

---

# SETUP MANUAL FOR ARDUINO DATA LOGGER

---

Version 1.0

AUGUST 18, 2022  
Callum Anderson

## Table of Contents

Table of Figures .....	ii
1.0 Introduction.....	1
1.1 Hardware for Arduino.....	1
1.2 Software for Arduino .....	1
1.3 SDI-12 Sensors .....	1
1.4 Telemetry .....	2
1.5 Setup Tutorial.....	2
2.0 Data logger Component .....	3
2.1 Data Logger Components List .....	3
2.2 Software .....	4
3.0 Assembling the Data logger.....	6
3.1 Pinout for Components .....	6
3.2 Connecting SDI-12 Sensors.....	6
4.0 Data logger Setup.....	8
4.1 Setting up Arduino IDE .....	8
4.2 Testing the SD Card Module .....	9
4.3 Setting up DS3231 Real Time Clock.....	11
4.4 Setting up the SDI-12 Data Bus.....	11
5.0 Data logger Sketch.....	14
5.1 Initializing the Setup Parameters .....	14
5.2 Adding SDI-12 Measurements .....	15
5.3 Adding Analog Sensors .....	17
5.4 Checking the Datalogging Sketch.....	19
6.0 Retrieving Data .....	20
6.1 Setting up a Modem.....	20
6.2 Connecting to Putty.....	20
References.....	22
Appendix.....	23
SDI-12 Commands [8].....	23
Physical Builds.....	25

## Table of Figures

Figure 1 Pinout diagram [4,5,6,7].	6
Figure 2 5V SDI-12 Sensor connection [4].	7
Figure 3 12V SDI-12 Sensor connection [4].	7
Figure 4 Arduino IDE settings.	8
Figure 5 Verify/Upload buttons (white when selected).	8
Figure 6 Verify/Upload shortcuts.	8
Figure 7 Format selection on MiniTool Partition Wizard.	9
Figure 8 Partition formatting.	9
Figure 9 Apply the formatting to the SD card	10
Figure 10 Parameters for the SD card test.	10
Figure 11 Successful SD card test.	10
Figure 12 Time parameters.	11
Figure 13 Successful DS3231 setup.	11
Figure 14 Assigning serial baud rate and a data pin.	12
Figure 15 Output of sensor address change.	12
Figure 16 Reassigning the sensor's address to 1.	12
Figure 17 Sensor is now at address 1.	12
Figure 18 SDI address labeled on the sensor	13
Figure 19 Code that will reset data in files.	14
Figure 20 Serial baud rate definitions.	14
Figure 21 SDI data bus pin.	14
Figure 22 CS pin on SD card is connected to Arduino.	14
Figure 23 Filename definition.	14
Figure 24 Minute timestamp definition.	15
Figure 25 Definition of sensor address.	15
Figure 26 Defining multiple addresses.	15
Figure 27 Floats of variables.	15
Figure 28 Measurement function.	16
Figure 29 Two measurements were taken from the PT12 sensor.	16
Figure 30 Adding variables to the data string.	16
Figure 31 Definitions of different parameters.	16
Figure 32 Taking different measurements.	17
Figure 33 Updated data string.	17
Figure 34 Parameters copied into next section	17
Figure 35 Serial monitor printed text.	17
Figure 36 Initializing analog pins for sensors.	18
Figure 37 Initializing float for measurement variable.	18
Figure 38 Adding sensors to the data.	18
Figure 39 Live reading from the serial monitor.	19
Figure 40 Data from SD card on the serial monitor.	19
Figure 41 Creating a new Solar-Putty session.	20

Figure 42 Readings on Solar-PuTTY. ....	21
---	----

## 1.0 Introduction

### 1.1 Hardware for Arduino

The Arduino is a microcontroller that can use and run “sketches”. A microcontroller is a small integrated circuit that uses CPU and onboard RAM to take inputs and create outputs from uploaded programs. Sketches are the programs created in the Arduino IDE that can take inputs and create desired outputs [1]. They have an onboard memory that stores the sketch and when connected to power will continuously execute the onboard sketch [1].

The device that will be used for the data logger is the Arduino Uno R3. This device has an operating voltage of 5V that is connected via USB-A [2]. This board has 14 Analog GPIO Pins and 18 Digital GPIO pins that allow for the integration of modules, sensors, and shields [2]. Shields are pre-designed boards that allow for easy expansion of the Uno’s functionality. For this project, we will be using an RS232 module, a DS3231 real-time clock module, and an SD card module. The RS232 allows for the Arduino to connect to devices via an RS232 serial cable. The RTC is used to keep track of time with an external battery in case of a power loss. Adding the SD card module will allow for the Arduino to log and store data and can use commands to get the data off of the Arduino.

These devices are easy to work with and have many informative videos, forums, and web information.

The Arduino website can be found here: [Arduino - Home](#)

The Arduino forum can be found here: [Arduino Forum](#)

### 1.2 Software for Arduino

Arduino IDE is open-source software that is used to program sketches for Arduino microcontrollers. The IDE can be downloaded or used in the Arduino web editor, linked below. Note that the Arduino can only hold and read one sketch at a time, which is run on a perpetual loop.

Download Arduino IDE: [Software | Arduino](#)

Arduino Web Editor: [Arduino SSO](#)

To get introduced to the Arduino IDE please watch:

[Arduino Tutorial 1: Setting Up and Programming the Arduino for Absolute Beginners](#)

This video walks through downloading, connecting, and programming an Arduino.

### 1.3 SDI-12 Sensors

Serial Data Interface at 1200 baud or SDI-12 is common amongst many environmental sensors and data loggers. These sensors use a simple 3-wire interface that includes; VCC (positive input voltage), GROUND, and DATA [3]. They work by being assigned to a specific sensor address on a data bus. The data logger can then connect to the bus and look for measurements on specific sensor addresses. This allows for the simple integration of multiple sensors on the same data bus as long as each sensor has its specific address.

SDI-12 sensor interacts with data loggers and other devices using their specific language of commands. These commands can be used to determine the type of sensor attached to a data bus, what that sensor's address is, reassign the sensor to a new address, and take measurements from the sensor. All these commands are listed under [Appendix: SDI-12 Commands](#).

## 1.4 Telemetry

The data logger uses an RS232 connection to connect to an external modem. This allows for a remote connection to the Arduino. Using a terminal emulator like Solar-PuTTY and connecting to the modem that is connected to the Arduino allows for a user interface. This can **only** be used to send pre-programmed commands and view the outputs on the Arduino. Solar-PuTTY can be downloaded at [Solar-PuTTY for Windows – Free SSH Download | SolarWinds](#)

## 1.5 Setup Tutorial

The setup tutorial video can be found here:

[SDI-12 Data Logger Using Arduino - Setup Tutorial - YouTube](#)

## 2.0 Data logger Component

For the data logger to get data from the SDI sensors into the WISKI database, it must be able to connect and interact with the SDI sensors, take measurements at the desired time interval, store the data, and then use telemetry to dump the data onto a platform that can interact with WISKI.

### 2.1 Data Logger Components List

Note that the links are just suggested purchases, and these are the components that are required to build and deploy the data logger in the field complete with remote connection and independent battery. If this is only being connected in area where there can be direct connection to power and to connect for data component 10 to 19 are not required. Please see [Appendix – Physical Builds](#) for diagrams containing the components.

Number	Component	Description	Amazon Link
1	Arduino Uno	The microcontroller is the main component of the data logger.	<a href="#">ELEGOO UNO R3 Board</a>
2	USB Cable (can come with board)	Used to connect the Arduino to a computer or external power.	<a href="#">USB A Male to B Male High</a>
3	SD Card Module	Connects an SD card to the Arduino to enable data storage.	<a href="#">Micro SD Card Micro Adapter</a>
4	Micro SD Card	Inserted into the SD card module to store data.	<a href="#">Micro SD Flash Memory Card 64GB - 5 Pack</a>
5	Screw Shield	Connects to Arduino to allow for more secure connections.	<a href="#">Gikfun Screw Shield Expansion</a>
6	Breadboard	Allows for easy external connections. Good to have a data logger with multiple sensors.	<a href="#">ELEGOO 6PCS 170 tie-Points Mini Breadboard kit</a>
7	Breadboard Jumper Wires	Connects the Arduino to the components.	<a href="#">Jumper Wires Solderless Breadboard Jumper Wires</a>
8	Waterproof Case	Case for Arduino to ensure it is not exposed to water. Maybe a good idea to have desiccants inside.	<a href="#">Pelican 1040 Micro Case (Aqua/Clear)</a>
9	Sensors and Manuals	These will be the sensors used to take measurements. The manuals are required for slopes and wiring diagrams.	N/A

10	12V to 5V converter	Converts 12V coming off of a battery to 5V to help protect the Arduino from power overload.	<a href="#">DC to DC Step Down Converter Module 12V/24V to 5V 3A</a>
11	DS3231 Module (Real Time Clock)	Arduino can keep the correct time even if disconnected from power.	<a href="#">Wishiot DS3231 Real Time Clock Module High Precision</a>
12	CR2032 3V Battery	Externally powers the DS3231 to keep time without direct power.	<a href="#">LiCB CR2032 3v Lithium Batteries</a>
13	RS232 to TTL Converter	Enables an easy connection from the Arduino to an external modem.	<a href="#">Serial Port Converter Module RS232 to TTL Female</a>
14	Serial Cable	Used to connect the data logger to the mode. (Male-Male).	<a href="#">RS232 Serial Male to Male</a>
15	Microhard Modem	This must be set up and able to connect to the cellular network.	<a href="#">Microhard Systems Bullet-LTE</a>
16	DC35-12 Battery	It powers the data logger and is charged by the solar panel.	<a href="#">DC35-12A 12V Battery</a>
17	Wire & Connections	Used for many connections to the battery, data logger, sensors, and SunSaver.	<a href="#">14 Gauge AWG Wire True Spec</a>
18	Solar Panel	Will charge the 12 V battery.	<a href="#">10 Watt 12V Solar Panel Kit</a>
19	Solar Charge Controller	Controls the voltage and current charging the battery and being used on the loads (data logger, sensors).	<a href="#">30A Solar Charge Controller 12V/24V</a>

## 2.2 Software

The best way to implement the sketches onto the Arduino is by using the Arduino IDE. This can be downloaded here: [Software | Arduino](#)

The Arduino Web Editor can also be used: [Arduino SSO](#)

The sketches that are used to set up the Arduino data logger are found at:

[Summer 2022 - MVCA | Callum Anderson \(anderc478.wixsite.com\)](#)

[Anderc47/Data-Logger---Arduino \(github.com\)](#)

The purpose of the sketches is as follows:



Name	Purpose
Datalogger	Main data logger program that will tell the data logger what sensors are taking readings and when to take those readings.
SDCard_Test	Test to see if the Arduino can read and write to the SD card connected to the module.
SDISensorSetup	Allows to assign SDI-12 sensors to specific addresses to set up sensors on different addresses, these will be where the readings are taken from.
SetDateTime	Connects to the DS3231 real-time clock and allows to set the current desired time to be used for the data logging.

Note some sketches within the Datalogger folder are “tabs” that separate specific functions. They are all connected to one sketch and by opening any, the main data logger program will also open.

Each sketch will be walked through in different sections, as well as in the setup video. It would be beneficial if the Arduino IDE user interface was a familiar program. Please see the tutorial linked in [section 1.2](#) if not.

**Before continuing:** if the components 10 to 19 were not acquired then ignore sections on the RS232 connections, setup, and remote connection.

### 3.0 Assembling the Data logger

Before using the sketches, the modules and other components must be attached to the Arduino. This can be done directly with a jumper wire, jumper wires, and a breadboard, or soldered to a prototyping board. This will then allow for components to be checked for functionality before being deployed into the field. **NOTE: When using the USB-A input to verify and upload sketches, please disconnect the RS232 module from Arduino.** If the module is connected at the same time as the computer, the computer will not be able to upload sketches to the Arduino. This is because the Arduino is confused as to which port to send and receive data from. Please see the [Appendix: Physical Builds](#) for more images on physical implementations.

#### 3.1 Pinout for Components

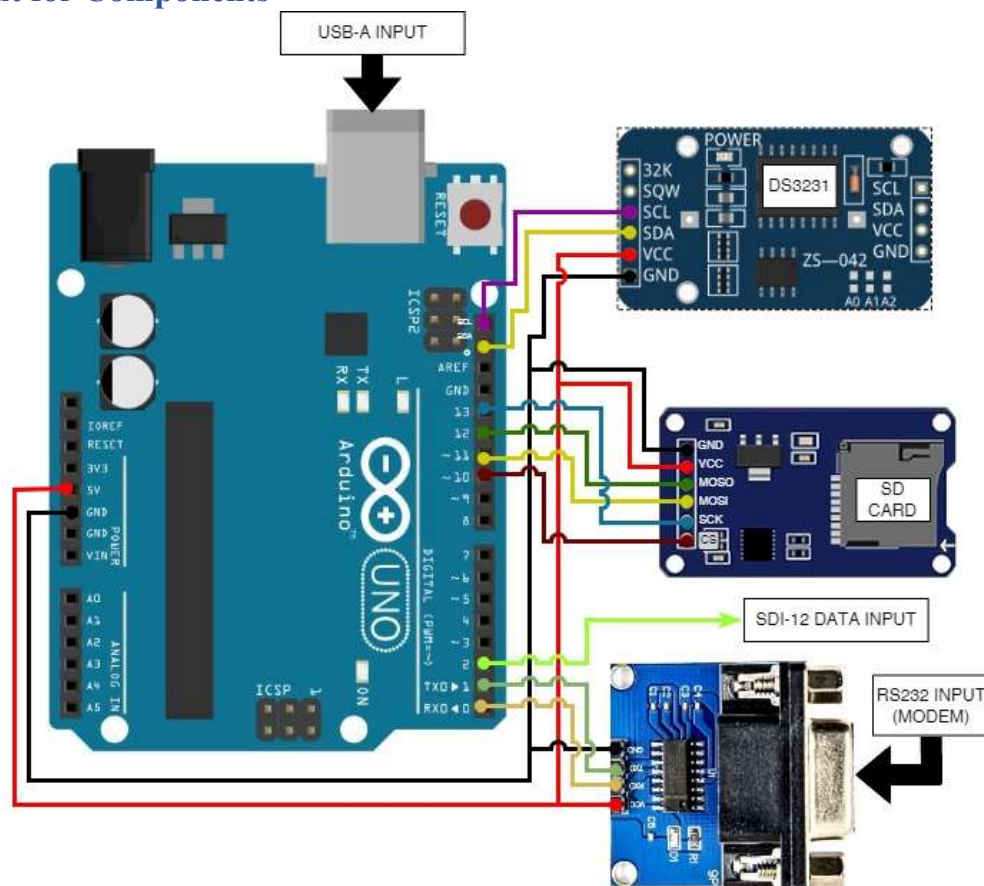


Figure 1 Pinout diagram [4,5,6,7].

#### 3.2 Connecting SDI-12 Sensors

To connect SDI-12 Sensors please review the sensor's manual for the input voltage. Also, review the colours of the wires for the specific SDI-12 sensor. Note that the external connections made to the other modules should stay connected while connecting the SDI-12 Sensors.

If the SDI-12 sensor has an input voltage of **5V**:

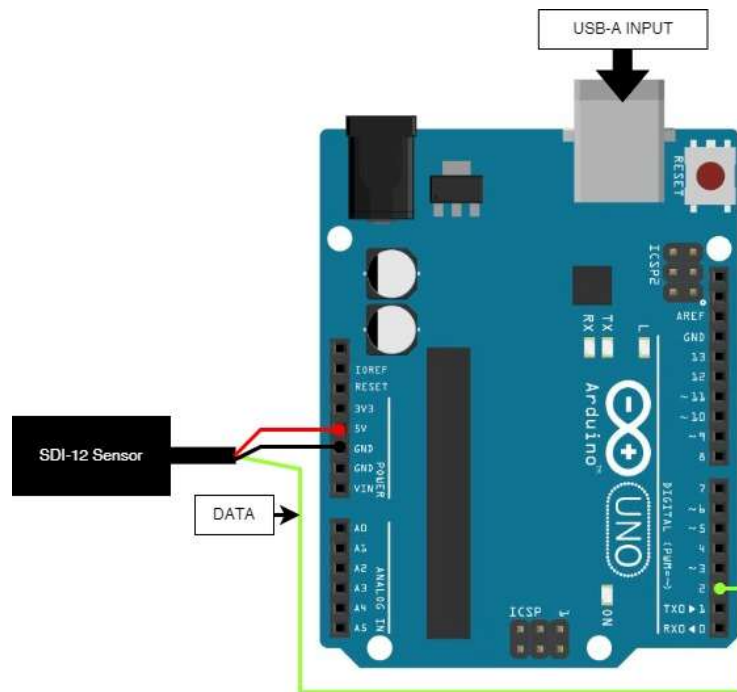


Figure 2 5V SDI-12 Sensor connection [4].

If the SDI-12 sensor has an input voltage of **12V**:

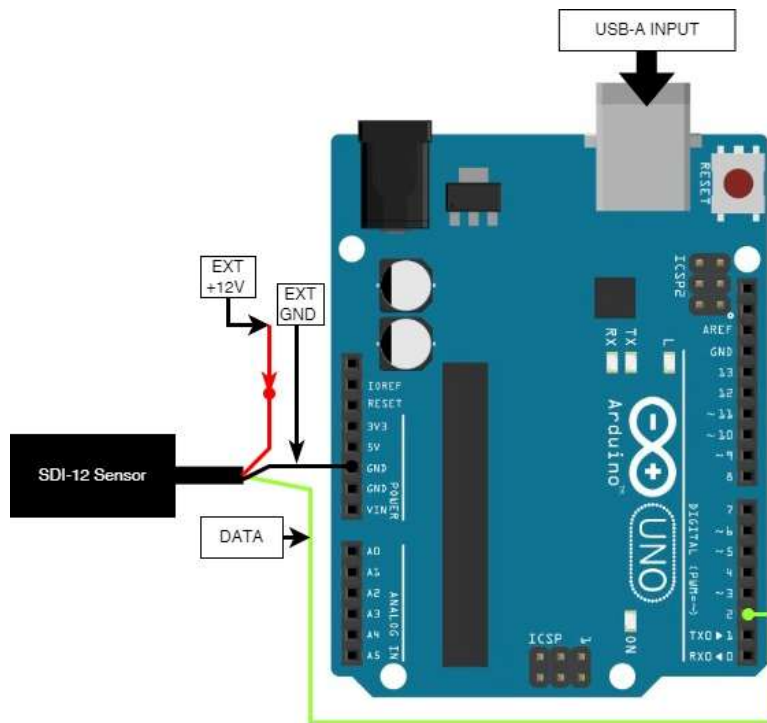


Figure 3 12V SDI-12 Sensor connection [4].

## 4.0 Data logger Setup

Before completing these next steps ensure that there is access to the Arduino IDE and that there is some familiarity with the Arduino language and how the IDE works. If this is not the case please see [section 1.2](#) before proceeding.

All the files needed can be found on:

### 4.1 Setting up Arduino IDE

Before uploading any sketches please ensure that the board and port settings are correct. Note that these can change when using different makes and models of Arduino boards. These can be found on the top menu under TOOLS. This is also where the Serial monitor option can be found:

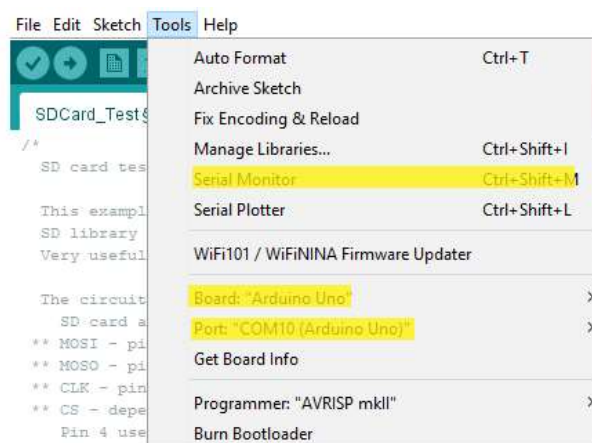


Figure 4 Arduino IDE settings.

To verify and upload the sketch there are buttons at the top left corner, or we can utilize the shortcuts:

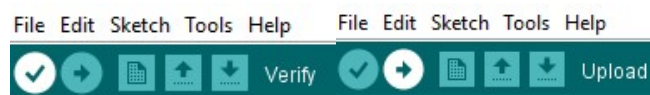


Figure 5 Verify/Upload buttons (white when selected).

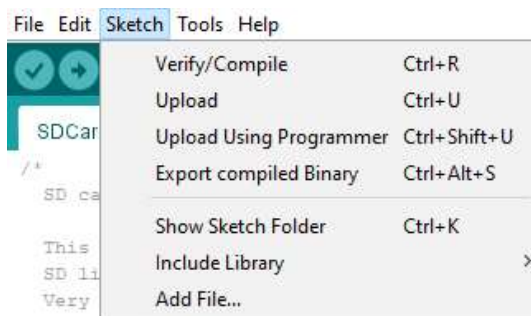


Figure 6 Verify/Upload shortcuts.

## 4.2 Testing the SD Card Module

Before putting the SD card into the module, the SD card must be formatted properly. Note that formatting the SD card will delete all data. Before formatting ensure it is either a new, unformatted SD card or the data is no longer required. The Arduino can only read SD cards in FAT16 or FAT32 formats. To do this you will need to install: [MiniTool Partition Wizard Free](#). Insert your micro SD card into the computer and open MiniTool Partition Wizard. Once open, find the micro SD card and right-click on the device. Then select the option FORMAT:

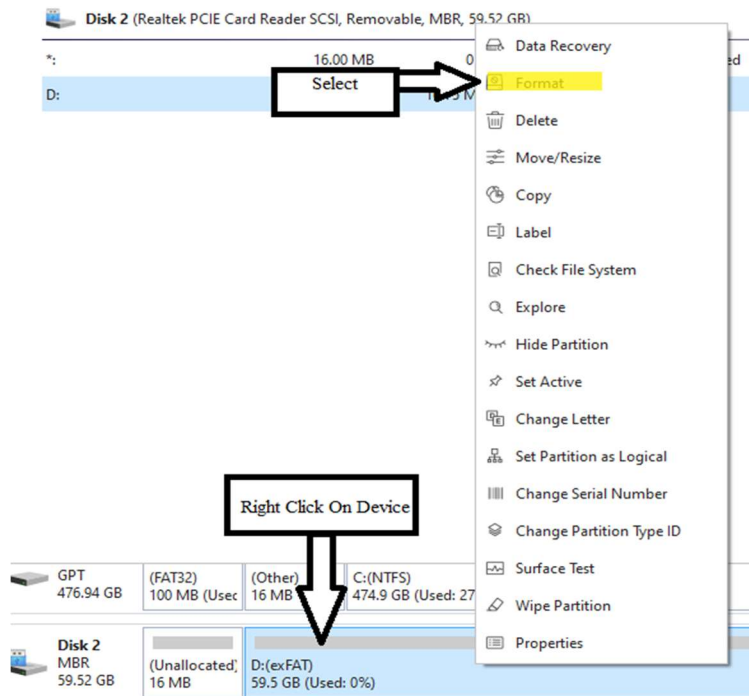


Figure 7 Format selection on MiniTool Partition Wizard.

Once selected format, label the partition, and select FAT32 under file system, and hit ok.

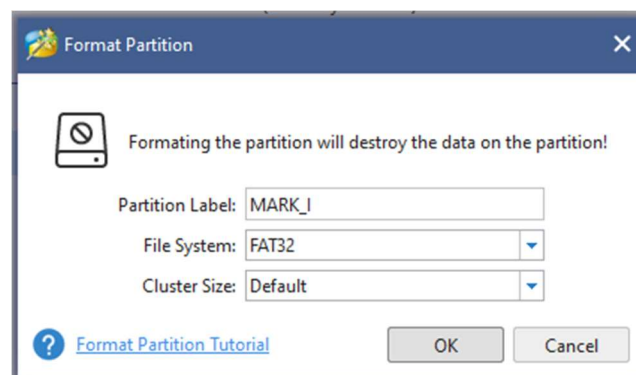


Figure 8 Partition formatting.

Finally, click APPLY in the bottom left corner to finalize the formatting of the SD card:

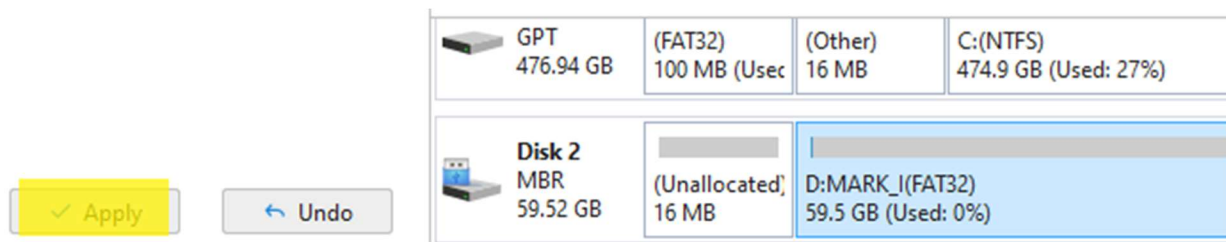


Figure 9 Apply the formatting to the SD card

To ensure the data logger can read and write to files on the SD card, we must next test that the Arduino can identify the SD card is connected. To do this, open the sketch **SDCard Test**.

Once open ensure that the serial baud rate is set to 9600, and the chip select pin is set to the same as the PIN connected to the CS pin on the SD card module, see figure 10. In figure 1 the CS pin is connected to pin 10 on the Arduino, thus we will use 10:

```
#define SERIAL_BAUD 9600
const int chipSelect = 10; // CS PIN CAN BE CHANGED
```

Figure 10 Parameters for the SD card test.

Now we can verify and upload the sketch to the Arduino. Once the sketch is uploaded open the serial monitor. If the Arduino is successfully connected to the SD card, the output should be:

```
Initializing SD card...Wiring is correct and a card is present.

Card type:          SDHC
Clusters:           3898766
Blocks x Cluster:   32
Total Blocks:       124760512

Volume type is:     FAT32
Volume size (Kb):    62380256
Volume size (Mb):    60918
Volume size (Gb):    59.49

Files found on the card (name, date and size in bytes):
SYSTEM~1/           2022-07-29 08:27:46
  WPSETT~1.DAT       2022-07-29 08:27:46 12
  INDEXE~1           2022-07-29 08:27:48 76
DATALOG.TXT         2000-01-01 01:00:00 0
TEST.TXT             2000-01-01 01:00:00 2540
```

Figure 11 Successful SD card test.

If the output in figure 11 is not seen please check:

- SD card formatting.
- SD card is inserted with a tight seal and correctly.
- The wiring to the SD card is correct with no obstructions.

### 4.3 Setting up DS3231 Real Time Clock

The DS3231 RTC module makes it easy and reliable to keep the correct time for measurements.

To set up the DS3231 open the sketch **SetDateTime**.

Next please ensure the Serial Baud rate is set to 9600 just like in the previous section.

Now set the current time into the parameters using 24HR time, and add 10 to 15 seconds to account for upload time. We must add 10 to 15 seconds because the RTC sets the time once the sketch is uploaded and run (i.e., the 10 to 15 seconds accounts for the time duration between defining the time in the SetDateTime sketch, and setting the RTC time by uploading and running the sketch).:

```
//Please set parameters to current time +10 sec upload time|
MyTimestamp.Day    = 4;
MyTimestamp.Month  = 8;
MyTimestamp.Year   = 22;
MyTimestamp.Hour   = 13;
MyTimestamp.Minute = 7;
MyTimestamp.Second = 30;
```

*Figure 12 Time parameters.*

In this example it is set to the 4<sup>th</sup> of August 2022 at 13:07:30. Once this is complete upload the sketch and then open the serial monitor. If the DS3231 setup was successful the date and time set in the parameters should be written in the serial monitor:

```
The time has been set to: 2022-08-04T13:07:30
End Of Program (RESET to run again)
```

*Figure 13 Successful DS3231 setup.*

If the output in figure 13 is not seen please check:

- The wiring to the DS3231 card is correct with no obstructions.
- DS3231 battery is inserted with a tight seal and correctly.

### 4.4 Setting up the SDI-12 Data Bus

When a new SDI-12 Sensor is added to a data bus, they are automatically assigned to address 0.

To have more than 1 sensor on a data bus each sensor must have its unique address. Thus, we will change every sensor address before deploying the Arduino in the field. SDI-12 sensors can be assigned to an address between 0-10, A-Z, and a-z. Note that this should be done one at a time so sensors assigned to the same address are not trying to communicate at the same time. To determine the sensor's address and re-assign it to a new one, open the sketch **SensorSetup**.

Once open ensure the serial baud rate is set to 9600, and the data pin is assigned to the pin where the SDI-12 sensor's data wire is connected to. We will use pin 2 as indicated in figure 1:



```
#define SERIAL_BAUD 9600    /*!< The baud rate for the output serial port */
#define DATA_PIN 2        /*!< The pin of the SDI-12 data bus */
```

*Figure 14 Assigning serial baud rate and a data pin.*

Now upload the sketch and open the serial monitor. If the sensor is connected properly the serial output should be:

```
Opening SDI-12 bus...
Checking address 0...Occupied
Sensor active at address 0.
Enter new address.
```

*Figure 15 Output of sensor address change.*

This says the current sensor is assigned to address 0. To reassign the sensor's address input a number from 1-10. Note that we could use letters, but until there are more than 10 sensors per data logger it is easier to keep the addresses as numbers in case they need to be transferred to a Sutron setup. It is a good idea to keep the address 0 free of sensors as that is the default address. We will reassign this sensor to address 1. Do this by typing 1 into the serial input line at the top of the serial monitor, and hitting enter:

```
1
Opening SDI-12 bus...
Checking address 0...Occupied
Sensor active at address 0.
Enter new address.
```

*Figure 16 Reassigning the sensor's address to 1.*

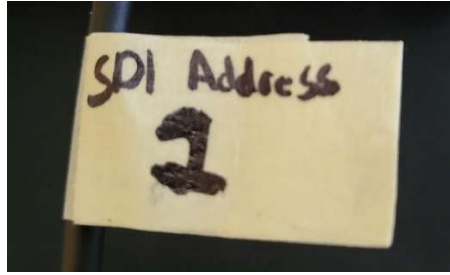
Now the sensor is reassigned to address 1 and the new output of the sketch will verify. Note that the sketch is on a never-ending loop, thus figure 17 will be repeated until you close the serial monitor and upload a new sketch :

```
Readdressing sensor.
Success. Rescanning for verification.
Checking address 0...Vacant
Vacant
Vacant
Checking address 1...Occupied
Sensor active at address 1.
Enter new address.
```

*Figure 17 Sensor is now at address 1.*

Do this for **every** sensor that is needed for the station. Make sure that each sensor has a unique address and is good practice to keep the address visible on the sensor:





*Figure 18 SDI address labeled on the sensor*

If the output in figure 17 is not seen, please check:

- Wiring to SDI-12 sensors is correct with no obstructions.
- Ensure the SDI-12 is connected to one common ground.
- Ensure the SDI-12 is connected to the proper form of power source.
- Ensure there is only one SDI-12 sensor connected at once.
  - This means that when setting up multiple sensors to new addresses, connect one and assign it to the desired address. Then disconnect the sensor and reconnect the next one. Repeat this cycle until all the sensors are set to the addresses of choice. Once they are **all set up** on different addresses, they can all be reconnected at once.

## 5.0 Data logger Sketch

This is where the program that completes the data logger will be set up, uploaded, and tested. Before continuing onto this section ensure that the SD card is connected and tested, the DS3231 clock is set up and working, and all the sensors are set up on distinct addresses. If not please see [Section 4.0: Data logger Setup](#).

To reset ALL the data in the files that will be initialized please uncomment the code found in the setup loop:

```
// THE CODE BELLOW WILL RESET THE ENTIRE FILE ONLY UNCOMMENT WHEN NEEDED
// if(SD.exists(DAILYDATA)){
//   SD.remove(DAILYDATA);    // Resetting the daily data
// }
// if(SD.exists(ORIGINALDATA)){
//   SD.remove(ORIGINALDATA);  // Resetting the original data
// }
//
```

Figure 19 Code that will reset data in files.

Note that this code should only be used when CERTAIN the data in the files are backed-up or unneeded. Otherwise, leave this section commented out of the program.

### 5.1 Initializing the Setup Parameters

To communicate properly the Arduino must use the same COM rates and defined PINs as seen in previous sections. This includes serial baud rate set to 9600, and defining the SDI data pin, and the SD card pin. In this example we use:

```
#define SERIAL_BAUD 9600    // Serial COM rate
```

Figure 20 Serial baud rate definitions.

```
#define DATA_PIN 2          // SDI-12 bus data pin
```

Figure 21 SDI data bus pin.

```
const int CHIP_SELECT = 10;  // The pin that the SD card pin CS is connected to
```

Figure 22 CS pin on SD card is connected to Arduino.

We can also define the file name for the data when it is logged. This is done by changing the text found here:

```
String ORIGINALDATA = "name_O.txt";
String DAILYDATA = "name_D.txt";
```

Figure 23 Filename definition.

For good practice change the name to the site location of the data logger. The first file “name\_O.txt” is the location where all the data collected will go, regardless of the WISKI

scraper. This file can be used to back up data or fill in missing data points. The second file “name\_D.txt” will be reset after the WISKI scraper takes the data. This means that any data inside will be lost after the WISKI command is sent to the Arduino.

Next, we can set the minute on which the data logger will take a measurement and log the data taken to the SD card. The data logger sketch is already set to take a measurement every hour, but it needs a defined minute of when to measure. The parameter that changes this can be changed here:

```
MyTimestamp.Minute = 0;    // Defining the minute to set the hourly alarm on
```

*Figure 24 Minute timestamp definition.*

Thus, in this example, the measurement(s) will be logged on the 0<sup>th</sup> minute of every hour.

## 5.2 Adding SDI-12 Measurements

Now we will add the measurements that are going to be taken from the sensors connected to the Arduino. Note that there are multiple locations where parameters need to be changed. Each measurement variable should be consistent from section to section. Also, note that there should only be one SDI data bus. This means each SDI sensor’s data output will be connected to the same pin and each one will have a specific address. For this example, all the SDI sensors data wires are connected to pin 2 of the Arduino, as seen in [figure 1](#). First, we will define the sensor's addresses, this can be found at the top of the sketch under the initialization section:

```
#define PT12_SENSOR_ADDRESS 1    // Defines PT12 sensor address
```

*Figure 25 Definition of sensor address.*

Note that it is good practice to include the sensor's name/type when defining the sensor’s address as it will make it easier to understand which sensor is taking specific measurements.

To add multiple sensors, copy and paste this code in a new line below and redefine the name of the sensor, and assign the proper address:

```
#define PT12_SENSOR_ADDRESS 1    // Defines PT12 sensor address  
#define SOIL_SENSOR_ADDRESS 2    // Defines Soil sensor address
```

*Figure 26 Defining multiple addresses.*

This data logger will now take measurements from a PT12 sensor on SDI address 1 and measurements from a Soil sensor on SDI address 2. Note that these sensors can have multiple measurements, which we will define next. To add measurement variables being measured, we must first define them as a float point number at the beginning of the main loop section:

```
// ADD THE VARIABLES BEING MEASURED AS FLOATS HERE:  
float HG; // Taken off of PT12  
float TW; // Taken off of PT12
```

*Figure 27 Floats of variables.*

The two variables that the PT12 sensor can take are HG (water level), and TW (water temperature). For reference to the hydrologic codes please see:

[https://www.weather.gov/media/mdl/SHEF\\_CodeManual\\_5July2012.pdf](https://www.weather.gov/media/mdl/SHEF_CodeManual_5July2012.pdf)

Now we will add the measurements into two sections of the sketch, where the data logger will log the measurements to the SD card, and where the data logger will print a live measurement to the serial monitor. Before this, we should understand the function that takes the measurement.

This function is defined as:

```
float Measurement_Output(int Sensor_Address, int num_reading, float Slope, float Offset )
```

*Figure 28 Measurement function.*

This function takes in the sensor address as an integer, which we defined earlier in the section. The reading number is an integer, which defines which variable is returned from multivariable sensors. For example, if the sensor can take three different variable measurements X, Y, and Z, the reading numbers would be as follows: X = reading number 1, Y = reading number 2, and Z = reading number 3. The slope and offset are applied to the reading as follows:

$$\text{Output Reading} = (\text{Raw Measurement} * \text{Slope}) + \text{Offset}$$

It is common to find the slope of variables in the sensor's manual. The offset is commonly applied due to external factors such as location. Now we can apply this to the sketch. First in the data logging to SD card section:

```
// Add measurements here:
HG = Measurement_Output(PT12_SENSOR_ADDRESS, 1, 0.704, 0.0); // 1st measurement on PT12_SENSOR_ADDRESS
TW = Measurement_Output(PT12_SENSOR_ADDRESS, 2, 1.0, 0.0);    // 2nd measurement on PT12_SENSOR_ADDRESS
```

*Figure 29 Two measurements were taken from the PT12 sensor.*

As seen in figure 29 the variables HG, and TW have applied the correct slopes taken from the PT12 manual and no offsets. Note that the HG variable will normally have an applied offset due to the location of the gauge station. We should also ensure that the string of data outputted to the SD card is updated to have the correct parameters in line with the headings. Note “\t” adds a tab to the outputted string:

```
// Adding the parameters to the string that will be logged to the SD card
DataString += String(HG) + "\t" + String(TW);
```

*Figure 30 Adding variables to the data string.*

To add a new parameter or edit others, changes must be made to the variable definitions and the measurement definitions:

```
// ADD THE VARIABLES BEING MEASURED AS FLOATS HERE:
float VWC; // Taken off of Soil Sensor
float TW;  // Taken off of PT12
```

*Figure 31 Definitions of different parameters.*

```
// Add measurements here:
VWC = Measurement_Output(SOIL_SENSOR_ADDRESS, 1, 1.0, 0.0); // 1st measurement on SOIL_SESNOR_ADDRESS
TW = Measurement_Output(PT12_SENSOR_ADDRESS, 2, 1.0, 0.0); // 2nd measurement on PT12_SENSOR_ADDRESS
```

*Figure 32 Taking different measurements.*

Make sure the data string is also updated:

```
// Adding the parameters to thestring that will be logged to the SD card
DataString += String(VWC) + "\t" + String(TW);
```

*Figure 33 Updated data string.*

To ensure that the live readings are also the correct measurements desired for the output, the parameters created can be copied into the program where serial input is required. Note that the parameters are changed back to the PT12 sensor that was connected for the example:

```
// Add measurements here (should be same as logging to sd card):
HG = Measurement_Output(PT12_SENSOR_ADDRESS, 1, 0.704, 0.0); // 1st measurement on PT12_SENSOR_ADDRESS
TW = Measurement_Output(PT12_SENSOR_ADDRESS, 2, 1.0, 0.0); // 2nd measurement on PT12_SENSOR_ADDRESS

// Adding the parameters to thestring that will be printed to serial( should be same as logging to sd card )
DataString += String(HG) + "\t" + String(TW);
```

*Figure 34 Parameters copied into next section*

Before this is complete, we also need to ensure that when the data logger is told to take a live reading the string of text outputted to the monitor is aligned to the measurements being taken on the data logger:

```
DataString += String(HG) + "\t" + String(TW);
Serial.print("\nDate\t\tTime\t\tHG\tTW\n" + String(DataString));
```

*Figure 35 Serial monitor printed text.*

If the variables are changed, please ensure all lines of code where this affects the sketch are changed. Once all these variables have been updated and changed to match that of the current sensors on the data bus, and the variables that are being logged and outputted. This sketch can be verified and uploaded, but if there are analog sensors needed to be added please see the next section before uploading the sketch.

Note that all the data collected from the sensors are recorded in the two files we named in [section 5.1](#). These parameters must all be separated by a single tab (“\t”) for the data to be properly scrapped into WISKI.

### 5.3 Adding Analog Sensors

**Note:** This section is only required if there is going to be a new analog sensor connected to the data logger. There is no integration of these in the original code as most sensors are SDI-12 compatible.

The Arduino microcontrollers have analog input pins, but this input is not in the unit volt. The analog inputs measure in a unitless measurement. These units can be converted into Volts by multiplying by 4.9 V/Unit.

Before adding analog sensors, it should be known that in version 1 of the data logger setup manual there is only one function coded into the data logger program. This is for the Campbell scientific temperature sensor. To code in a new analog sensor please see the sensor manual for the correct formula for integration of the measured and input voltage into an output value.

To add the Campbell Scientific temperature sensor, attach the wires according to the manual. Note that the output voltage should be connected to an analog pin. Also, connect the 5 V output the sensor is attached to, to a **different** analog input pin. These two pins will be the inputs to the function. Before adding the function, we need to initialize the analog pins (before the setup loop) and the measurement as a floating-point number (in the main loop):

```
int analog5V = 0;    // Analog pin 0 connected to 5V power
int analogSensor = 1; // Analog pin 1 connected to sensor
```

*Figure 36 Initializing analog pins for sensors.*

```
// ADD THE VARIABLES BEING MEASURED AS FLOATS HERE:
float HG; // Taken off of PT12
float TW; // Taken off of PT12
float CampblTemp; // Initializong floating analog temp sensor
```

*Figure 37 Initializing float for measurement variable.*

Next, add the new measurements into every section where a measurement is taken. This also includes the text displayed as well as the headers for the data on the SD card. A good way to do this is by renaming the file names and adding the headers in the setup loop.

```
// Add measurements here:
HG = Measurement_Output(PT12_SENSOR_ADDRESS, 1, 0.704, 0.0); // 1st n
TW = Measurement_Output(PT12_SENSOR_ADDRESS, 2, 1.0, 0.0);    // 2nd n

Temp = CampbellSciTemp( analogSensor, analog5V);

// Adding the parameters to thestring that will be logged to the SD card
DataString += String(HG) + "\t" + String(TW) + "\t" + String(Temp);
```

*Figure 38 Adding sensors to the data.*

Note that this will have to be done in other locations. Once it has been added in all locations (including headers), please move on to the next section.



## 5.4 Checking the Datalogging Sketch

Before disconnecting the Arduino from the computer, we can check to see if the data logger is working. Thus, after uploading the data logger sketch, open the serial monitor. There are three commands the data logger will respond to:

- 1) Taking a live reading input = 111
- 2) View the complete original dataset = 222
- 3) WISKI import command = 9999 \*\*Should not be used by user\*\*

The third command should only be used by the WISKI data scrapper. This is because every time the command is sent the data within that file gets pulled into the database by the scrapper, and then the Arduino resets the file. This allows for smaller amounts of data to be pulled from the Arduino, but it could also be deleted by accident. Note that if the data is deleted before importing to WISKI, the original data file should have the deleted data.

Now with the serial monitor open, we can send both of these commands. In this example (with the PT12 sensor attached) the live reading command (111) should return the water level (HG) and the water temperature at the correct time:

Date	Time	HG	TW
08/09/22	13:00:42	0.23	16.13

*Figure 39 Live reading from the serial monitor.*

To view the logged data, send the proper command (222). If the alarm has not been set off then the output will only be the heading of the file. If the alarm has been set off (it is past the timestamp step in the setup of the data logger sketch), then there will be a datapoint logged and outputted to the monitor:

Date	Time	HG	TW
------	------	----	----

*Figure 40 Data from SD card on the serial monitor.*

If these outcomes can be seen in the serial monitor and all the variables are outputting correct values, the data logger program is now set up on the Arduino.

If there are missing values and/or values are not reading correctly, please start at the beginning of [Section 5](#) and ensure all steps are correctly followed. If all the steps are correctly followed, please check the [wiring](#), and the [testing sketches](#) again.

## 6.0 Retrieving Data

The data can be retrieved through a direct connection to the computer to the USB-A cable, or a remote connection the RS232 connection.

We can connect the Arduino to the modem to allow for telemetry communication. Note that this section also expects familiarity with Solar-PuTTY. This is a terminal emulator that will allow for a remote connection to be made between the data logger and a computer/laptop. Please see this video for a tutorial for more information on Solar-PuTTY: [Solar-PuTTY Free Tool Overview - YouTube](#)

### 6.1 Setting up a Modem

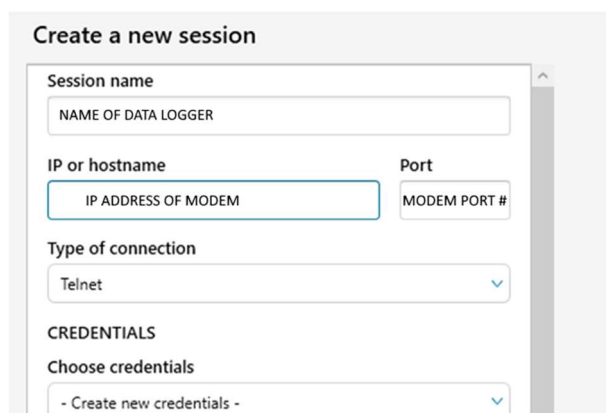
To set up the modem, please refer to microhard modem manual supplied with the product. The modem setup is complete when the modem can be plugged in, and it will connect to the network. It will also be able to connect and be managed on the microhard NMS site.

Using other modems is possible as well, just ensure that they can connect to a network and be managed in some way.

### 6.2 Connecting to Putty

Now that the modem is set up, we can connect it to the Arduino and test to ensure that results can be seen before connecting the Arduino's data to the WISKI database. To do this disconnect the Arduino from the computer and reconnect the Arduino to an external power source via the 12V DC to 5V DC USB converter component. Next, reconnect the RS232 adaptor to the Arduino according to [Figure 1](#), then connect the RS232 adaptor to the modem via a serial cable. Before moving any further be sure the Arduino is powered on and the serial connection is tight.

Open Solar-PuTTY and add the modem to a session by selecting the create new session button in the top right corner. Once in the create new session menu, select the Telnet connection type, and change the port to the same port number as the modem. Also, add the name of the data logger and the IP address of the modem. Note that the other options under credentials are not needed:



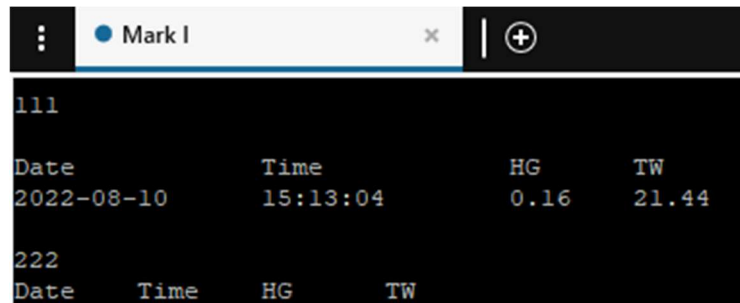
The screenshot shows a 'Create a new session' window. It contains the following fields and options:

- Session name:** A text box with the placeholder text 'NAME OF DATA LOGGER'.
- IP or hostname:** A text box with the placeholder text 'IP ADDRESS OF MODEM'.
- Port:** A text box with the placeholder text 'MODEM PORT #'.
- Type of connection:** A dropdown menu currently showing 'Telnet'.
- CREDENTIALS:** A section header.
- Choose credentials:** A dropdown menu currently showing '- Create new credentials -'.

Figure 41 Creating a new Solar-Putty session.



Once the new data logger inputs have been added hit the create button. The session should automatically start and connect to the data logger. To ensure the connection is working type the live reading command “111” and the SD card data command “222” and wait for a response:



The screenshot shows a PuTTY terminal window with a title bar that includes a menu icon, a tab labeled 'Mark I', and a close button. The terminal output is as follows:

```
111  
  
Date          Time          HG          TW  
2022-08-10    15:13:04    0.16       21.44  
  
222  
Date    Time    HG    TW
```

*Figure 42 Readings on Solar-PuTTY.*

Now we know that the data logger has successfully connected and can communicate with the modem.

## References

- [1] T. A. Team, “What is Arduino?,” *Arduino*. [Online]. Available: <https://www.arduino.cc/en/Guide/Introduction>. [Accessed: 04-Aug-2022].
- [2] “Arduino Uno REV3,” *Arduino Official Store*. [Online]. Available: [https://store.arduino.cc/products/arduino-uno-rev3?\\_gl=1%2Asrcnao%2A\\_ga%2AODE5MjAzOTQ4LjE2NTM1Nzc4MzQ.%2A\\_ga\\_NEXN8H46L5%2AMTY1OTYzNjc4Ni4yOS4xLjE2NTk2MzkyODguMjU](https://store.arduino.cc/products/arduino-uno-rev3?_gl=1%2Asrcnao%2A_ga%2AODE5MjAzOTQ4LjE2NTM1Nzc4MzQ.%2A_ga_NEXN8H46L5%2AMTY1OTYzNjc4Ni4yOS4xLjE2NTk2MzkyODguMjU). [Accessed: 04-Aug-2022].
- [3] “All-in-one, simply powerful remote monitoring,” *sdi12*. [Online]. Available: <https://sdi12.com/>. [Accessed: 04-Aug-2022].
- [4] “Arduino Diagram,” *coeleveld.com*. [Online]. Available: <https://coeleveld.com/arduino-led/>. [Accessed: 09-Aug-2022].
- [5] “Arduino DS3231 Diagram,” *FWD Skill Zone*. [Online]. Available: <https://www.fwdskillzone.com/arduino-103.html>. [Accessed: 09-Aug-2022].
- [6] W. by M. Akbari, “SD Card Module W/ arduino: How to read/write data,” *Electropeak*, 09-Jan-2022. [Online]. Available: <https://electropeak.com/learn/sd-card-module-read-write-arduino-tutorial/>. [Accessed: 09-Aug-2022].
- [7] “RS232 to TTL converter module female DB9 COM connector,” *Udvabony.com - Electronics, Sensors, Robotics Online Shop*, 04-Jun-2022. [Online]. Available: <https://udvabony.com/product/rs232-to-ttl-converter-module/>. [Accessed: 09-Aug-2022].
- [8] “Common SDI-12 commands and their meanings - cas.” [Online]. Available: <http://lighthouse.tamucc.edu/dnrpub/Sutron/XPert/Troubleshooting%20Documents/Common%20SDI-12%20commands%20and%20explanations.pdf>. [Accessed: 04-Aug-2022].

## Appendix

### SDI-12 Commands [8]

The first character of all commands and responses is a device address “a”. The last character of a command is the “!” character. The “!” can only be used in a command as the last character. For additional information on the SDI-12 protocol and the commands, please go to [www.sdi-12.org](http://www.sdi-12.org).

#### **a! Acknowledge Active**

This command is used to ensure that a sensor is responding to a data recorder or another SDI-12 device. It asks a sensor to acknowledge its presence on the SDI-12 bus. a! Send Identification

This command is used to query sensors for their SDI-12 compatibility level, model number, and firmware version number.

#### **?! Address Query**

When a question mark is used as the address character with the acknowledge active command (a!), the sensor will respond as if it is being addressed on the SDI-12 bus. Users should understand that if more than one sensor is connected to the bus, they will all respond, causing a bus contention.

#### **aAb! Change Address**

This command changes the address of a sensor. If the sensor supports software changeable addresses, it must support the change address command. After this command has been issued and responded to, the sensor is not required to respond to another command for one second. This gives the sensor time to write the new address to non-volatile memory. a is the current sensor address and b is the new address.

#### **aM! Start Measurement**

This command tells the sensor to take a measurement. The sensor does not, however, return the measurement to the data recorder after this command. It returns the time until one or more measurements will be ready and the number of measurements that it will make. The send data (D0!) command must be issued to get the measurement(s).

#### **aC! Start Concurrent Measurement**

This command tells the sensor to take a concurrent measurement. A concurrent measurement occurs while other SDI-12 sensors on the bus are also taking measurements. The send data (D0!) command must be issued to collect the measurements(s).

#### **aD0!...aD09! Send Data**

This command is used to get groups of data from the sensor. D0! is issued after an M, MC, C, CC, or V command. The sensor responds by sending the data. If the response to a D command is valid, but no data are returned, the sensor has aborted the measurement. To obtain data the recorder must issue another M, C, or V command.

#### **aR0!... aR9! Continuous Measurements**

Sensors -- such as shaft encoders -- that can continuously monitor the parameter to be measured

do not require a start measurement command (M!, M1! . . . M9!). They can be read directly with the R commands (R0! ... R9!).

**(D1! . . . D9!) Return of Multiple Measurements (Parameters)**

The commands D1 . . . D9 is used with sensors that return multiple measurements. The purpose of the D commands is for the sensor to return as many measurements as possible in response to each command.

**(aM1! . . . aM9!) Additional Measurement Commands**

Additional M commands provide a means to request different types of measurements from a sensor or to instruct a sensor to do calibration or a control function.

**(aC1! . . . aC9!) Additional Concurrent Measurement Commands**

Additional C commands provide a means to request different types of measurements from a sensor or to instruct a sensor to do calibration or a control function.

**(aV!) Start Verification**

This command tells the sensor to return verification in response to a subsequent D command. A verification sequence may include ROM signatures, CRCs, RAM test results, or the results of other diagnostics in the sensor.

## Physical Builds

