id) == switched) {
      // just keep reading, LED_BUILTIN will automatically be flipped for us
      // so we dont need to do anything else
    }
  } while (true);
}
```
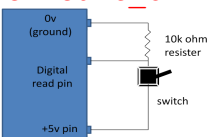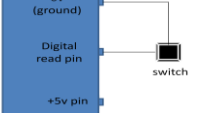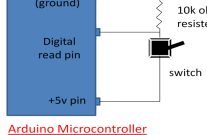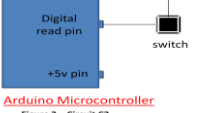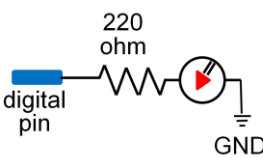
Example 3.1 - Turning Multiple LEDs On/Off With Multiple Button & Toggle Switches, Directly Coded

In this example we build on previous examples and see how we can implement and manage a number of button and toggle switches with ease by defining our switch and associated switching output (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.

The sketch can be accessed from github here.

Components required	Circuit schemes
1 x button switch	circuit_C1 0v (ground) 10k ohm resister Digital read pin switch +5v pin Arduino Microcontroller **Figure 1 – Circuit C1**
1 x button switch	circuit_C2 0v (ground) Digital read pin switch +5v pin Arduino Microcontroller **Figure 2 – Circuit C2**
1 x toggle switch	circuit_C1 0v (ground) 10k ohm resister Digital read pin switch +5v pin Arduino Microcontroller **Figure 1 – Circuit C1**
1 x toggle switch	circuit_C2 0v (ground) Digital read pin switch +5v pin Arduino Microcontroller **Figure 2 – Circuit C2**
2 x 10k ohm resistors	1 each for each circuit_C1
4 x LEDs	Standard wiring scheme for connected LED
4 x 220 ohm resistors	220 ohm digital pin GND

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

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   -       Example of use of the switch_lib library       -
   Example 3.1
   Reading multiple button & toggle switches wired with
   different circuit types with each switch turning associated
   leds on/off.
   This example uses a struct(ure) data type to define the
```

```cpp
   switch and led data.
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   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\

#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 {      // struct member meanings:
  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

  //.................switch data.............>  <led data, initial setting level>
  button_switch, 2, circuit_C1, not_configured, 8,  HIGH,
  button_switch, 3, circuit_C2, not_configured, 9,   LOW,
  toggle_switch, 4, circuit_C1, not_configured, 10, HIGH,
  toggle_switch, 5, circuit_C2, not_configured, 11,  LOW
};

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

void setup() {
  // Attach each switch 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++) {
    int 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, bad_params),
      // or no room left to add switch (-1, add_failure).
      Serial.begin(9600);
      Serial.println(F("Failure to add a switch"));
      if (switch_id == add_failure) {
        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("!!PROGRAM 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 on/off according to
preset
    pinMode(btl[sw].sw_led_pin, OUTPUT);
    digitalWrite(btl[sw].sw_led_pin, btl[sw].sw_led_status);
```
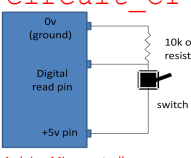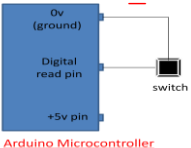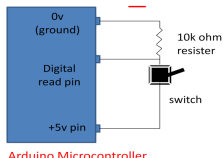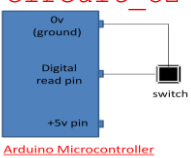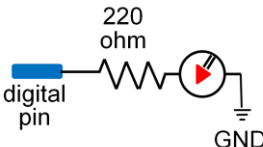
```
  }
}

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

Example 3.2 - Turning Multiple LEDs On/Off With Multiple Button & Toggle Switches, Indirectly Coded

In this example we build on the previous example 3.1 and see how we can implement and manage a number of button and toggle switches *without* direct coding for the outputs.   In the example sketch we remove the direct coding that deals with the switching of the LED outputs and, instead, we use the function `link_switch_to_output` to associate each switch to an LED output pin. It is essentially the same sketch as in example 3.1, above. Compare this sketch with example 3.1 sketch and note the differences.

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

The sketch can be accessed from github here.

Components required	Circuit schemes
1 x button switch	**circuit_C1** Arduino Microcontroller Figure 1 – Circuit C1
1 x button switch	**circuit_C2** Arduino Microcontroller Figure 2 – Circuit C2
1 x toggle switch	**circuit_C1** Arduino Microcontroller Figure 1 – Circuit C1
1 x toggle switch	**circuit_C2** Arduino Microcontroller Figure 2 – Circuit C2
2 x 10k ohm resistors	1 each for each **circuit C1**
4 x LEDs	Standard wiring scheme for connected LED
4 x 220 ohm resistors	

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

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   -          Example of use of the switch_lib library        -
   Example 3.2
   Reading multiple button & toggle switches wired with
   different circuit types with each switch linked to an
   output pin using a switch_lib function, so that when
   activated, the associated switch output pin will be
   automatically flipped, HIGH-LOW etc  each time the switch
   is actuated WITHOUT any further coding.

   To demonstrate, the switch associated outputs are connected
   to 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\

#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 {      // struct member meanings:
  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_output_pin;       // digital pin to associate switch to
  bool  sw_output_level;     // define the status level of the defined output pin on
set up
} btl[num_switches] = {      // 'btl' = buttons, toggles & LEDs

  //................switch data...............>  <output pin initial setting
level>
  button_switch, 2, circuit_C1, not_configured,  8, HIGH,
  button_switch, 3, circuit_C2, not_configured,  9,  LOW,
  toggle_switch, 4, circuit_C1, not_configured, 10, HIGH,
  toggle_switch, 5, circuit_C2, not_configured, 11,  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++) {
    int 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, bad_params),
      // or no room left to add switch (-1, add_failure).
      Serial.begin(9600);
```

```
      Serial.println(F("Failure to add a switch"));
      if (switch_id == add_failure) {
        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("!!PROGRAM TERMINATED!!"));
      Serial.flush();
      exit(1);
    }
    btl[sw].sw_id = switch_id; // store given switch id for this sw for use later

    // Now associate the defined out for this switch so that every time the switch
    // is activated the associated output will be automatically be flipped.
    // set the output level to whatever is defined in the initialisation data.
    int link_result = my_switches.link_switch_to_output(
                        switch_id,
                        btl[sw].sw_output_pin,
                        btl[sw].sw_output_level);
    if (link_result == link_failure) {
      // Linking failed, invalid switch id
      Serial.begin(9600);
      Serial.println(F("Failure to link an output to a switch"));
      Serial.println(F("!!PROGRAM TERMINATED!!"));
      Serial.flush();
      exit(2);
    }
  }
}

void loop() {
  do {
    for (byte sw = 0; sw < num_switches; sw++) {
      if (my_switches.read_switch(btl[sw].sw_id) == switched) {
        // Just keep reading, the read function will automatically
        // flip the associated switch output pins for us so we
        // dont need to do anything else
      }
    }
  } while (true);
}
```

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 we shall incorporate outputs to the serial monitor, LEDs and relays, as follows:

The sketch is configured for 6 switches, 3 x toggle and 3 x button with switches performing the following actions:

- toggle 1 switches a relay, without direct coding, using output linking
- toggle 2 switches a led, without direct coding, using output linking
- toggle 3 also switches a led by direct coding , i.e. not via switch linking
- button 1 produces a switch report using a `<switch_lib>` function
- button 2 switches a relay, without direct coding, using output linking
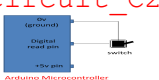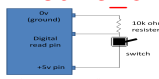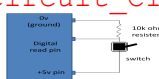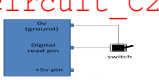- button 3 switches a led, without direct coding, using output linking

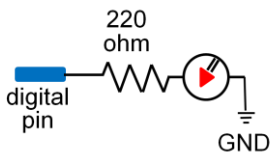The switches are polled in succession and processing occurs via a switch-case set of control statements.

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 a multidimensional array to hold our switch data, rather than a struct(ure) as in example 3.1/3.2 – you decide with approach is best. 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 and set to 9600 baud and note the switch circuit schemes are changed from previous set ups, just to further mix things up!

The sketch can be accessed from github [here](#).

Components required	Circuit schemes
2 x button switch	circuit_C2
1 x button switch	circuit_C1
2 x toggle switch	circuit_C1
1 x toggle switch	circuit_C2
3 x 10k ohm resistors	1 each for each circuit_C1
3 x LEDs	Standard wiring scheme
3 x 220 ohm resistors	for connected LED

Components required	Circuit schemes
2 x 5v relays	
Serial monitor	9600 baud

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

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   -           Example of use of the switch_lib library           -
   Example 4
   Reading multiple switches (6) of different types and of mixed wiring
   schemes. Additionally, some switches are linked to a digital
   output such that when they are actuated the linked output level
   is flipped (HIGH->LOW, or LOW->HIGH) automatically without the
   need for any end user coding.

   Switch data in this example is preset in a two dimension array
   and may be varied as appropriate.

   The sketch is configured for 6 switches, 3 toggle and 3 button
   with switches performing the following actions:
   toggle 1 switches a relay, without direct coding, using output linking
   toggle 2 switches a led, without direct coding, using output linking
   toggle 3 also switches a led, but not via switch linking
   button 1 produces a switch report using a switch_lib function
   button 2 switches a relay, without direct coding, using output linking
   button 3 switches a led, without direct coding, using output linking

   The switches are polled in succession and processing occurs
   via a switch-case set of control statements.
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   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_toggle_pin_1  2 // digital pin number
#define my_toggle_pin_2  3 // etc
#define my_toggle_pin_3  4

#define my_button_pin_1  5
#define my_button_pin_2  6
#define my_button_pin_3  7

// Digital output pins for linking to switches:
#define relay_1    8  // output pin for relay 1
#define relay_2    9  // output pin for relay 2
#define led_1     10  // output pin for led 1
#define led_2     11  // output pin for led 2
#define led_3     12  // output pin for led 3
```

```
#define not_configured 255  // used to indicate if the my_switch_data switch output
pin is to 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.
//
// Array row definitions are:
// [sw][0] = switch type, button or toggle
// [sw][1] = digital input pin for the switch
// [sw][2] = how the switch is wired/connected
// [sw][3] = this stores the switch_id returned from add.switch function
//           to be used in all calls to the library's functions where
//           switches are referenced
// [sw][4] = the digital output pin linked to this switch, if defined
// [sw][5] = the level the output pin is to be set to at initialisation (linking),
//           if a linked output is configured - must be LOW or HIGH
//
// Note that:
// 'on', 'switched', 'button_switch', 'toggle_switch', 'circuit_C1'
// and 'circuit_C2' are reserved library defined macros.

byte my_switch_data[num_switches][6] =
{
  // <...............switch data.............>  <.output pin data.>
  toggle_switch, my_toggle_pin_1, circuit_C1, 0, relay_1, HIGH,// linked to relay_1
  toggle_switch, my_toggle_pin_2, circuit_C2, 0, led_1, LOW,// linked to led_1
  toggle_switch, my_toggle_pin_3, circuit_C1, 0, not_configured, 0,// not linked,
flip led_3 by direct code

  button_switch, my_button_pin_1, circuit_C2, 0, not_configured, 0,// not linked,
produces switch report
  button_switch, my_button_pin_2, circuit_C1, 0, relay_2, HIGH,// linked to relay_2
  button_switch, my_button_pin_3, circuit_C2, 0, led_2, LOW // linked to led_2
};

// 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();

  // Set debounce for 20 msecs
  my_switches.set_debounce(20);

  // initialise the output pin for led_3, as we will deal with
  // flipping this led by direct coding
  pinMode(led_3, OUTPUT);
  digitalWrite(led_3, LOW);
}

void loop()
{
  do {
    // Poll all switches - examine each connected switch in turn and, if switched,
    // process its associated purpose.
    for (int sw = 0; sw < num_switches; sw++) {
      byte switch_id = my_switch_data[sw][3];  // extract the switch id for this
switch, sw
      if (my_switches.read_switch(switch_id) == switched) {
```

```
        // This switch ('switch_id') 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_toggle_pin_1:
            // toggle switch 1 triggers a relay (1) which is a linked output
            // so nothing to do here to process the relay
            Serial.print(F("relay 1 switched"));
            break;
          case my_toggle_pin_2:
            // toggle switch 2 flips a led (1) which is a linked output
            // so nothing to do here to process the relay
            Serial.print(F("led 1 switched"));
            break;
          case my_toggle_pin_3:
            // direct coding to flip led_3 following switch actuation (toggle 3)
            static bool led_3_status = LOW; // static because we need to retain
current state between switching
            led_3_status = HIGH - led_3_status; // flip led status
            digitalWrite(led_3, led_3_status);
            Serial.print(F("led 3 switched "));
            break;
          case my_button_pin_1:
            // button switch 1 used to reveal the current status of the switch
control structure
            // members, number of free switch control slots and the on/off status
of all
            // all toggle switches as their status is maintained
            my_switches.print_switches();  // confirm we are set up correctly
            // Report number of free switch slots remaining
            Serial.print(F("\nNumber of free switch slots remaining = "));
            Serial.println(my_switches.num_free_switch_slots());
            // Report on the current status of the toggle switches
            print_toggle_status();
            Serial.flush();
            break;
          case my_button_pin_2:
            // button switch 2 triggers a relay (2) which is a linked output
            // so nothing to do here to process the relay
            Serial.print(F("relay 2 switched"));
            break;
          case my_button_pin_3:
            // button switch 3 flips a led (2) which is a linked output
            // so nothing to do here to process the relay
            Serial.print(F("led 2 switched"));
            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.
```

```cpp
// We scan down my_switch_data to pick out toggle switches and if they are
// configured access their status.
//

void print_toggle_status() {
  Serial.println(F("\nToggle switches setting: "));
  for (byte sw = 0; sw < num_switches; sw++) {
    if (my_switch_data[sw][0] == toggle_switch) {
      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 number
        my_switch_data[sw][2]);// circuit type
    if (switch_id < 0)
    { // There is a data compatibilty mismatch (-2, bad_params),
      // or no room left to add switch (-1, add_failure).
      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("!!PROGRAM TERMINATED!!"));
      Serial.flush();
      exit(1);
    } else {
      // 'switch_id' is the switch control slot entry for this switch (sw),
      // so we can use this to know where our switches are
      // in the control structure by keeping a note of them in their
      // my_switch_data config settings.
      my_switch_data[sw][3] = switch_id;

      // Now deal with any linked output requirement
      if (my_switch_data[sw][4] != not_configured) {
        // there is an output defined for this switch, so link it
        int link_result =
          my_switches.link_switch_to_output(
            switch_id,              // id of switch to link output to
            my_switch_data[sw][4],  // digital output pin number
            my_switch_data[sw][5]); // initial level, HIGH or LOW
        if (link_result == link_failure ) {
```

```
            // linking failed, invalid switch id
            Serial.println(F("Failure to link an output to a switch"));
            Serial.println(F("!!PROGRAM TERMINATED!!"));
            Serial.flush();
            exit(2);
        }
      }
    }
  }
}  // End create_my_switches
```