RASPBERRY PI

If you have not set up I²C communications on your Raspberry Pi, you will need to do this first (this only needs to be done once).

Section 1

Get I²C tools

• Download the I²C-tools utility by entering the following command in the terminal

```
sudo apt-get install i2c-tools
```

• Enable I²C support in the kernel using the raspi-config utility

```
sudo raspi-config
```

• In raspi-config, go to advanced options and select enable I²C
This should set up I²C on the Raspberry Pi automatically
Afterwards reboot the Raspberry Pi

```
sudo reboot
```

• Test that the I²C works with the following command

```
sudo i2cdetect -y 1 (or sudo i2cdetect -y 0 on older models)
```

This command will show which devices are at which addresses on the I²C bus

If the steps above didn't successfully enable I²C, check that everything is set correctly by following these steps:

• Edit the module files

```
sudo nano /etc/modules
```

• Add the following lines at the end if they aren't there already

```
i2c-bcm2708
i2c-dev
```

• Save the files

• Edit the blacklist file

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

• And remove I²C from the blacklist
comment it out by putting a # in front of the line

```
#blacklist i2c-bcm2708
```

Kernals past 3.18 need to enable I²C in the device tree

```
sudo nano /boot/config.tx
```

Add the following lines at the end of the file if they aren't there already

```
dtparam=i2c1=on (or dtparam=i2c0=on on older models)
dtparam=i2c_arm=on
```

Section 2

Sample code

```
#1/usr/bin/python

import io # used to create file streams
import fcntl # used to access I²C parameters like addresses

import time # used for sleep delay and timestamps
import string # helps parse strings


class atlas_i2c:
    long_timeout = 1.5 # the timeout needed to query readings and calibrations
    short_timeout = .3 # timeout for regular commands
    default_bus = 1 # the default bus for I²C on the newer Raspberry Pis, certain older boards use bus 0
    default_address = 99 # the default address for the pH sensor

    def __init__(self, address = default_address, bus = default_bus):
        # open two file streams, one for reading and one for writing
        # the specific I²C channel is selected with bus
        # it is usually 1, except for older revisions where its 0
        # wb and rb indicate binary read and write
        self.file_read = io.open("/dev/i2c-"+str(bus), "rb", buffering = 0)
        self.file_write = io.open("/dev/i2c-"+str(bus), "wb", buffering = 0)

        # initializes I²C to either a user specified or default address
        self.set_i2c_address(address)

    def set_i2c_address(self, addr):
        # set the I²C communications to the slave specified by the address
        # The commands for I²C dev using the ioctl functions are specified in
        # the i2c-dev.h file from i2c-tools
        I2C_SLAVE = 0x703
        fcntl.ioctl(self.file_read, I2C_SLAVE, addr)
        fcntl.ioctl(self.file_write, I2C_SLAVE, addr)

    def write(self, string):
        # appends the null character and sends the string over I²C
        string += "\00"
        self.file_write.write(string)

    def read(self, num_of_bytes = 31):
        # reads a specified number of bytes from I²C, then parses and displays the result
        res = self.file_read.read(num_of_bytes) # read from the board
        response = filter(lambda x: x != '\x00', res) # remove the null characters to get the response
        if(ord(response[0]) == 1): # if the response isnt an error
            char_list = map(lambda x: chr(ord(x) & ~0x80), list(response[1:])) # change MSB to 0 for all received characters except the first and get a list of characters
            # NOTE: having to change the MSB to 0 is a glitch in the raspberry pi, and you shouldn't have to do this!
            return "Command succeeded " + ''.join(char_list) # convert the char list to a string and returns it
        else:
            return "Error " + str(ord(response[0]))

    def query(self, string):
        # write a command to the board, wait the correct timeout, and read the response
        self.write(string)

        # the read and calibration commands require a longer timeout
        if((string.upper().startswith("R")) or
            (string.upper().startswith("CAL"))):
            time.sleep(self.long_timeout)
        else:
            time.sleep(self.short_timeout)

        return self.read()

    def close(self):
        self.file_read.close()
        self.file_write.close()

def main():
    device = atlas_i2c() # creates the I²C port object, specify the address or bus if necessary

    print(">> Atlas Scientific sample code")
    print(">> Any commands entered are passed to the board via I2C except:")
    print(">> Address,xx changes the I2C address the Raspberry Pi communicates with.")
    print(">> Poll,xx.x command continuously polls the board every xx.x seconds")
    print(" where xx.x is longer than the %0.2f second timeout." %  atlas_i2c.long_timeout)
    print(" Pressing ctrl-c will stop the polling")

    # main loop
    while True:
        input = raw_input("Enter command: ")

        # address command lets you change which address the Raspberry Pi will poll
        if(input.upper().startswith("ADDRESS")):
            addr = int(string.split(input, ',')[1])
            device.set_i2c_address(addr)
            print("I2C address set to " + str(addr))

        # contiuous polling command automatically polls the board
        elif(input.upper().startswith("POLL")):
            delaytime = float(string.split(input, ',')[1])

            # check for polling time being too short, change it to the minimum timeout if too short
            if(delaytime < atlas_i2c.long_timeout):
                print("Polling time is shorter than timeout, setting polling time to %0.2f" % atlas_i2c.long_timeout)
                delaytime =  atlas_i2c.long_timeout

            # get the information of the board you're polling
            info = string.split(device.query("I"), ",")[1]
            print("Polling %s sensor every %0.2f seconds, press ctrl-c to stop polling" % (info, delaytime))

            try:
                while True:
                    print(device.query("R"))
                    time.sleep(delaytime - atlas_i2c.long_timeout)
            except KeyboardInterrupt: # catches the ctrl-c command, which breaks the loop above
                print("Continuous polling stopped")

        # if not a special keyword, pass commands straight to board
        else:
            print(device.query(input))



if __name__ == '__main__':
    main()
```

Click here to download the *.py file