Laboratories of Day 1

Table 1. Day one

Laboratory	Code
1: Installing the Arduino IDE and Mbili features	Hello_world.ino, Leds1_2.ino
2: Serial Communication	Echo.ino, add_sub.ino, add_sub_div_mul.ino, Delimiters_sum.ino
3: RTC: setting time, reading time	RTC_date_update.ino
4: Reading temperature from internal RTC temperature sensor	RTC_date_Volt_Temp.ino
5: SD: writing data to card	SD_write.5.ino
6: reading temperature, timestamping reading and saving to SD card	RTC_date_Volt_Temp_SD.ino
7: Adding a Timer to Schedule Readings	RTC_TPH_SD_Timer.ino, RTC_date_Volt_Temp_SD_Timer.ino

1. Getting Started

Here are the steps required in order to get started with the SODAQ Mbili board.

1. Install the Arduino IDE & Software[1]
2. Install the SODAQ Mbili Files and Libraries[2]
3. Select the SODAQ Mbili Hardware Profile[3]
4. Configure the Serial Port[4]
5. Configure a Serial Monitor[5]

[1] http://mbili.sodaq.net/howtodownloads/#step1
[2] http://mbili.sodaq.net/howtodownloads/#step2
[3] http://mbili.sodaq.net/howtodownloads/#step3
[4] http://mbili.sodaq.net/howtodownloads/#step4
[5] http://mbili.sodaq.net/howtodownloads/#step5

1.1. Installing the Arduino IDE & Software

The first step required is to download and install, the latest version of the Arduino IDE. You will need version 1.5.X or higher (presently 1.5.8 Beta). The Arduino IDE is available both as an installer and as a zipped version.

You can find the files and installation instructions for different platforms here: Arduino Software[6]

After downloading and installing the software, you need to run the Arduino IDE once for it to create a the sketchbook directory called *Arduino* inside your documents folder. You can then close the program.

More advanced users may wish to use Eclipse, instructions and details of the setup process can be found here: Using Eclipse with Arduino[7]

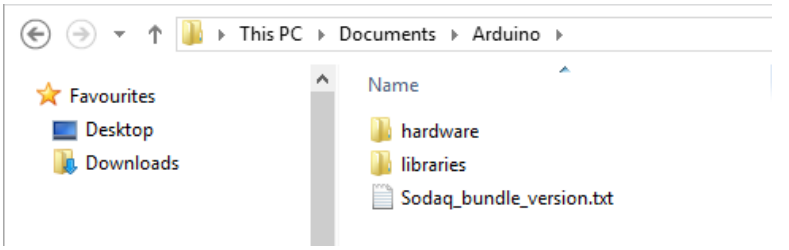
1.2. Installing the SODAQ Mbili Files and Libraries

The next step is to download the SODAQ Mbili files here: Sodaq_bundle[8]

You must unpack this zip file and place the contents in the *Arduino* sub-folder of your documents folder (the folder that was created by the Arduino IDE the first time it was run).

In Windows it is located in: *C:\Users\yourusername\Documents\Arduino*.

The contents of that *Arduino* folder should now look like this:



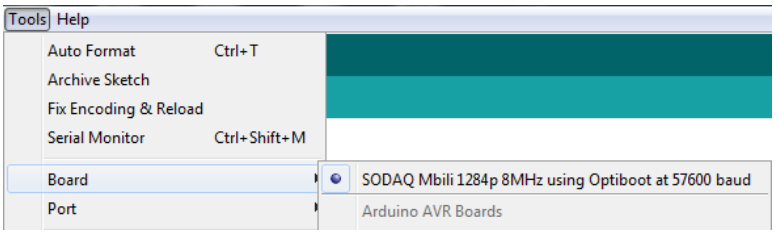
In Linux it is located in: /usr/share/Arduino

In Apple OSX it is located in: /Users/*username*/Documents/Arduino

6 http://arduino.cc/en/Main/Software
7 http://playground.arduino.cc/Code/Eclipse
8 http://mbili.sodaq.net/wp-content/uploads/2015/04/Sodaq_bundle.zip

1.3. Selecting the SODAQ Mbili Hardware Profile

The SODAQ Mbili files you copied in step two include a hardware profile for the SODAQ Mbili board. Restart the Arduino IDE. The IDE will now have the SODAQ board added to the list found in the menu *Tools→Board*,



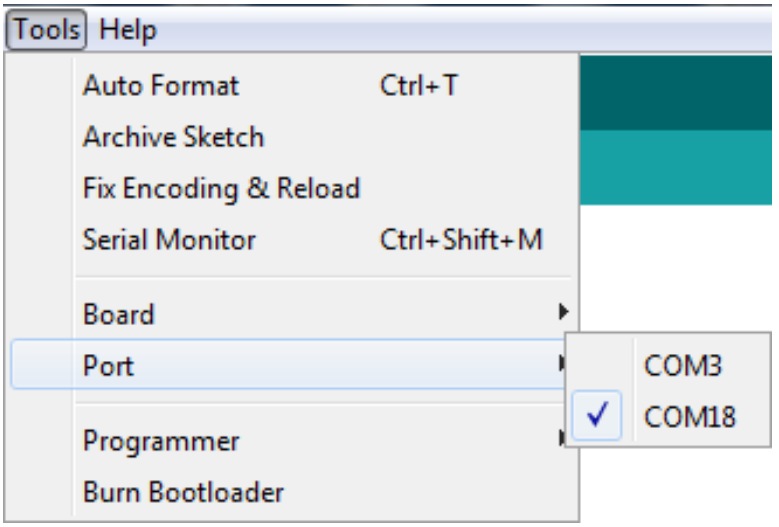
You simply need to select that hardware profile as the board that the Arduino IDE will use.

1.4. Configuring the Serial Port

Windows versions 7 and 8 will normally find the right USB driver when you plug in the SODAQ Mbili for the first time. The same is also true for Mac and Linux. If your system doesn't find the driver you will have to download the FTDI drivers from here: FTDI Drivers[9].

The FTDI driver adds a virtual serial port. In Windows this is *COMx* (so *COM1, COM8,* etc.). On Linux and Mac the port name starts with */dev/tty.*

You can find the list of available serial ports in the menu under *Tools→Port*.

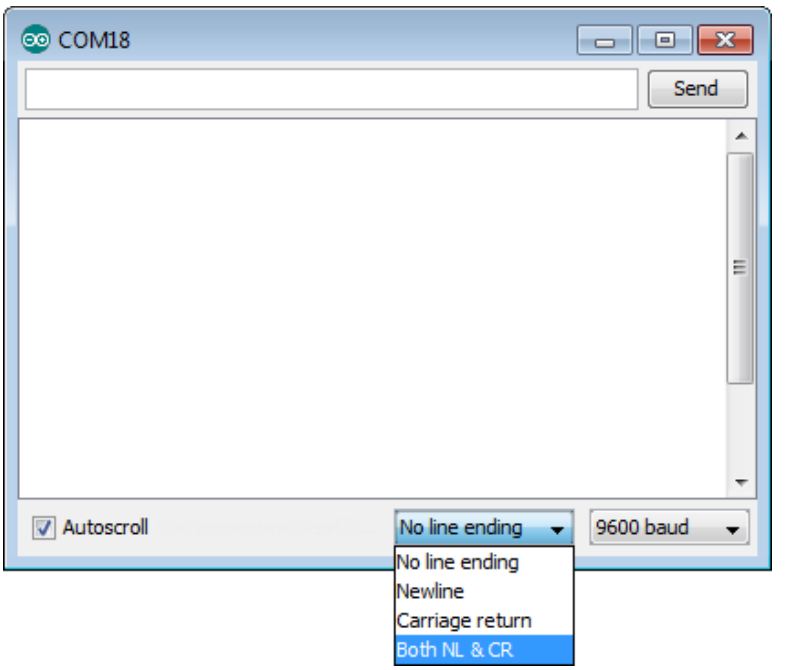


[9] http://www.ftdichip.com/Drivers/VCP.htm

You must select the serial port associated with the SODAQ Mbili board. The associated serial port is only visible in that list when the SODAQ Mbili board is connected and switched on. If you are unsure which is the correct port simply check what new port has been added after you switch the device on.

1.5. Configuring Serial Monitor

In order to read data coming from the board, we will use the Serial Monitor. You can open the serial monitor built in the IDE by pressing *Ctrl-Shift-M* or through the menu *Tools→Serial Monitor*. The default setting does not add *CR/LF* to any commands you send. You will want to change this using the drop down options shown below:



Alternatively, you can use a terminal emulator such as PuTTY. The download page can be found here: PuTTY Download[10] and you can find Instructions on the configuration settings here: PuTTY Configuration.[11]

The Arduino IDE will automatically close the built in serial monitor before uploading a sketch. If you are using a terminal emulator you must close

[10] http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
[11] https://binglongx.wordpress.com/2011/09/27/arduino-and-putty/

the session before you can upload a sketch. If you do not, you will get an error message stating that access is denied to the communications port (it is busy due to another active session).

Whenever a terminal session is opened, a reset command is sent to the device. Your uploaded sketch will then restart and you will see any messages that might be displayed at startup. There is no need to rush to try and open the serial monitor immediately after uploading your sketch.

2. Loading and Running a Basic Demo

You are now ready to upload and run your first sketch. Ensure that the SODAQ Mbili board is connected and switched on, and that you have configured it as described above (selecting the SODAQ Mbili hardware profile and the right serial port).

Simply copy and paste the following code into a new sketch.

Hello_world.ino

```
void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("Starting...");
}

void loop()
{
  // put your main code here, to run repeatedly:
  Serial.println("Hello World");
  delay(1000);
}
```

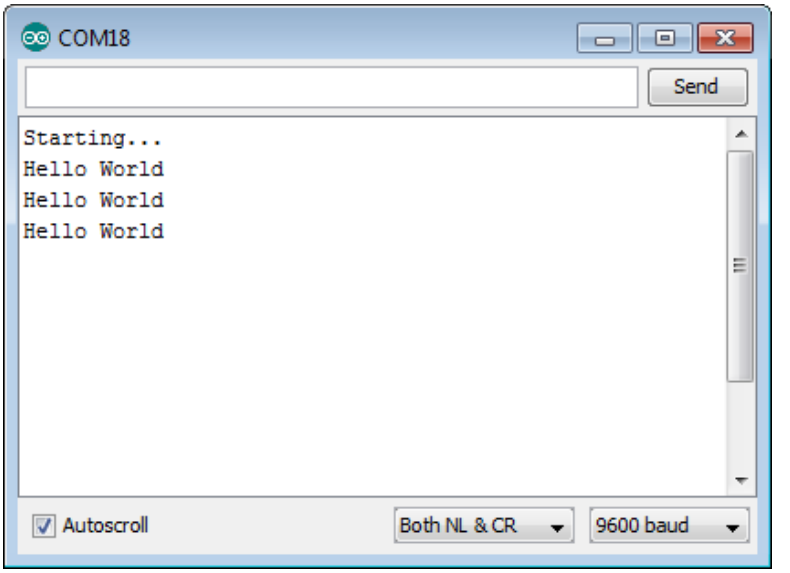
Click 

to optionally compile and test the code and then

Click 

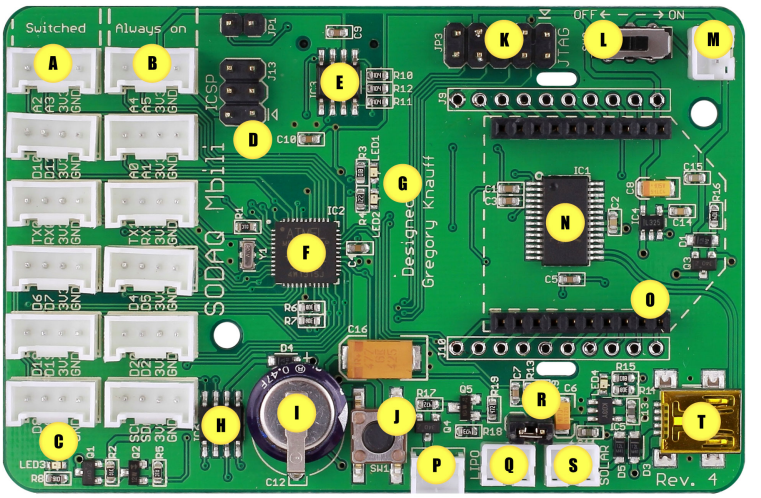
to compile and upload it to the SODAQ Mbili board.

If you open the built in serial monitor by pressing *Ctrl-Shift-M* you should see a startup message displayed, and then “Hello World” displayed repeatedly at intervals of one second.



3. Onboard LEDs

The SODAQ Mbili board has two user programmable onboard LEDs. These are LED1 (green) and LED2 (red). The onboard position of these can be seen here indicated as G.



These two LEDs can be controlled using the output from the digital pins they are connected to. In this example we will demonstrate how to switch on and off these LEDs using the same methods used for digital I/O (output only). The same methods can be used for controlling the output to components attached to other digital pins.

LED1 & *LED2* are the constants which define which digital pin each LED is attached to.

Table 2. LED example

Required Components	SODAQ Mbili Board
Required Libraries	None
Hardware Setup	Connect the SODAQ board to the PC via USB.
Source Code	**Leds1_2.ino**

Turn on the SODAQ Mbili board, compile and upload the following sketch from the Arduino IDE onto the device. You should see the green and red LEDs alternately light up for one second at a time.

Leds1_2.ino

```
//How long to activate each LED
#define DELAY_TIME 1000

void setup()
{
  //LED1
  pinMode(LED1, OUTPUT);
  digitalWrite(LED1, LOW);

  //LED2
  pinMode(LED2, OUTPUT);
  digitalWrite(LED2, LOW);
}

void loop()
{
  //Switch LED1 on then off again after DELAY_TIME (ms)
  digitalWrite(LED1, HIGH);
  delay(DELAY_TIME);
  digitalWrite(LED1, LOW);

  //Repeat for LED2
  digitalWrite(LED2, HIGH);
  delay(DELAY_TIME);
  digitalWrite(LED2, LOW);
}
```

Let's understand what each part of the code is doing.

- Here we define a constant which is used to control how long each LED is turned on for. The specified value is in milliseconds and so each LED remains lit for one second at a time.

```
//How long to activate each LED
#define DELAY_TIME 1000
```

- Here we set each of the digital pins for the LEDs to OUTPUT[12] mode using the pinMode()[13] method. Additionally, we turn them off initially using the digitalWrite()[14] method and the constant LOW[15].

```
void setup()
{
  //LED1
  pinMode(LED1, OUTPUT);
  digitalWrite(LED1, LOW);

  //LED2
  pinMode(LED2, OUTPUT);
  digitalWrite(LED2, LOW);
}
```

- First we switch *LED1* on using digitalWrite()[16] and the constant HIGH[17]. We then call the method delay()[18] which returns after the specified number of milliseconds (*DELAY_TIME*) have elapsed. Finally, we switch off *LED1* with another call to digitalWrite()[19] this time using the constant LOW[20]. The same process is repeated for *LED2*.

```
void loop()
{
  //Switch LED1 on then off again after DELAY_TIME (ms)
  digitalWrite(LED1, HIGH);
```

12 http://arduino.cc/en/Reference/Constants
13 http://arduino.cc/en/Reference/pinMode
14 http://arduino.cc/en/Reference/digitalWrite
15 http://arduino.cc/en/Reference/Constants
16 http://arduino.cc/en/Reference/digitalWrite
17 http://arduino.cc/en/Reference/Constants
18 http://arduino.cc/en/reference/delay
19 http://arduino.cc/en/Reference/digitalWrite
20 http://arduino.cc/en/Reference/Constants

```
  delay(DELAY_TIME);
  digitalWrite(LED1, LOW);

  //Repeat for LED2
  digitalWrite(LED2, HIGH);
  delay(DELAY_TIME);
  digitalWrite(LED2, LOW);
}
```

Excercise: Change delay time by a factor of 0.5 in LED1 and 1.5 in LED2.

4. Serial Communication

The SODAQ Mbili has two hardware serial connections *Serial* & *Serial1*. *Serial* is the USB connection and *Serial1* the connection to the Bee module. They are also connected to two of the Grove sockets. See here for details: Grove sockets[21].

In this tutorial we will discuss sending data over a serial connection (USB) between the SODAQ Mbili and a connected PC. This can be a useful way of displaying debug data (in the Serial Monitor) or for sending commands to the device.

The Serial[22] class inherits from the Stream[23] class and the behaviour is similar to other Stream[24] based classes. Most of the functionality involves reading from and writing to the incoming and outgoing stream buffers. Below is a brief description of some of the methods which you might use for this purpose.

Some of the links here reference the documentation of the parent class Stream[25].

To open the serial connection to the PC a call must be made to Serial.begin(speed)[26]. The parameter *speed* specifies the bits per second or baud rate. This must match the baud rate set in the Serial Monitor or in the terminal emulator program you are using. Similarly a call to Serial.end()[27] will close the connection.

21 http://mbili.sodaq.net/?page_id=81
22 http://arduino.cc/en/reference/serial
23 http://arduino.cc/en/Reference/Stream
24 http://arduino.cc/en/Reference/Stream
25 http://arduino.cc/en/Reference/Stream
26 http://arduino.cc/en/Serial/Begin
27 http://arduino.cc/en/Serial/End

4.1. Reading from the Serial Connection (Input)

Data which is received from the PC is stored in the incoming stream buffer. Here are some of the methods available for reading from the incoming stream buffer:

- read()[28]: Reads and returns a single byte (character) from the incoming stream buffer.
- readBytes(buffer, length)[29]: Copies *length* number of bytes (characters) from the incoming stream buffer into the parameter *buffer*.
- readString()[30]: Reads characters from the incoming stream buffer and returns the data as a String. Each of these methods read varying amounts of data from the incoming stream buffer and either return that data directly, or in the case ofhttp://arduino.cc/en/Serial/ReadBytes[readBytes()] writes the data into the supplied parameter *buffer*. The data which is read is also removed from the incoming stream buffer.

These methods are only able to return data when the incoming stream buffer is not empty. Calling them when the incoming stream buffer is empty will likely result in a timeout (unless new data is received before the timeout duration has elapsed). A call to the method available()[31] returns the number of bytes (characters) currently stored in the incoming stream buffer and can used to determine if any data has been received. Additionally, the timeout duration can be set with a call to the method setTimeout()[32].

4.2. Writing to the Serial Connection (Output)

Sending data to the PC involves writing data to the outgoing stream buffer. Here are several of the methods which can be used for writing to the outgoing stream buffer:

- print(val, base)[33]: Writes the data from *val* into the outgoing stream buffer. If val is a numerical type, the optional second parameter *base* can be used to specify the formatting.
- println(val, base)[34]: Functions the same as the print() method but also appends the carriage return (ASCII 13, or '\r') and a newline characters (ASCII 10, or '\n').

28 http://arduino.cc/en/Serial/Read
29 http://arduino.cc/en/Serial/ReadBytes
30 http://arduino.cc/en/Reference/StreamReadString
31 http://arduino.cc/en/Serial/Available
32 http://arduino.cc/en/Serial/setTimeout
33 http://arduino.cc/en/Serial/Print
34 http://arduino.cc/en/Serial/Println

4.3. ECHO ECHO

Table 3. ECHO ECHO example

Required Components	SODAQ Mbili Board
Required Libraries	none
Hardware Setup	Connect the SODAQ board to the PC via USB.
Source Code	**Echo.ino**

The following code will make the Arduino ECHO anything you send to it. Therefore, if you type a 3, the Arduino will send back a 3. If you type a letter F, the Arduino will send back a letter F. Enter the following code into your Arduino IDE and upload it to your Arduino.

Echo.ino

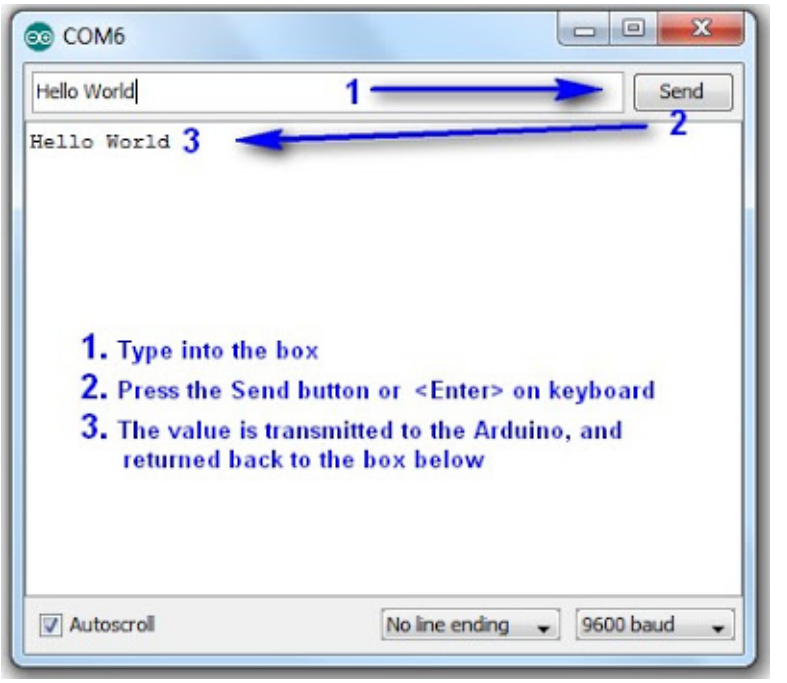
```
/* Use a variable called byteRead to temporarily store
   the data coming from the computer */
byte byteRead;

void setup() {
// Turn the Serial Protocol ON
  Serial.begin(9600);
}

void loop() {
   /*  check if data has been sent from the computer: */
  if (Serial.available()) {
    /* read the most recent byte */
    byteRead = Serial.read();
    /*ECHO the value that was read, back to the serial port. */
    Serial.write(byteRead);
  }
}
```

- Once the Arduino sketch has been uploaded to the Arduino. Open the Serial monitor, which looks like a magnifying glass at the top right section of the Arduino IDE. Please note, that you need to keep the USB connected to the Arduino during this process, as the USB cable is your communication link between your computer and the Arduino.

- Type anything into the top box of the Serial Monitor and press <Enter> on your keyboard. This will send a series of bytes to the Arduino. The Arduino will respond by sending back your typed message in the larger textbox.



- Please note that we are using **Serial.write(byteRead);** on line 18 to get the Arduino to ECHO the message back to you on your computer.

Exercises

1. Delete lines 16 to 18, and replace them with the following line : **Serial.write(Serial.read());**

This essentially eliminates the byteRead variable in the sketch above. But we will be using it later on, so once you have tested it out, put the code back together as originally displayed.

2. Replace line 18 with a **Serial.println** instead of **Serial.write Serial.println(byteRead);**

Once uploaded, type **1** <enter> **2** <enter> **3** <enter> into the Serial Monitor.You should see: 495051 Serial.print and Serial.println will send back the actual ASCII code, whereas Serial.write will send back the actual text. See ASCII codes[35] for more information.

[35] http://www.asciitable.com/

3. Try typing in numbers like 1.5 or 2.003 or -15.6 into the Serial Monitor using Serial.write and Serial.print or Serial.println commands as described before. You will notice that the decimal point transmits as a number using Serial.print or Serial.println, and will transmit as a decimal point when using Serial.write

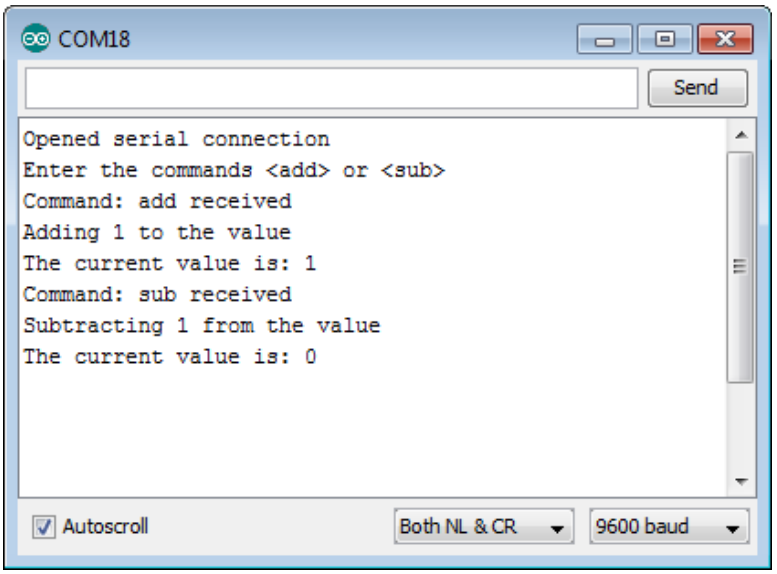
4.4. Operations

Table 4. Operations example

Required Components	SODAQ Mbili Board
Required Libraries	none
Hardware Setup	Connect the SODAQ board to the PC via USB.
Source Code	**add_sub.ino**

Turn on the SODAQ Mbili board, compile and upload the sketch below from the Arduino IDE onto the SODAQ Mbili board. Leave the USB cable plugged in and open the Serial Monitor (Ctrl-Shift-M) and ensure that it is set to the 9600 baud rate.

If you try sending the command ‘add’ followed by ‘sub’ you should see output similar to this:



add_sub.ino

```
#define INPUT_DELAY 100 //Specifies how often we check for input
int value = 0; //A variable which is modified by commands from the serial connection
```

```
void setup()
{
  //Open the port
  Serial.begin(9600);
  Serial.println("Opened serial connection");

  //Print initial message
  Serial.println("Enter the commands <add> or <sub>");
}

void loop()
{
  //Check if any data has been received
  //If so call the method handleInput() to process it
  if (Serial.available() > 0)
  {
    handleInput();
  }

  //This delay dictates how often the device will check for input
  delay(INPUT_DELAY);
}

void handleInput()
{
  //Get the input string
  String input = Serial.readString();

  //Remove any whitespace or CR/LF
  input.trim();

  //Echo the input
  Serial.println("Command: " + input + " received");

  //Process the input
  if (input == "add")
  {
    Serial.println("Adding 1 to the value");
    value++;
  }
  else if (input == "sub")
  {
    Serial.println("Subtracting 1 from the value");
    value--;
  }
  else
```

```
  {
    Serial.println("Unknown command: " + input);
  }

  //Echo the changes
  Serial.print("The current value is: ");
  Serial.println(value);
}
```

- Here we define a delay value which is used to control the effective frequency that the sketch checks for any new incoming data. We also declare a global variable, *value*, which we will modify via commands sent from the PC.

```
#define INPUT_DELAY 100 //Specifies how often we check for input
int value = 0; //A variable which is modified by commands from the serial connection
```

- In the setup()[36] method we first open the serial connection. This is done with a call to Serial.begin()[37] on the hardware serial object *Serial*. We then use the output method println()[38] to write two strings to the outgoing stream buffer.

```
void setup()
{
  //Open the port
  Serial.begin(9600);
  Serial.println("Opened serial connection");

  //Print initial message
  Serial.println("Enter the commands <add> or <sub>");
}
```

- Here we check if there is any data in the incoming stream buffer. This is done with a call to Serial.available()[39] which returns the number of bytes (characters) which have been received and are currently held in the incoming stream buffer. If the returned value is greater than zero then we call the user defined method *handleInput()* which processes the incoming data. We have also added a small delay by calling the method delay()[40]. This affects how quickly the sketch responds to data received over the serial connection.

[36] http://arduino.cc/en/Reference/Setup
[37] http://arduino.cc/en/Serial/begin
[38] http://arduino.cc/en/Serial/Println
[39] http://arduino.cc/en/Serial/Available
[40] http://arduino.cc/en/reference/delay

Note: An alternative way of handling the data in the incoming stream buffer would be to use the automatically triggered method serialEvent()[41].

```
void loop()
{
  //Check if any data has been received
  //If so call the method handleInput() to process it
  if (Serial.available() > 0)
  {
    handleInput();
  }

  //This delay dictates how often the device will check for input
  delay(INPUT_DELAY);
}
```

This user defined method *handleInput()* is called from loop()[42] if any data is detected in the incoming stream buffer. Here we read the data into the String[43] *input* using the method readString()[44]. This copies the data from the incoming stream buffer and stores it in *input*. It also clears the stream buffer of the data it copied. We then use trim()[45] to remove any additional termination characters which might have be added by the terminal program. Using the String[46] comparison operator ==[47] we compare the received command against the two specified ones and then modify *value* accordingly.

```
void handleInput()
{
  //Get the input string
  String input = Serial.readString();

  //Remove any whitespace or CR/LF
  input.trim();

  //Echo the input
  Serial.println("Command: " + input + " received");

  //Process the input
```

41 http://arduino.cc/en/Reference/SerialEvent
42 http://arduino.cc/en/Reference/loop
43 http://arduino.cc/en/Reference/string
44 http://arduino.cc/en/Reference/StreamReadString
45 http://arduino.cc/en/Reference/StringTrim
46 http://arduino.cc/en/Reference/string
47 http://arduino.cc/en/Reference/StringComparison

```
  if (input == "add")
  {
    Serial.println("Adding 1 to the value");
    value++;
  }
  else if (input == "sub")
  {
    Serial.println("Subtracting 1 from the value");
    value--;
  }
  else
  {
    Serial.println("Unknown command: " + input);
  }

  //Echo the changes
  Serial.print("The current value is: ");
  Serial.println(value);
}
```

4.5. Exercise

Change the example and add a multiplier by 2 and a divider by 2.

```
Serial.println("Enter the commands <add> or <sub> or <div> or <mul>");
```

Also you can change integer value to long float value to have decimals when for example dividing 3/2=1.5.

```
float value = 0;
```

Hint…

```
else if (input == "div")
  {
    Serial.println("Dividing the value by 2");
    value=value/2.0;
  }
  else if (input == "mul")
  {
    Serial.println("Multiplying the value by 2");
    value=value*2.0;
  }
  else
```

```
  {
    Serial.println("Unknown command: " + input);
  }
```

4.6. RTC: setting time, reading time

In this example we will demonstrate how to manually set and read time in RTC from the SODAQ Mbili and display those readings using Serial communication.

Table 5. RTC example

Required Components	SODAQ Mbili Board, 0.5W Solar Panel, 1aH Battery Pack
Required Libraries	Wire, Sodaq_DS3231
Hardware Setup	Connect the SODAQ board to the PC via USB.
Source Code	**RTC_date_time_update.ino**

Turn on the SODAQ Mbili board, compile and upload the following sketch from the Arduino IDE onto the SODAQ Mbili board.

RTC_date_time_update.ino

```
//Include the necessary libraries
#include <Wire.h>
#include <Sodaq_DS3231.h>

Sodaq_DS3231 RTC; //Create RTC object for DS3231
//year, month, date, hour, min, sec and week-day(starts from 0 and goes to 6)
//writing any non-existent time-data may interfere with normal operation of the RTC.
//Take care of week-day also.*/
DateTime dt(2014, 06, 05, 11, 5, 00, 4);

void setup()
{
  //Start serial
  Serial.begin(9600);
  Serial.println("Date,     Time");

  //Start the I2C protocol
  Wire.begin();

  //initialize the DS3231
  RTC.begin();
```

```
  RTC.setDateTime(dt); //Adjust date-time as defined 'dt' above
}

void loop()
{

  String data = getDateTime();
  Serial.println(data);
}

String getDateTime()
{
  String dateTimeStr;

  //Create a DateTime object from the current time
  DateTime dt(RTC.makeDateTime(RTC.now().getEpoch()));

  //Convert it to a String
  dt.addToString(dateTimeStr);

  return dateTimeStr;
}
```

- Here the necessary library files are included in the sketch using the #include[48] compiler directive.

```
//Include the necessary libraries
#include <Wire.h>
#include <Sodaq_DS3231.h>
```

- Here we prepare DateTime format on dt to update RTC.

```
Sodaq_DS3231 RTC; //Create RTC object for DS3231 RTC come Temp Sensor
//year, month, date, hour, min, sec and week-day(starts from 0 and goes to 6)
//writing any non-existent time-data may interfere with normal operation of the RTC.
//Take care of week-day also.
DateTime dt(2014, 06, 05, 11, 5, 00, 4);
```

Note the data format: year, month, date, hour, min, sec and week-day. Week-day starts with 0 (Sunday) and finishes with 6 (Saturday).

[48] http://arduino.cc/en/Reference/Include

- We initialize the Serial, I2C protocol and the DS3231 using calls to *Serial.begin(9600);* , *Wire.begin()* and *rtc.begin()*.

```
void setup()
{
   //Start serial
  Serial.begin(9600);
  Serial.println("Date, Time, Temperature, Voltage, Charging status");

  //Start the I2C protocol
  Wire.begin();

  //initialize the DS3231
  RTC.begin();
  RTC.setDateTime(dt); //Adjust date-time as defined 'dt' above
}
```

Once the RTC has been updated, line *RTC.setDateTime(dt);* has to be commented in order not to update every reboot of the board.

- The readings values are then converted into String mode and glued all together into *data* String and print into serial comm.

```
  String data = getDateTime();
  Serial.println(data);
```

4.7. Reading temperature from internal RTC temperature sensor

Here we will take a temperature reading and date and time reading from the DS3231 chip, display that reading on the Serial comm. The DS3231 chip provides a range of functionality including a real time clock, an interrupt timer, and a temperature sensor. Here we will only look at taking the temperature reading.

In addition to the temperature reading we also read and display the battery voltage level using a reading from one of the analog pins[49] . For further details of the voltage reading refer to the *JST* section of the schema: SODAQ Mbili Schema[50] . We will be also displaying if the battery is being charged or not.

[49] http://arduino.cc/en/Tutorial/AnalogInputPins
[50] http://mbili.sodaq.net/?page_id=20

Connect the solar panel and the LiPo battery into their respective sockets, as per picture below. You don’t need to connect the external sensor yet. Connect the SODAQ board to the PC via USB.

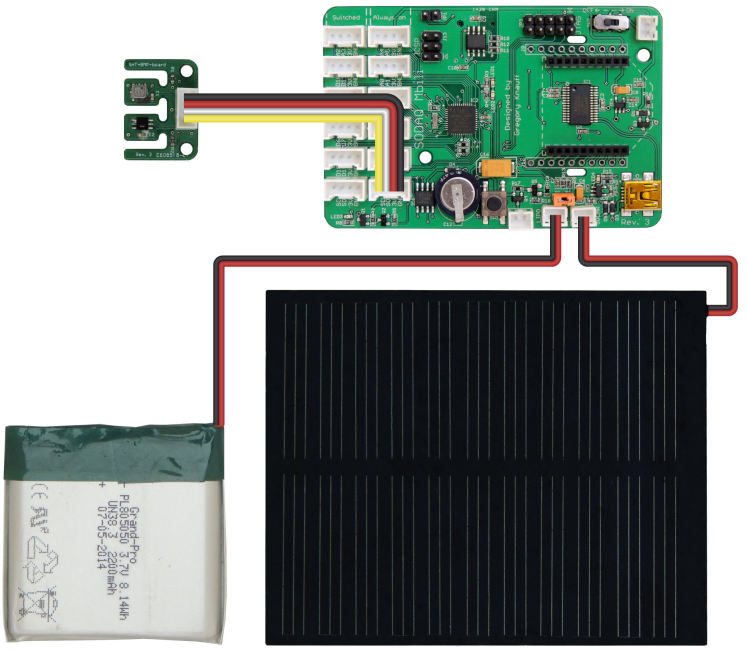


Table 6. RTC temperature example

Required Components	SODAQ Mbili Board, 0.5W Solar Panel, 1aH Battery Pack
Required Libraries	Wire, Sodaq_DS3231
Hardware Setup	Plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets. Connect the SODAQ board to the PC via USB.
Source Code	**RTC_date_Volt_Temp.ino**

Turn on the SODAQ Mbili board, compile and upload the following sketch from the Arduino IDE to the SODAQ Mbili board, and then unplug the USB cable from the computer when it has completed the upload.

RTC_date_Volt_Temp.ino

```
//Include the necessary libraries
#include <Wire.h>
#include <Sodaq_DS3231.h>

//These constants are used for reading the battery voltage
#define ADC_AREF 3.3
#define BATVOLTPIN A6
#define BATVOLT_R1 4.7
#define BATVOLT_R2 10

Sodaq_DS3231 RTC; //Create RTC object for DS3231 RTC come Temp Sensor
char weekDay[][4] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
//year, month, date, hour, min, sec and week-day(starts from 0 and goes to 6)
//writing any non-existent time-data may interfere with normal operation of the RTC.
//Take care of week-day also.*/
DateTime dt(2014, 06, 05, 11, 5, 00, 4);


void setup()
{
  //Start serial
  Serial.begin(9600);
  Serial.println("Date, Time, Temperature, Voltage");

  //Start the I2C protocol
  Wire.begin();

  //initialize the DS3231
  RTC.begin();
  RTC.setDateTime(dt); //Adjust date-time as defined 'dt' above
}

void loop()
{
  ///Read the temperature
  RTC.convertTemperature();
  float temp = RTC.getTemperature();

  // Convert temperature voltage to string
  char buffer[14]; //make buffer large enough for 7 digits
  String temperatureS = dtostrf(temp, 7,2,buffer);
  //'7' digits including '-' negative, decimal and white space. '2' decimal places
  temperatureS.trim(); //trim whitespace, important so Ubidots will treat it as a
 number

  //Read the voltage
```

```
  int mv = getRealBatteryVoltage() * 1000.0;

  String data = getDateTime()+ ", ";
  data += String(temperatureS)+ "C, ";
  data += String(mv)+ "mV");
  Serial.println(data);
}

String getDateTime()
{
  String dateTimeStr;

  //Create a DateTime object from the current time
  DateTime dt(RTC.makeDateTime(RTC.now().getEpoch()));

  //Convert it to a String
  dt.addToString(dateTimeStr);

  return dateTimeStr;
}

float getRealBatteryVoltage()
{
  uint16_t batteryVoltage = analogRead(BATVOLTPIN);
  return (ADC_AREF / 1023.0) * (BATVOLT_R1 + BATVOLT_R2) / BATVOLT_R2 *
 batteryVoltage;
}
```

- Here the necessary library files are included in the sketch using the #include[51] compiler directive.

```
//Include the necessary libraries
#include <Wire.h>
#include <Sodaq_DS3231.h>
```

- Here we define several constants which will be used for reading the voltage level of the battery. *ADC_AREF* specifies the pin voltage and *BATVOLTPIN* the analog pin that the reading is taken from. *BATVOLT_R1* and *BATVOLT_R2* specify the resistors used on the voltage reading circuit. For further details refer to the *JST* section of the schema: SODAQ Mbili Schema[52].

[51] http://arduino.cc/en/Reference/Include
[52] http://mbili.sodaq.net/?page_id=20

```
//These constants are used for reading the battery voltage
#define ADC_AREF 3.3
#define BATVOLTPIN A6
#define BATVOLT_R1 4.7
#define BATVOLT_R2 10
```

- Here we prepare DateTime format on dt to update RTC.

```
Sodaq_DS3231 RTC; //Create RTC object for DS3231 RTC come Temp Sensor
char weekDay[][4] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
//year, month, date, hour, min, sec and week-day(starts from 0 and goes to 6)
//writing any non-existent time-data may interfere with normal operation of the RTC.
//Take care of week-day also.
DateTime dt(2014, 06, 05, 11, 5, 00, 4);
```

- We initialize the Serial, I2C protocol and the DS3231 using calls to *Serial.begin(9600)* , *Wire.begin()* and *rtc.begin()*.

```
void setup()
{
   //Start serial
  Serial.begin(9600);
  Serial.println("Date, Time, Temperature, Voltage, Charging status");

  //Start the I2C protocol
  Wire.begin();

  //initialize the DS3231
  RTC.begin();
  RTC.setDateTime(dt); //Adjust date-time as defined 'dt' above
}
```

Once the RTC has been updated, line *RTC.setDateTime(dt);* has to be commented in order not to update every reboot of the board.

- First we get the temperature reading. The call to *rtc.convertTemperature()* instructs the DS3231 to take a immediate temperature reading. The value of that reading is then retreived with a call to *rtc.getTemperature()*. The user defined method *getRealBatteryVoltage()* is then called and the returned value is converted to millivolts. Then we read the charging status with *read_charge_status()*

The readings values are then converted into String mode and glued all together into *data* String and print into serial comm.

```
  ///Read the temperature
  RTC.convertTemperature();
  float temp = RTC.getTemperature();

  // Convert temperature voltage to string
  char buffer[14]; //make buffer large enough for 7 digits
  String temperatureS = dtostrf(temp, 7,2,buffer);
  //'7' digits including '-' negative, decimal and white space. '2' decimal places
  temperatureS.trim(); //trim whitespace, important so Ubidots will treat it as a
number

  //Read the voltage
  int mv = getRealBatteryVoltage() * 1000.0;

  String data = getDateTime()+ ", ";
  data += String(temperatureS)+ "C, ";
  data += String(mv)+ "mV");
  Serial.println(data);
```

- This method reads an analog signal from the specified analog pin *BATVOLTPIN* and converts it to a voltage reading using the specified constants for the pin voltage *ADC_AREF* and the circuit's resistor values *BATVOLT_R1* & *BATVOLT_R2*. The circuit used here is a voltage divider[53] as the battery voltage is too high to connect directly to the analog pin.

```
float getRealBatteryVoltage()
{
  uint16_t batteryVoltage = analogRead(BATVOLTPIN);
  return (ADC_AREF / 1023.0) * (BATVOLT_R1 + BATVOLT_R2) / BATVOLT_R2 *
 batteryVoltage;
}
```

4.8. Writing to SD card

The SODAQ Mbili has two storage devices available. These are the MicroSD and the Serial Flash devices. In this example we will demonstrate the use of the storage capabilities of the MicroSD device. We will be creating a simple write which, at fixed intervals, logs a sentence to a storage file and to the Serial Monitor. The log file will consists of a sentence in each line

[53] https://learn.sparkfun.com/tutorials/voltage-dividers

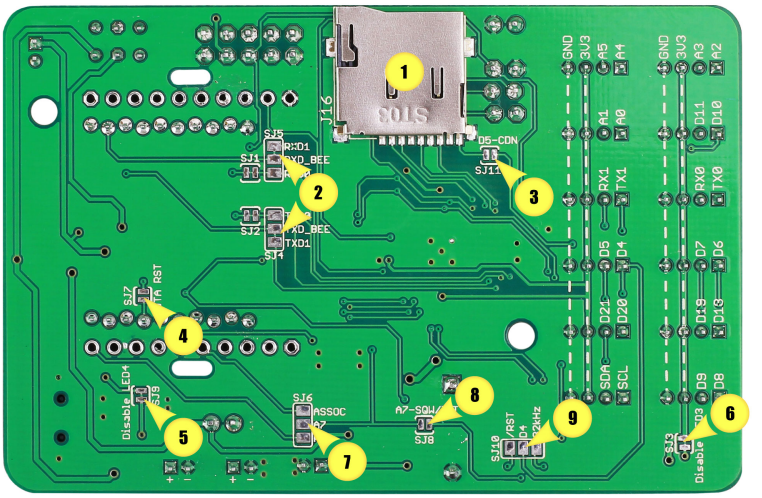
The SD Library[54] provides the functionality for working with the MicroSD device. It contains two classes *SD* and *File*. The *SD* class provides the functionality for working with the MicroSD device and file system, and the *File* class provides the functionality for file operations. For further information about these classes, and the methods they provide, refer to the reference pages available here: SD Library[55].

Table 7. SD write example

Required Components	SODAQ Mbili Board, 0.5W Solar Panel, 1aH Battery Pack, MicroSD card
Required Libraries	SPI, SD
Hardware Setup	Plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets. Insert the MicroSD card. Connect the SODAQ board to the PC via USB.
Source Code	**SD_write.ino**

The *SPI* and *SD* Libraries come pre-installed with the Arduino IDE, and so there is no need to download or install either of them.

Insert the memory card in the socket labelled as "1":



[54] http://arduino.cc/en/Reference/SD
[55] http://arduino.cc/en/Reference/SD

Turn on the SODAQ Mbili board, compile and upload the following sketch from the Arduino IDE onto the SODAQ Mbili board. Leave the USB cable plugged in and open the Serial Monitor (Ctrl-Shift-M) and ensure that it is set to the 9600 baud rate.

SD_write.ino

```
//Include the necessary libraries
#include <SPI.h>
#include <SD.h>

//The delay between the sensor readings
#define READ_DELAY 1000

//Digital pin 11 is the MicroSD slave select pin on the Mbili
#define SD_SS_PIN 11

//The data log file
#define FILE_NAME "DataLog.txt"

//Data header
#define DATA_HEADER "Hello world"

void setup()
{
  //Start serial
  Serial.begin(9600);
  Serial.println(DATA_HEADER);

  //initialize log file
  setupLogFile();
}

void loop()
{
   //Create the data record
  String dataRec = createDataRecord();

  //Save the data record to the log file
  logData(dataRec);

  //Echo the data to the serial connection
  Serial.println(dataRec);

  //Wait before taking the next reading
  delay(READ_DELAY);
}
```

```
void setupLogFile()
{
  //initialize the SD card
  if (!SD.begin(SD_SS_PIN))
  {
    Serial.println("Error: SD card failed to initialize or is missing.");
    //Hang
    while (true);
  }

  //Check if the file already exists
  bool oldFile = SD.exists(FILE_NAME);

  //Open the file in write mode
  File logFile = SD.open(FILE_NAME, FILE_WRITE);

  //Add header information if the file did not already exist
  if (!oldFile)
  {
    logFile.println(DATA_HEADER);
  }

  //Close the file to save it
  logFile.close();
}

void logData(String rec)
{
  //Re-open the file
  File logFile = SD.open(FILE_NAME, FILE_WRITE);

  //Write the CSV data
  logFile.println(rec);

  //Close the file to save it
  logFile.close();
}

String createDataRecord()
{
  //Create a String type data record in csv format

  String data = "1st East-African Workshop on the Internet Of Things";
  return data;
}
```

[56] http://arduino.cc/en/Reference/Include

- Here the necessary library files are included in the sketch using the #include[56] compiler directive.

```
#include <SPI.h>
#include <SD.h>
```

- Here we define the delay between sensor readings, the SPI slave select (SS) pin for the MicroSD device, the log file for the sensor readings. and the data header for the log file.

```
//The delay between the sensor readings
#define READ_DELAY 1000

//Digital pin 11 is the MicroSD slave select pin on the Mbili
#define SD_SS_PIN 11

//The data log file
#define FILE_NAME "DataLog.txt"

//Data header
#define DATA_HEADER "Hello world"
```

- Here we start by initializing the serial connection with a call to Serial.begin()[57]. Next we call to initialize the storage system and log file. We will be displaying the sensor data in the Serial Monitor (as well as saving it to a log file on a MicroSD card). For readability, we first send the header information to the Serial Monitor with a call to the Serial.println()[58] method passing the parameter *DATA_HEADER*.

```
void setup()
{
  //initialize the serial connection
  Serial.begin(9600);

  //initialize log file
  setupLogFile();

  //Echo the data header to the serial connection
  Serial.println(DATA_HEADER);
}
```

[57] http://arduino.cc/en/Serial/Begin
[58] http://arduino.cc/en/Serial/Println

- Here we start by getting the data with a call to the user defined method *createDataRecord()* which returns us a String[59] containing the current sentence. We then pass that String[60] to the user defined method *logData()* which writes it to the log file. We also send the data to the Serial Monitor using the Serial.println()[61] method. Finally, we call delay()[62] to wait *READ_DELAY* number of milliseconds before returning from the *loop()*[63] method. This roughly controls the frequency of the sensor readings.

```
void loop()
{
  //Create the data record
  String dataRec = createDataRecord();

  //Save the data record to the log file
  logData(dataRec);

  //Echo the data to the serial connection
  Serial.println(dataRec);

  //Wait before taking the next reading
  delay(READ_DELAY);
}
```

- Here we start by initializing the MicroSD device. This is done by a call to SD.begin()[64] passing the SPI slave select (SS) pin that the MicroSD device is connected to. On the SODAQ Mbili this is digitial pin 11 (defined here as *SD_SS_PIN*). We then make a call to SD.exists()[65] which tells us whether our log file already exists or not (we will use this information later). We then use thehttp://arduino.cc/en/Reference/SDopen[SD.open()] method with the FILE_WRITE[66] parameter to open the log file in write mode.

When a file is opened in FILE_WRITE[67] mode, if the file already exists, any data written to file will be appended to end of the existing file. If the file does not exist it will be created.

59 http://arduino.cc/en/Reference/string
60 http://arduino.cc/en/Reference/string
61 http://arduino.cc/en/Serial/Println
62 http://arduino.cc/en/reference/delay
63 http://arduino.cc/en/Reference/loop
64 http://arduino.cc/en/Reference/SDbegin
65 http://arduino.cc/en/Reference/SDexists
66 http://arduino.cc/en/Reference/SDopen
67 http://arduino.cc/en/Reference/SDopen

If the file did not previously exist, we write the header information as the first line of that file. This is done with a call to File.println()[68] using the constant *DATA_HEADER*. This data header gives a description of what sensor data is being written to the log file and the order of that data.

```
void setupLogFile()
{
  //initialize the SD card
  if (!SD.begin(SD_SS_PIN))
  {
    Serial.println("Error: SD card failed to initialize or is missing.");
    //Hang
    while (true);
  }

  //Check if the file already exists
  bool oldFile = SD.exists(FILE_NAME);

  //Open the file in write mode
  File logFile = SD.open(FILE_NAME, FILE_WRITE);

  //Add header information if the file did not already exist
  if (!oldFile)
  {
    logFile.println(DATA_HEADER);
  }

  //Close the file to save it
  logFile.close();
}
```

- Here we reopen the log file using the SD.open()[69] method and FILE_WRITE[70] parameter. Since the file already exists any new data will be appended to the existing file. Using the File.println()[71] method we write a new line to log file using the String[72] passed as the parameter *rec*. Finally, we close the file to ensure that the data is saved.

```
void logData(String rec)
{
  //Re-open the file
```

[68] http://arduino.cc/en/Reference/FilePrintln
[69] http://arduino.cc/en/Reference/SDopen
[70] http://arduino.cc/en/Reference/SDopen
[71] http://arduino.cc/en/Reference/FilePrintln
[72] http://arduino.cc/en/Reference/string

```
  logFile = SD.open(FILE_NAME, FILE_WRITE);

  //Write the CSV data
  logFile.println(rec);

  //Close the file to save it
  logFile.close();
}
```

- Here we create and return a String[73] which contains the data readings from the sensors in comma separated format (CSV). The String[74] contains the Time & Date which is queried using the user defined method *getTimeDate()*. We then append the sensor data, using the String += operator[75], from each of the four sensors with a comma separator between each reading. The complete String[76] containing all the data is then returned from this method.

```
String createDataRecord()
{
  //Create a String type data record in csv format

  String data = "1st East-African Workshop on the Internet Of Things";

  return data;
}
```

4.9. Writing Date, Voltage and Temperature to the SD card

In this example we will demonstrate the use of RTC to read date, time and temperature. We will also measure battery voltage and will be creating a simple data logger that logs this information at fixed intervals in a file and shows it in the Serial Monitor. The log file will consists of Comma Separated Values (CSV) in ASCII format.

Table 8. RTC and SD write example

Required Components	SODAQ Mbili Board, 0.5W Solar Panel, 1aH Battery Pack, MicroSD card
Required Libraries	SPI, SD, Sodaq_DS3231
Source Code	**RTC_date_Volt_Temp_SD.ino**

[73] http://arduino.cc/en/Reference/string
[74] http://arduino.cc/en/Reference/string
[75] http://arduino.cc/en/Reference/StringAppend
[76] http://arduino.cc/en/Reference/string

Hardware Setup	Plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets. Insert the MicroSD card. Connect the SODAQ board to the PC via USB.
Source Code	**RTC_date_Volt_Temp_SD.ino**

The *SPI* and *SD* Libraries come pre-installed with the Arduino IDE, and so there is no need to download or install either of them. The *Sodaq_DS3231* libraries is included with the SODAQ Mbili files that you have already installed.

Turn on the SODAQ Mbili board, compile and upload the following sketch from the Arduino IDE onto the SODAQ Mbili board. Leave the USB cable plugged in and open the Serial Monitor (Ctrl-Shift-M) and ensure that it is set to the 9600 baud rate.

RTC_date_Volt_Temp_SD.ino

```
//Include the necessary libraries
#include <Wire.h>
#include <Sodaq_DS3231.h>
#include <SPI.h>
#include <SD.h>

//These constants are used for reading the battery voltage
#define ADC_AREF 3.3
#define BATVOLTPIN A6
#define BATVOLT_R1 4.7
#define BATVOLT_R2 10

//The delay between the sensor readings
#define READ_DELAY 1000

//Digital pin 11 is the MicroSD slave select pin on the Mbili
#define SD_SS_PIN 11

//The data log file
#define FILE_NAME "DataLog.txt"

//Data header
#define DATA_HEADER "DateTime, Temperature, Voltage"

Sodaq_DS3231 RTC; //Create RTC object for DS3231 RTC come Temp Sensor
//year, month, date, hour, min, sec and week-day(starts from 0 and goes to 6)
//writing any non-existent time-data may interfere with normal operation of the RTC.
//Take care of week-day also.*/
DateTime dt(2014, 06, 05, 11, 5, 00, 4);
```

```
void setup()
{
  //Start serial
  Serial.begin(9600);
  Serial.println(DATA_HEADER);

  //initialize sensors
  setupSensors();

  //initialize log file
  setupLogFile();
}

void loop()
{
   //Create the data record
  String dataRec = createDataRecord();

  //Save the data record to the log file
  logData(dataRec);

  //Echo the data to the serial connection
  Serial.println(dataRec);

  //Wait before taking the next reading
  delay(READ_DELAY);
}

void setupSensors()
{
  //initialize I2C
  Wire.begin();

  //initialize the DS3231
  RTC.begin();

  //remember to comment this line once RTC is updated
  RTC.setDateTime(dt); //Adjust date-time as defined 'dt' above

}

void setupLogFile()
{
  //initialize the SD card
  if (!SD.begin(SD_SS_PIN))
  {
```

```
    Serial.println("Error: SD card failed to initialize or is missing.");
    //Hang
    while (true);
  }

  //Check if the file already exists
  bool oldFile = SD.exists(FILE_NAME);

  //Open the file in write mode
  File logFile = SD.open(FILE_NAME, FILE_WRITE);

  //Add header information if the file did not already exist
  if (!oldFile)
  {
    logFile.println(DATA_HEADER);
  }

  //Close the file to save it
  logFile.close();
}

void logData(String rec)
{
  //Re-open the file
  File logFile = SD.open(FILE_NAME, FILE_WRITE);

  //Write the CSV data
  logFile.println(rec);

  //Close the file to save it
  logFile.close();
}

String createDataRecord()
{
  //Create a String type data record in csv format
  ///Read the temperature
  RTC.convertTemperature();
  float temp = RTC.getTemperature();

  // Convert temperature voltage to string
  char buffer[14]; //make buffer large enough for 7 digits
  String temperatureS = dtostrf(temp, 7,2,buffer);
  //'7' digits including '-' negative, decimal and white space. '2' decimal places
  temperatureS.trim(); //trim whitespace, important so Ubidots will treat it as a
 number
```

```
  //Read the voltage
  int mv = getRealBatteryVoltage() * 1000.0;

  String data = getDateTime()+ ", ";
  data += String(temperatureS)+ "C, ";
  data += String(mv)+ "mV";

  return data;
}

String getDateTime()
{
  String dateTimeStr;

  //Create a DateTime object from the current time
  DateTime dt(RTC.makeDateTime(RTC.now().getEpoch()));

  //Convert it to a String
  dt.addToString(dateTimeStr);

  return dateTimeStr;
}

float getRealBatteryVoltage()
{
  uint16_t batteryVoltage = analogRead(BATVOLTPIN);
  return (ADC_AREF / 1023.0) * (BATVOLT_R1 + BATVOLT_R2) / BATVOLT_R2 *
 batteryVoltage;
}
```

- Here the necessary library files are included in the sketch using the #include[77] compiler directive.

```
#include <Wire.h>
#include <SPI.h>
#include <SD.h>

//SODAQ Mbili libraries
#include <Sodaq_DS3231.h>
```

- Here we define the delay between sensor readings, the SPI slave select (SS) pin for the MicroSD device, the log file for the sensor readings and the data header for the log file. We also define constants are used for reading the battery voltage.

[77] http://arduino.cc/en/Reference/Include

```
//These constants are used for reading the battery voltage
#define ADC_AREF 3.3
#define BATVOLTPIN A6
#define BATVOLT_R1 4.7
#define BATVOLT_R2 10

//The delay between the sensor readings
#define READ_DELAY 1000

//Digital pin 11 is the MicroSD slave select pin on the Mbili
#define SD_SS_PIN 11

//The data log file
#define FILE_NAME "DataLog.txt"

//Data header
#define DATA_HEADER "DateTime, Temperature, Voltage"
```

- Here we start by initializing the serial connection with a call to Serial.begin()[78]. Next we call several user defined methods, one to initialize the readings we will be using and the other to initialize the storage system and log file. We will be displaying the readings data in the Serial Monitor (as well as saving it to a log file on a MicroSD card). For readability, we first send the header information to the Serial Monitor with a call to the Serial.println()[79] method passing the parameter *DATA_HEADER*.

```
void setup()
{
  //initialize the serial connection
  Serial.begin(9600);

  //initialize sensors
  setupSensors();

  //initialize log file
  setupLogFile();

  //Echo the data header to the serial connection
  Serial.println(DATA_HEADER);
}
```

[78] http://arduino.cc/en/Serial/Begin
[79] http://arduino.cc/en/Serial/Println

- Here we start by getting the readings with a call to the user defined method *createDataRecord()* which returns us a String[80] containing the current readings. We then pass that String[81] to the user defined method *logData()* which writes it to the log file. We also send the data to the Serial Monitor using the Serial.println()[82] method. Finally, we call delay()[83] to wait *READ_DELAY* number of milliseconds before returning from the *loop()*[84] method. This roughly controls the frequency of the readings.

```
void loop()
{
  //Create the data record
  String dataRec = createDataRecord();

  //Save the data record to the log file
  logData(dataRec);

  //Echo the data to the serial connection
  Serial.println(dataRec);

  //Wait before taking the next reading
  delay(READ_DELAY);
}
```

- Here we make the necessary calls in order to setup the readingssensors we will be using. We initialize the Real Time Clock (RTC) on the DS3231 chip with a call to *RTC.begin()*.

> The SHT2x does not require initialisation.

```
void setupSensors()
{
  //initialize the wire protocol for the TPH sensors
  Wire.begin();

  //initialize the DS3231 RTC
  RTC.begin();
}
```

80 http://arduino.cc/en/Reference/string
81 http://arduino.cc/en/Reference/string
82 http://arduino.cc/en/Serial/Println
83 http://arduino.cc/en/reference/delay
84 http://arduino.cc/en/Reference/loop

- Here we start by initialising the MicroSD device. This is done by a call to SD.begin()[85] passing the SPI slave select (SS) pin that the MicroSD device is connected to. On the SODAQ Mbili this is digitial pin 11 (defined here as *SD_SS_PIN*). We then make a call to SD.exists()[86] which tells us whether our log file already exists or not (we will use this information later). We then use the SD.open()[87] method with the FILE_WRITE[88] parameter to open the log file in write mode.

When a file is opened in FILE_WRITE[89] mode, if the file already exists, any data written to file will be appended to end of the existing file. If the file does not exist it will be created.

If the file did not previously exist, we write the header information as the first line of that file. This is done with a call to File.println()[90] using the constant *DATA_HEADER*. This data header gives a description of what sensor data is being written to the log file and the order of that data.

```
void setupLogFile()
{
  //initialize the SD card
  if (!SD.begin(SD_SS_PIN))
  {
    Serial.println("Error: SD card failed to initialize or is missing.");
    //Hang
    while (true);
  }

  //Check if the file already exists
  bool oldFile = SD.exists(FILE_NAME);

  //Open the file in write mode
  File logFile = SD.open(FILE_NAME, FILE_WRITE);

  //Add header information if the file did not already exist
  if (!oldFile)
  {
    logFile.println(DATA_HEADER);
  }
```

85 http://arduino.cc/en/Reference/SDbegin
86 http://arduino.cc/en/Reference/SDexists
87 http://arduino.cc/en/Reference/SDopen
88 http://arduino.cc/en/Reference/SDopen
89 http://arduino.cc/en/Reference/SDopen
90 http://arduino.cc/en/Reference/FilePrintln

```
  //Close the file to save it
  logFile.close();
}
```

- Here we reopen the log file using the SD.open()[91] method and FILE_WRITE[92] parameter. Since the file already exists any new data will be appended to the existing file. Using the File.println()[93] method we write a new line to log file using the String[94] passed as the parameter *rec*. Finally, we close the file to ensure that the data is saved.

```
void logData(String rec)
{
  //Re-open the file
  logFile = SD.open(FILE_NAME, FILE_WRITE);

  //Write the CSV data
  logFile.println(rec);

  //Close the file to save it
  logFile.close();
}
```

- Here we create and return a String[95] which contains the data readings from the sensors in comma separated format (CSV). The String[96] contains the Time & Date which is queried using the user defined method *getTimeDate()*. We then append the readings. The complete String[97] containing all the data is then returned from this method.

```
String createDataRecord()
{
  //Create a String type data record in csv format
  ///Read the temperature
  RTC.convertTemperature();
  float temp = RTC.getTemperature();

  // Convert temperature voltage to string
  char buffer[14]; //make buffer large enough for 7 digits
```

91 http://arduino.cc/en/Reference/SDopen
92 http://arduino.cc/en/Reference/SDopen
93 http://arduino.cc/en/Reference/FilePrintln
94 http://arduino.cc/en/Reference/string
95 http://arduino.cc/en/Reference/string
96 http://arduino.cc/en/Reference/string
97 http://arduino.cc/en/Reference/string

```
  String temperatureS = dtostrf(temp, 7,2,buffer);
  //'7' digits including '-' negative, decimal and white space. '2' decimal places
  temperatureS.trim(); //trim whitespace, important so Ubidots will treat it as a
 number

  //Read the voltage
  int mv = getRealBatteryVoltage() * 1000.0;

  String data = getDateTime()+ ", ";
  data += String(temperatureS)+ "C, ";
  data += String(mv)+ "mV";

  return data;
}
```

- Here we return a Date and Time reading from the RTC in a String[98] format. First a DateTime[99] object is constructed using a time reading from the DS3231 RTC. This is then converted into a String[100] using the method *DateTime.addToString()* the result of which is then returned from this method.

```
String getDateTime()
{
  String dateTimeStr;

  //Create a DateTime object from the current time
  DateTime dt(rtc.makeDateTime(rtc.now().getEpoch()));

  //Convert it to a String
  dt.addToString(dateTimeStr);

  return dateTimeStr;
}
```

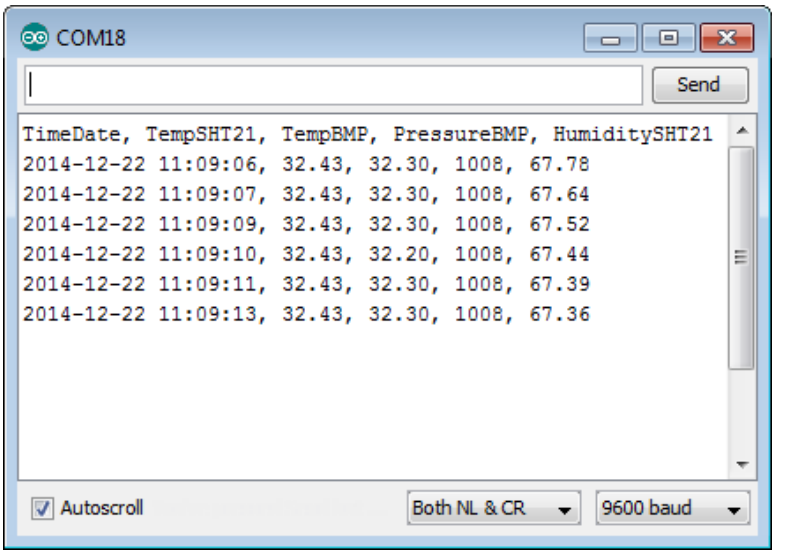
4.10. Adding a Timer to Schedule Readings

This example builds on the previous example. Only the sections of code which have been changed or added are discussed here. However, a full copy of the complete sketch can be found at the end of this page.

98 http://arduino.cc/en/Reference/string
99 http://playground.arduino.cc/Code/DateTime
100 http://arduino.cc/en/Reference/string

In this example we will demonstrate the use of a *RTCTimer* to schedule regular events. This example builds on the previous example, but instead of using the delay()[101] method, it uses a scheduling timer to control the frequency of the readings.

If you examine the output from the previous example you will see that the actual sensor readings are not exactly at one second intervals. This is due to the use of the method delay()[102] which pauses the execution for a specified number of milliseconds. However, since the actual time between readings is the sum of the execution time and the specified delay, this results in readings which in this case are taken at intervals of longer than one second.



The *RTCTimer* library provides the functionality for scheduling specific methods at fixed intervals. Here we will modify the previous example to take the sensor readings using a method which is called at fixed intervals by a *RTCTimer* object.

The required *RTCTimer* library is included with the SODAQ Mbili files that you have already installed.

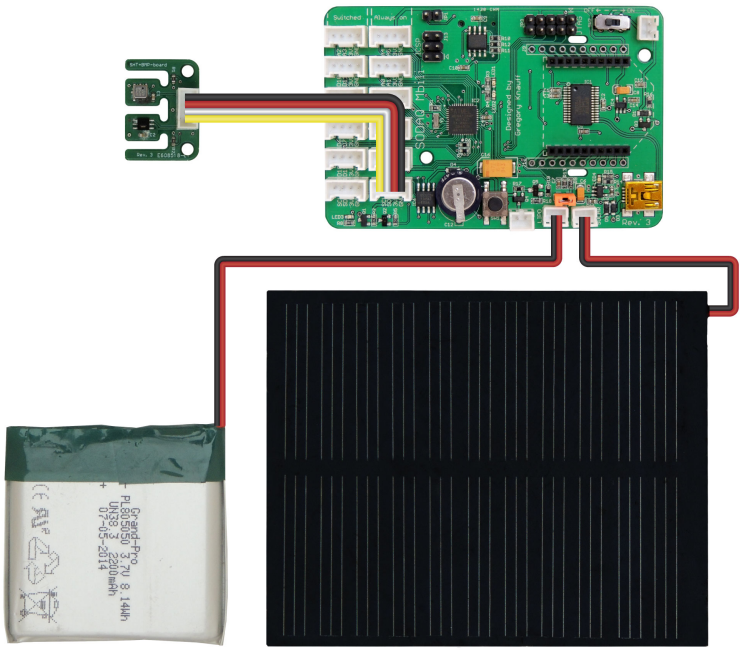
If necessary, refer to Section 2[103] of the Getting Started[104] guide for details on where to download from and how to install the SODAQ Mbili files.

Instead of using the temperature from the RTC, we will use an external sensor that provides temperature, pressure and humidity. We will refer to this sensor as TPH sensor.

101 http://arduino.cc/en/reference/delay
102 http://arduino.cc/en/reference/delay
103 http://mbili.sodaq.net/?page_id=23#step2
104 http://mbili.sodaq.net/?page_id=23

You should refer to both the board diagram[105] and Grove sockets page[106] for additional information.

1. First, plug the TPH Sensor into the Grove I^{2}C socket.
2. Then, plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets.



Turn on the SODAQ Mbili board, compile and upload the following sketch from the Arduino IDE onto the SODAQ Mbili board. Leave the USB cable plugged in and open the Serial Monitor (Ctrl-Shift-M) and ensure that it is set to the 9600 baud rate.

- In addition to the existing libraries, we must now also include the *RTCTimer* library in the sketch using the #include[107] compiler directive.

```
#include <RTCTimer.h>
```

- In addition to the existing *Globals* we declare a *RTCTimer* object.

105 http://mbili.sodaq.net/?page_id=13
106 http://mbili.sodaq.net/?page_id=81
107 http://arduino.cc/en/Reference/Include

```
//RTC Timer
RTCTimer timer;
```

- In addition to the existing setup code, we make a call to the user defined method *setupTimer()* which handles the initialisation of the timer and the scheduling of the main datalogging method *takeReading()*. We also make a call to *takeReading()* to take first sensor reading immediately instead of waiting for the first scheduled reading.

```
//Setup timer events
setupTimer();

//Echo the data header to the serial connection
Serial.println(DATA_HEADER);

//Take first reading immediately
takeReading(getNow())
```

- The main functionality of this sketch is handled by the scheduled callback method *takeReading(),* which is called automatically at a scheduled interval. Here in the loop()[108] method all we do is make a call to *RTCTimer.update()* which updates the *timer* object and ensures any scheduled events are called if they are now due.

```
void loop()
{
  //Update the timer
  timer.update();
}
```

- This method is scheduled to be called automatically by the *timer* object. It is scheduled to be called every *READ_DELAY* number of ticks. In this particular setup each tick is equal to one millisecond. Here we start by getting the sensor readings with a call to the user defined method *createDataRecord()* which returns us a String[109] containing the current sensor readings. We then pass that String[110] to the user defined method *logData()* which writes it to the log file. We also send the data to the Serial Monitor using the Serial.println()[111] method.

[108] http://arduino.cc/en/Reference/loop
[109] http://arduino.cc/en/Reference/string
[110] http://arduino.cc/en/Reference/string
[111] http://arduino.cc/en/Serial/Println

```
void takeReading(uint32_t ts)
{
  //Create the data record
  String dataRec = createDataRecord();

  //Save the data record to the log file
  logData(dataRec);

  //Echo the data to the serial connection
  Serial.println(dataRec);
}
```

- Here we schedule the *takeReading()* method to be called every *READ_DELAY* number of ticks. This is done by a call to the RTCTimer.every() method where the first parameter specifies the number of ticks between each call to the method specified by the second parameter.

```
void setupTimer()
{
  //Instruct the RTCTimer how to get the current time reading
  timer.setNowCallback(getNow);

  //Schedule the reading every second
  timer.every(READ_DELAY, takeReading);
}
```

- The scheduled callback methods must be declared in the following format, where *label* is replaced with the actual name of the method:

```
void label(uint32_t ts)
```

You must specify a separate callback method that *RTCTimer* will use to determine the current time reading (you should do this before scheduling any callbacks). This method must be declared in the following format, where *label* is replaced with the actual name of the method:

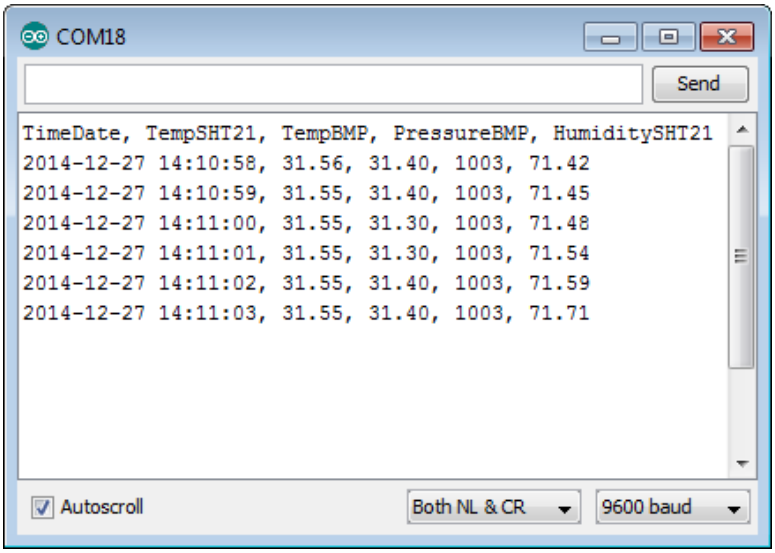
```
uint32_t __label__()
```

[112] http://arduino.cc/en/reference/millis

- This method is called by the *RTCTimer* object to update its current time reading. Here we use the method millis()[112] which returns the number of milliseconds which have elapsed since the current sketch began. In this example one tick or one increment to the time value is equal to one millisecond. It is also possible to use one second intervals by using readings from the Real Time Clock. Just remember that the scheduled callbacks will be called after the specified number of ticks have elapsed regardless of what unit of time that equates to.

```
uint32_t getNow()
{
  return millis();
}
```

If you open the Serial Monitor (Ctrl-Shift-M), you should see output similar to this. Notice that the sensor readings are now at regular one second intervals:



The complete source code is the following:

RTC_TPH_SD_Timer.ino

```
#include <Wire.h>
#include <SPI.h>
#include <SD.h>

//SODAQ Mbili libraries
#include <RTCTimer.h>
#include <Sodaq_BMP085.h>
#include <Sodaq_SHT2x.h>
```

```
#include <Sodaq_DS3231.h>

//The delay between the sensor readings
#define READ_DELAY 1000

//Digital pin 11 is the MicroSD slave select pin on the Mbili
#define SD_SS_PIN 11

//The data log file
#define FILE_NAME "DataLog.txt"

//Data header
#define DATA_HEADER "TimeDate, TempSHT21, TempBMP, PressureBMP, HumiditySHT21"

//TPH BMP sensor
Sodaq_BMP085 bmp;

//RTC Timer
RTCTimer timer;

void setup()
{
  //initialize the serial connection
  Serial.begin(9600);

  //initialize sensors
  setupSensors();

  //initialize log file
  setupLogFile();

  //Setup timer events
  setupTimer();

  //Echo the data header to the serial connection
  Serial.println(DATA_HEADER);

  //Take first reading immediately
  takeReading(getNow());
}

void loop()
{
  //Update the timer
  timer.update();
}
```

```
void takeReading(uint32_t ts)
{
  //Create the data record
  String dataRec = createDataRecord();

  //Save the data record to the log file
  logData(dataRec);

  //Echo the data to the serial connection
  Serial.println(dataRec);
}

void setupSensors()
{
  //initialize the wire protocol for the TPH sensors
  Wire.begin();

  //initialize the TPH BMP sensor
  bmp.begin();

  //initialize the DS3231 RTC
  rtc.begin();
}

void setupLogFile()
{
  //initialize the SD card
  if (!SD.begin(SD_SS_PIN))
  {
    Serial.println("Error: SD card failed to initialize or is missing.");
    //Hang
    while (true);
  }

  //Check if the file already exists
  bool oldFile = SD.exists(FILE_NAME);

  //Open the file in write mode
  File logFile = SD.open(FILE_NAME, FILE_WRITE);

  //Add header information if the file did not already exist
  if (!oldFile)
  {
    logFile.println(DATA_HEADER);
  }

  //Close the file to save it
```

```
  logFile.close();
}

void setupTimer()
{
  //Instruct the RTCTimer how to get the current time reading
  timer.setNowCallback(getNow);

  //Schedule the reading every second
  timer.every(READ_DELAY, takeReading);
}

void logData(String rec)
{
  //Re-open the file
  File logFile = SD.open(FILE_NAME, FILE_WRITE);

  //Write the CSV data
  logFile.println(rec);

  //Close the file to save it
  logFile.close();
}

String createDataRecord()
{
  //Create a String type data record in csv format
  //TimeDate, TempSHT21, TempBMP, PressureBMP, HumiditySHT21
  String data = getDateTime() + ", ";
  data += String(SHT2x.GetTemperature())  + ", ";
  data += String(bmp.readTemperature()) + ", ";
  data += String(bmp.readPressure() / 100)  + ", ";
  data += String(SHT2x.GetHumidity());

  return data;
}

String getDateTime()
{
  String dateTimeStr;

  //Create a DateTime object from the current time
  DateTime dt(rtc.makeDateTime(rtc.now().getEpoch()));

  //Convert it to a String
  dt.addToString(dateTimeStr);
```

```
  return dateTimeStr;
}

uint32_t getNow()
{
  return millis();
}
```