

# ControlIT Python Library

Version 1.0, 5/6/2015

This document describes the Deltronics Python library for the Control and Datacapture Interface and ControlIT Extra. The library uses PyUSB to handle the data communications. This has been tested on Raspberry Pi (see the installation instructions) and should also work on most Linux distributions. The library is written for Python 2.7.x. PyUSB reportedly supports Python 3, so it should also be possible to use the library with Python 3 if PyUSB is installed in the Python 3 path.

*Note regarding zero- and one-based indices.*

In this library all inputs, outputs and motors are referenced using zero-based indices; however, the labelling on some Deltronics control interface boxes is one-based. If the box has outputs labelled (for example) 1, 2, ..., 6, then these will need to be indexed as 0, 1, ..., 5 in any Python program. Some of the library functions accept (or return) Python lists to describe input and output states; since Python lists are zero-based, this scheme leads to more consistent code.

## Initialization

---

```
controlit.initialize()
```

Set up the connection to the interface. This function should be called before any other functions in the library.

---

## Output functions

---

```
controlit.set_output( ioutput, value )
```

Set the value of an output. The integer parameter `ioutput` is a zero-based index, and `value` is a boolean – `False` is off, and `True` is on.

Example:

```
controlit.set_output( 3, True )
```

– sets the fourth output (0-based) on. The LED next to output 4 (with one-based labelling) should illuminate.

---

`controlit.set_outputs( vallist )`

`vallist` is a list of boolean output values. The function traverses the list, setting outputs 0, 1, 2 etc to the values in the list.

Examples

```
vals = [True, False, False]
controlit.set_outputs( vals )
```

– sets output 0 on, outputs 1 and 2 off

```
vals = [ True for i in range(0, 16) ]
controlit.set_outputs( vals )
```

– sets outputs 0-15 on (16 outputs are available on ControlIT Extra only).

---

`controlit.clear_outputs()`

Turns all outputs off.

---

## Motor Functions

Motors are referred to with a zero-based index. Motor A is index 0, motor B is 1, and so on.

---

`controlit.set_motor_direction( imotor, onoff )`

Turns motor `imotor` (an integer from 0 to 3) on or off (parameter `onoff` is a boolean). Note that the speed of the motor defaults to zero.

---

`controlit.set_motor_speed( imotor, speed )`

Sets the speed of motor `imotor` (an integer from 0 to 3) to `speed` (an integer from 0 to 31).

---

`controlit.set_motor_direction( imotor, direction )`

Sets the direction of motor `imotor` (an integer from 0 to 3) – `direction` is an integer which must be set to 0 or 1.

---

## Digital Input Functions

---

`controlit.get_inputs()`

Returns a Python list of boolean values corresponding to the inputs. This has 6 or 16 values, corresponding to the number of inputs on the interface box.

Example: Testing whether inputs 0 and 4 are on

```
inputList = controlit.get_inputs()

if ( inputList[0] and inputList[4] ):
    print "inputs 0 and 4 on"
```

---

`controlit.get_input( iinput )`

Returns a boolean value corresponding to the input at `iinput` (a zero-based index).

---

## Analogue Input Functions

---

`controlit.get_analogue_data( port )`

Returns an integer corresponding to the value of the digitised analogue input at `port` (a zero-based index). The A-D converter returns a 10-bit value, so the integer will be in the range  $0 \rightarrow 2^{10} - 1$  (0-1023).

---

`controlit.get_analogue_sensor_id( port )`

Returns an integer which uniquely identifies the type of sensor attached to analogue input `port` (a 0-based index).

---

## Example program

This simple program updates the interface on an infinite loop, using the `time.sleep()` function to set the rate of updates to 100 per second. The program sequences the output lights, and reverses the direction of the sequence if input 0 on the control box is on.

```
import controlit as ci
import time

# initialize the interface
ci.initialize()

i = 0
direction = 1

while ( True ):
    i += direction

    # clamp i between 0 and 5
    if ( i > 5 ):
        i -= 6
    if ( i < 0 ):
        i += 6

    # set all outputs false
    ci.clear_outputs()
    # set output i on
    ci.set_output( i, True )

    # reverse direction is input 0 is connected
    if ( ci.get_input( 0 ) ):
        direction = -1
    else:
        direction = 1

    # sleep for 1/100 second
    time.sleep( 0.01 )
```