SETUP MANUAL FOR ARDUINO DATALOGGER

Version 2.0

AUGUST 22, 2023

MISSISSIPPI VALLEY CONSERVATION AUTHORITY
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

2.0 Datalogger Component 3

  2.1 Components List 3

  2.2 Software 3

3.0 Assembling the Datalogger 4

  3.1 Ordering the PCB 4

  3.2 Formatting the SD Card 5

  3.2 Assembling the Shield 6

4.0 Datalogger Setup 7

  4.1 Setting up Arduino IDE V.1.8.X 7

  4.2 Setting up the SDI-12 Data Bus 8

5.0 Datalogger Sketch 10

  5.1 Adding SDI-12 Measurements 10

  5.2 Adding Analog Sensors 13

  5.3 Checking the Datalogging Sketch 14

  5.4 Using the Menu 15

6.0 Remote Data Acquisition 18

References 19

Appendix 20

  SDI-12 Commands [4] 20

Table of Figures

Figure 2 File upload location on JLCPCB. 4
Figure 3 JLCPCB PCB Assembly option. 4
Figure 4 Format selection on MiniTool Partition Wizard. 5
Figure 5 Partition formatting. 5
Figure 6 Apply the formatting to the SD card 6
Figure 7 Assembled datalogger. 6
Figure 8 Arduino IDE settings. 7
Figure 9 Verify/Upload buttons (white when selected). 7
Figure 10 Verify/Upload shortcuts. 8
Figure 11 Assigning serial baud rate and a data pin. 8
Figure 12 Output of sensor address change. 8
Figure 13 Reassigning the sensor's address to 1. 9
Figure 14 Sensor is now on address 1. 9
Figure 15 SDI address labeled on the sensor 9
Figure 16 Code that will reset data in files. 10
Figure 17 Definition of sensor address. 10
Figure 18 Defining multiple addresses. 10
Figure 19 Floats of variables. 11
Figure 20 Measurement function. 11
Figure 21 Two measurements were taken from the PT12 sensor. 11
Figure 22 Adding variables to the data string. 11
Figure 23 Definitions of different parameters. 12
Figure 24 Taking different measurements. 12
Figure 25 Updated data string. 12
Figure 26 Parameters copied into next section 12
Figure 27 Serial monitor printed text. 12
Figure 28 Initializing analog pins for sensors. 13
Figure 29 Initializing float for measurement variable. 13
Figure 30 Adding sensors to the data. 14
Figure 31 Live reading from the serial monitor. 14
Figure 32 Data from SD card on the serial monitor. 15
Figure 33 The menu displayed when using the user interface. 15
Figure 34 Option 1. 15
Figure 35 Option 2 16
Figure 36 Option 3. 16
Figure 37 Option 4. 16
Figure 38 Option 5. 17
Figure 39 Option 6. 17
Figure 40 Serial cable wiring to connect Arduino to a Modem. 18

1.0 Introduction

1.1 Hardware for Arduino

The Arduino is a microcontroller that uses a CPU to use and run sketches. 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 datalogger is the Arduino Mega 2560. This device has an operating voltage of 5V that is connected via USB-A [2]. This board has 16 Analog GPIO Pins and 54 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 Mega's functionality. We are going to be using a PCB datalogger shield that was designed using EasyEDA.

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:
Download Arduino IDE: Software | Arduino
Arduino Web Editor: Arduino SSO

Note that this manual uses the Legacy version of the Arduino software. The newer version of the IDE can also be used with alteration to the code files. The newer version few minor differences in the user experience.

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 dataloggers. These sensors use a simple 3-wire interface that includes; VCC, GROUND, and DATA [3]. They work by being assigned to a specific sensor address on a data bus. The datalogger 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 dataloggers 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 datalogger uses an RS232 connection to connect to an external modem. This allows for 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 from the Arduino. Solar-PuTTY can be found on here: https://www.solarwinds.com/free-tools/solar-putty

2.0 Datalogger Component

For the datalogger to get data from the SDI sensors into a 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 the chosen database.

2.1 Components List

Note that the links are just suggested purchases.

Component	Description	Purchase Link
Arduino Mega & USB Cable	The microcontroller is the main component of the datalogger.	Arduino Mega 2560 Rev3
PCB data logging shield	Main components of datalogger including SD card, clock, serial connection and more.	SEE ORDERING DETAILS
Arduino Headers	Header pins to connect the Mega and the shield	Shield Stacking Header Set
12V to 5V converter	Converts 12V coming off of a battery to 5V to help protect the Arduino from power overload.	DC to DC Step Down Converter Module 12V/24V to 5V 3A
Micro SD Card	Inserted into the SD card module to store data.	Micro SD Flash Memory Card 64GB - 5 Pack
CR2032 3V Battery	Externally powers the DS3231 to keep time without direct power.	LiCB CR2032 3v Lithium Batteries
Waterproof Case	Case for Arduino to ensure it is not exposed to water. Maybe a good idea to have desiccants inside.	Waterproof ABS Project Box
SDI Sensors and Manuals	These will be the sensors used to take measurements. The manuals are required for slopes and wiring diagrams.	N/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 datalogger are found on the MVCA GitHub:

Mississippi-Valley-CA · GitHub

Each sketch will be walked through in different sketches. 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.

3.0 Assembling the Datalogger

Before using the sketches, the datalogger shield must be attached to the Arduino. This section will go through the process of ordering and setting up the datalogger shield. Note that a soldering iron will be required to assemble the header pins onto the datalogger shield.

3.1 Ordering the PCB

To order the PCB we need three files, the Gerber file, bill of materials (BOM), and pick and place files. All these files are available on the MVCA GitHub: Mississippi-Valley-CA · GitHub

Once all these files are downloaded open JLCPCB website. Upload the Gerber file to the small box seen in figure 3.

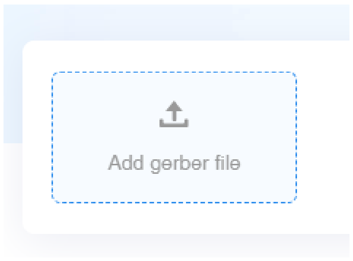
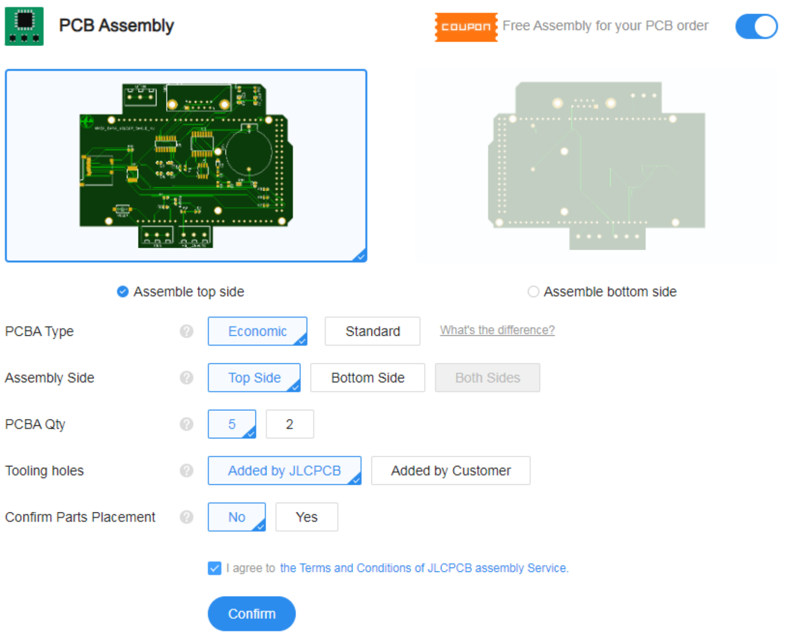


Figure 1 File upload location on JLCPCB.

Once uploaded there will be many options that can be changed on the PCB. Please select the quantity of the PCB that need to be ordered (minimum 5). Note that none needs to be changed in order for it to function, but can be changed if desired. Before moving on scroll down and select PCB Assembly option.



[✓ Assemble top side] Assemble bottom side

PCBA Type	[✓ Economic] Standard (What's the difference?)
Assembly Side	[✓ Top Side] Bottom Side Both Sides
PCBA Qty	[✓ 5] 2
Tooling holes	[✓ Added by JLCPCB] Added by Customer
Confirm Parts Placement	[✓ No] Yes

[✓] I agree to the Terms and Conditions of JLCPCB assembly Service.

Figure 2 JLCPCB PCB Assembly option.

Ensure that it is assembling on the top side and select how many of the PCB boards are going to be fully assembled (minimum 2 of the PCB being ordered). Hit confirm and move onto uploading the BOM and Pick and Place files. The Pick and Place file is uploaded to the CPL location. Once these files have been uploaded ensure that all the materials are in stock and selected. If a component is not in stock possibly go back to the PCB design and swap the component for one that is similar and in stock. Once the components are selected the placements will be shown. Ensure that the placements are correct by comparing this view to the 3D view option on EasyEDA PCB file. After all this is finished proceed to filling out the checkout information.

3.2 Formatting the SD Card

Before putting the SD card into the module, the SD card must be formatted properly. The Arduino can only read SD cards in FAT16 or FAT32 formats (FAT32 proffered). 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:

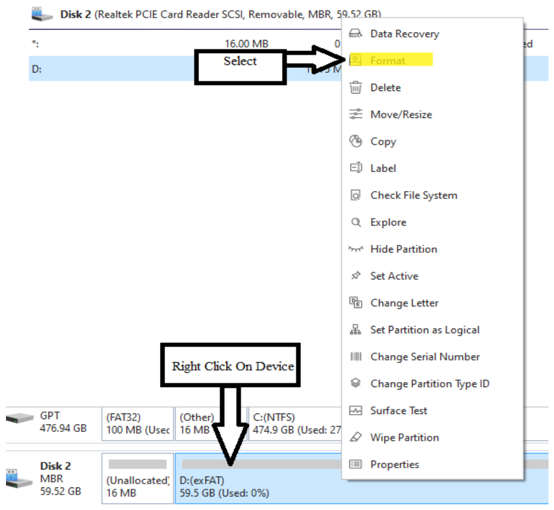
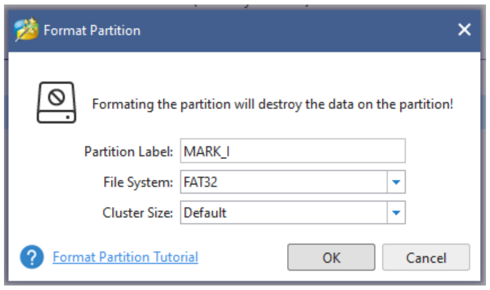


Figure 3 Format selection on MiniTool Partition Wizard.

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

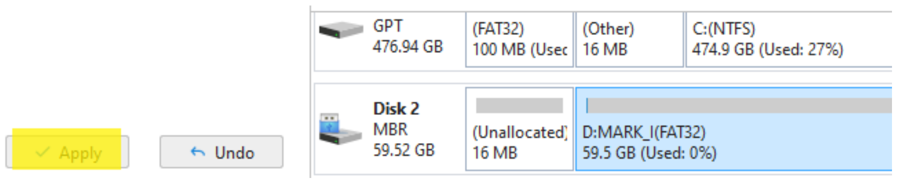

Format Partition

Formating the partition will destroy the data on the partition!

Partition Label: MARK_I
File System: FAT32
Cluster Size: Default

Figure 4 Partition formatting.

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



GPT 476.94 GB	(FAT32) 100 MB (Usec	(Other) 16 MB	C:(NTFS) 474.9 GB (Used: 27%)
Disk 2 MBR 59.52 GB	(Unallocated) 16 MB	D:MARK_I(FAT32) 59.5 GB (Used: 0%)	

Figure 5 Apply the formatting to the SD card

3.2 Assembling the Shield

Once the datalogger shield comes there will only be holes where the header pins go to mount the shield to the Arduino. Using a soldering iron, solder the header pins to the shield. Try to ensure that they are all even to have a good connection when the shield is mounted to the Arduino.

Once finished and the SD card is formatted, we can put the DS3231 battery and SD card into the proper slots. Then the shield can be carefully placed onto the Arduino Mega. The final product should look like figure 7.

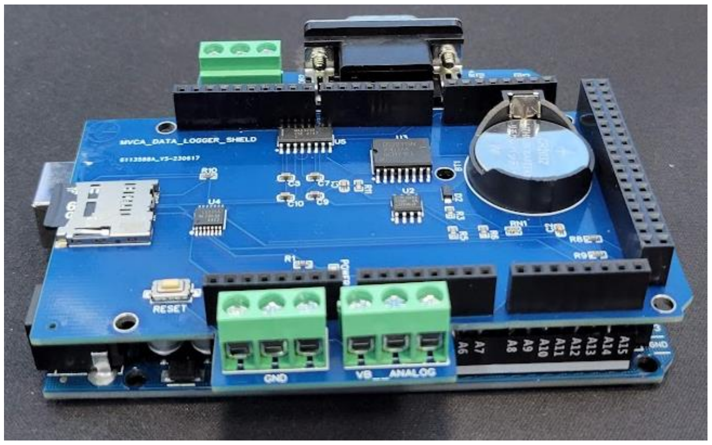


Figure 6 Assembled datalogger.

Before use please look at the pin connection between the Arduino and the shield. Ensure they are all in the proper slots, none are bent or damaged, and they are inserted far enough to ensure a good connection.

4.0 Datalogger Setup

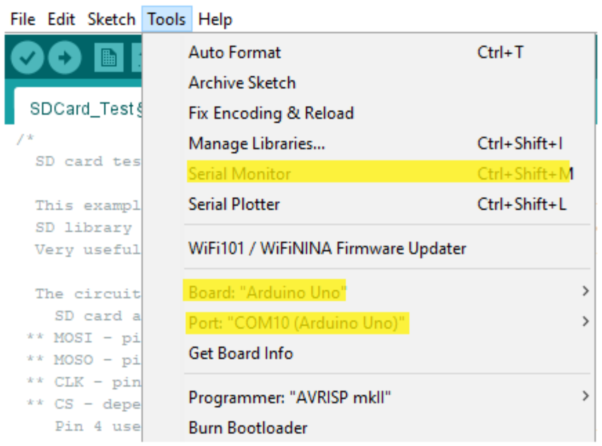
Before completing these next steps ensure that there is access to the Arduino IDE and that there is some familiarity with the Arduino language and how the IDE works. If this is not the case please see section 1.2 to become proceeding. Note that this is using Arduino IDE version 1.8.19. There is a new version 2.1.0 which can also be used just note that some features and menus are in different locations on the IDE terminal but the main functions and premises are the same.

All the files needed can be found on the MVCA GitHub:

Mississippi-Valley-CA · GitHub

4.1 Setting up Arduino IDE V.1.8.X

Before uploading any sketches please ensure that the board and port settings are correct. These can be found on the top menu under TOOLS. This is also where the Serial monitor option can be found:


File Edit Sketch Tools Help
SDCard_Test
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Get Board Info
Burn Bootloader

Board: "Arduino Uno"
Port: "COM10 (Arduino Uno)"
Programmer: "AVRISP mkII"

Figure 7 Arduino IDE settings.

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

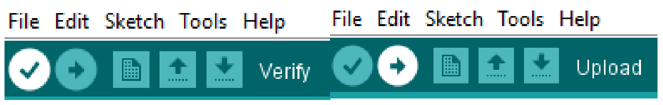


Figure 8 Verify/Upload buttons (white when selected).

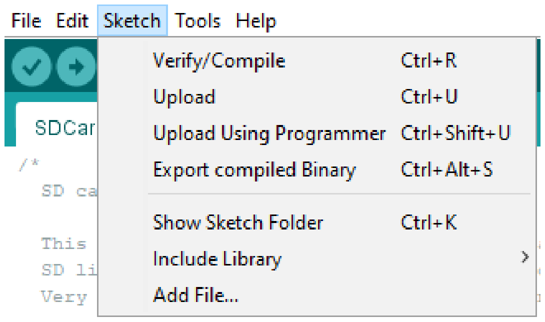
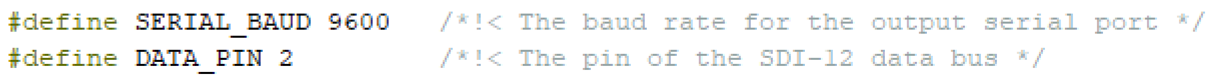

File
Edit
Sketch
Tools
Help
SDCar
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...

Figure 9 Verify/Upload shortcuts.

4.2 Setting up the SDI-12 Data Bus

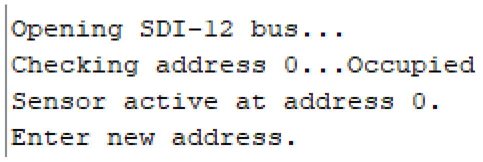
When a new SDI-12 Sensor is added to a data bus, they are automatically assigned to address 0. To have more than 1 sensor on a data bus each sensor must have its unique address. Thus, we will change every sensor address before deploying the Arduino in the field. Using Arduino SDI-12 sensors can be assigned to an address between 0-10. 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 10 Assigning serial baud rate and a data pin.

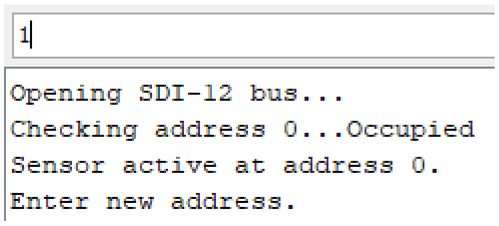
Now upload the sketch and open the serial monitor. If the sensor is connected properly the serial output should be:



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

Figure 11 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 datalogger 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:

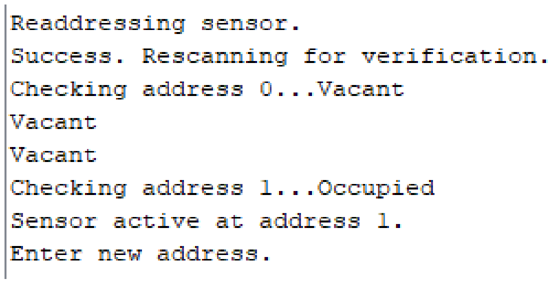


```
1

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

Figure 12 Reassigning the sensor's address to 1.

Now the sensor is reassigned to address 1 and the new output of the sketch will verify:



```
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 13 Sensor is now on address 1.

Do this for **every** SDI-12 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:

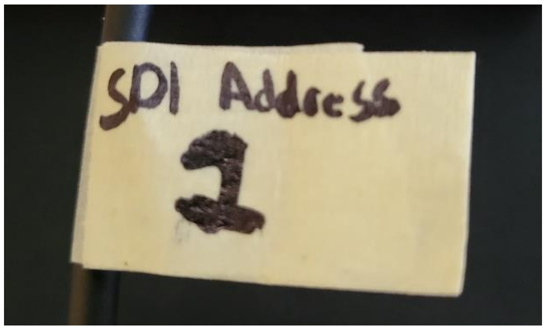


Figure 14 SDI address labeled on the sensor

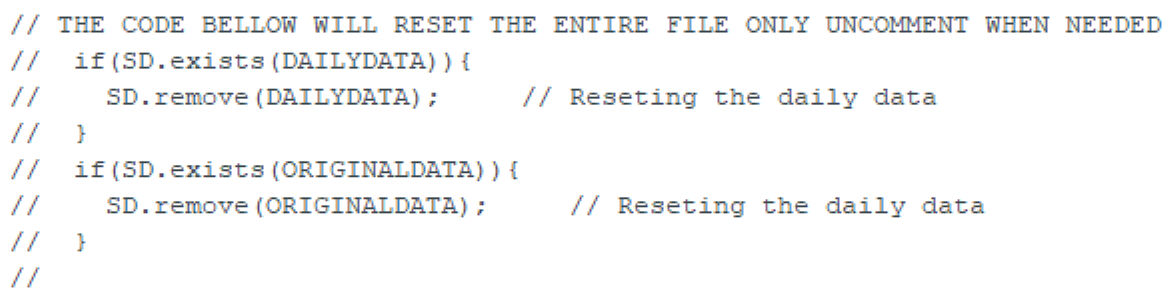
If the output in figure 18 is not seen please check:

- 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.
- There is only one SDI-12 sensor connected at once.
    - Need to conduct the address assignment one sensor at a time.

5.0 Datalogger Sketch

This is where the program that completes the datalogger 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: Datalogger Setup.

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



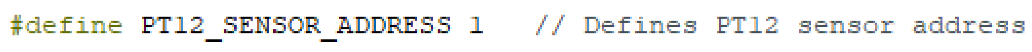
```
// THE CODE BELLOW WILL RESET THE ENTIRE FILE ONLY UNCOMMENT WHEN NEEDED
//  if(SD.exists(DAILYDATA)){
//    SD.remove(DAILYDATA);     // Reseting the daily data
//  }
//  if(SD.exists(ORIGINALDATA)){
//    SD.remove(ORIGINALDATA);     // Reseting the daily data
//  }
//
```

Figure 15 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. Also, please ensure that the last sketch uploaded to the datalogger has this section COMMENTED out otherwise every time the Arduino is reset the data on the SD card will also be reset.

5.1 Adding SDI-12 Measurements

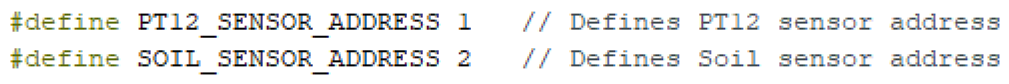
Now we will add the measurements that are going to be taken from the sensors connected to the Arduino. Note that there are multiple locations where parameters need to be changed. Each measurement variable should be consistent from section to section. Also, note that there should only be one SDI data bus. This means all SDI sensor data output are connected to the same pin and each has a specific address. 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 16 Definition of sensor address.

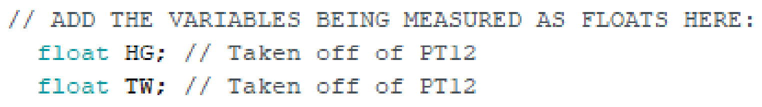
Note that it is good practice to include the sensor's name/type when defining the sensor's address as it will make it easier to understand which sensor is taking specific measurements.
To add multiple sensors, copy and paste this code in a new line below and redefine the name of the sensor, and assign the proper address:



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

Figure 17 Defining multiple addresses.

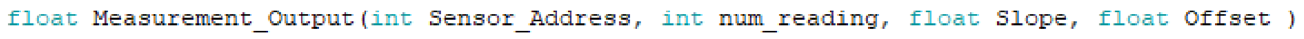
This datalogger will now take measurements from a PT12 sensor on SDI address 1 and measurements from a Soil sensor on SDI address 2. Note that these sensors can have multiple measurements, which we will define next. To add measurement variables being measured, we must first define them as a float point 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 18 Floats of variables.

The two variables that the PT12 sensor can take are HG (water level), and TW (water temperature). Now we will add the measurements into two sections of the sketch, where the datalogger will log the measurements to the SD card, and where the datalogger 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:



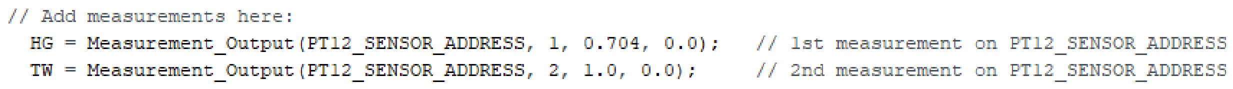
```
float Measurement_Output(int Sensor_Address, int num_reading, float Slope, float Offset )
```

Figure 19 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 X, 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:

Output Reading = (Raw Measurement * Slope) + Offset

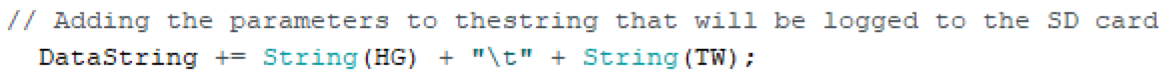
It is common to find the slope of variables in the sensor’s manual. The offset is commonly applied due to external factors such as location. Now we can apply this to the sketch. First in the data logging to SD card section:



```
// 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 20 Two measurements were taken from the PT12 sensor.

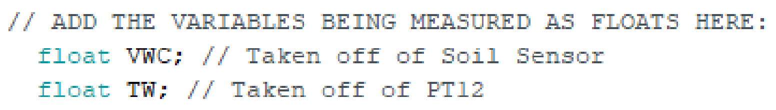
As seen in figure 28, the variables HG, and TW have applied the correct slopes taken from the PT12 manual and no slopes. Note that the HG variable will normally have an applied slope due to the location of the gauge station. We should also ensure that the string of data outputted to the SD card is updated to have the correct parameters in line with the headings:



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

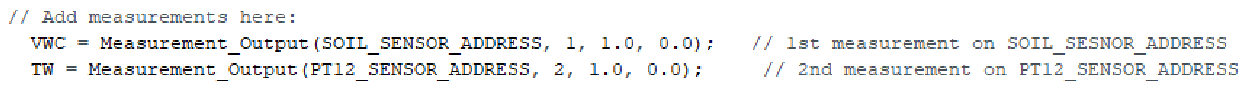
Figure 21 Adding variables to the data string.

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
```

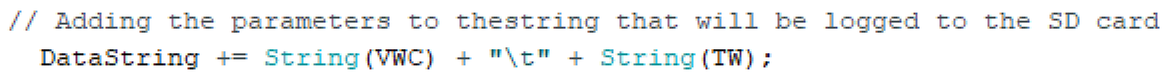
Figure 22 Definitions of different parameters.



```
// 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 23 Taking different measurements.

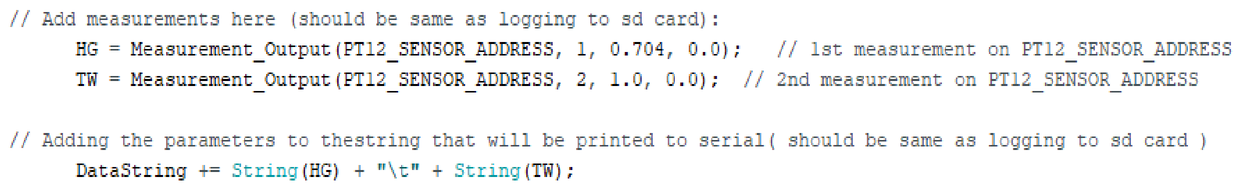
Make sure the data string is also updated:



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

Figure 24 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:

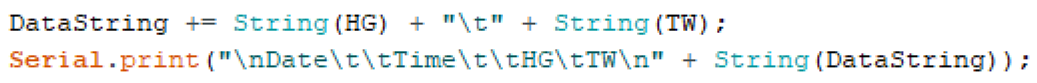


```
// Add measurements here (should be same as logging to sd card):
      HG = Measurement_Output(PT12_SENSOR_ADDRESS, 1, 0.704, 0.0);   // 1st measurement on PT12_SENSOR_ADDRESS
      TW = Measurement_Output(PT12_SENSOR_ADDRESS, 2, 1.0, 0.0);  // 2nd measurement on PT12_SENSOR_ADDRESS

// 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 25 Parameters copied into next section

Before this is complete, we also need to ensure that the string of text outputted to the serial monitor if asked to take a live reading is corrected to the measurement variables:



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

Figure 26 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 the next section before uploading the sketch.

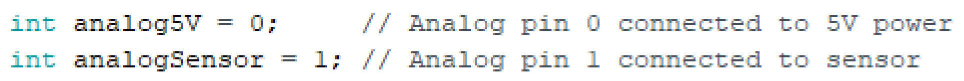
5.2 Adding Analog Sensors

Note: This section is only required if there is going to be a new analog sensor connected to the datalogger. 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 (5 / 1023).

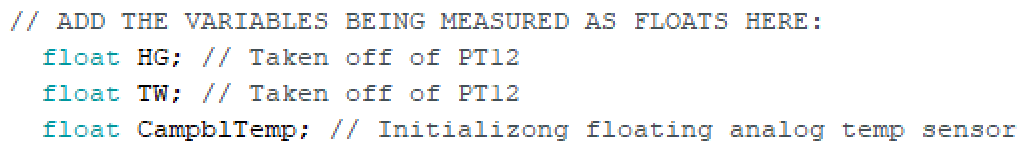
Before adding analog sensors, it should be known that in version 1 of the datalogger setup manual there is only one function coded into the datalogger program. This is for the Campbell scientific temperature sensor. To code in a new analog sensor please see the sensor manual for the correct formula for 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 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
```

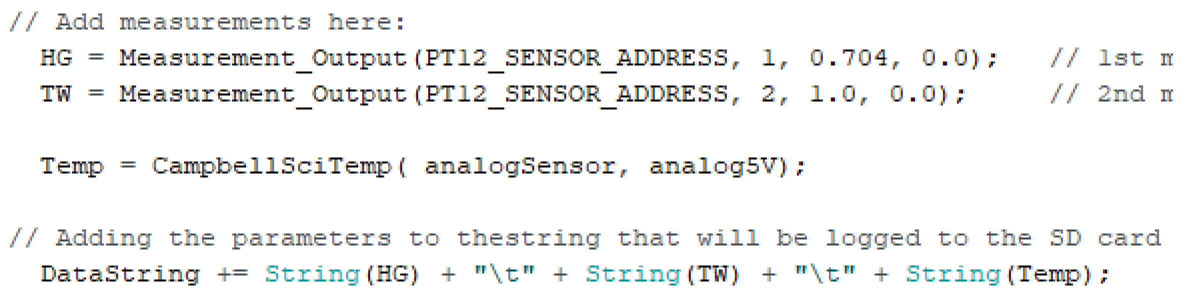
Figure 27 Initializing analog pins for sensors.



```
// 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 28 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 m
  TW = Measurement_Output(PT12_SENSOR_ADDRESS, 2, 1.0, 0.0);      // 2nd m

  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 29 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.

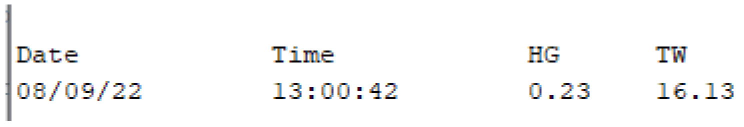
5.3 Checking the Datalogging Sketch

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

1) Taking a live reading input = live
2) View the complete original dataset = log
3) Open user interface = menu
4) Data import command = 9999 **Should not be used by user**

The fourth command should only be used when retrieving data remotely. This is because every time the command is sent the data within that file gets pulled into the database by a scrapper, the Arduino then 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 the database, the original data file should have the deleted data.

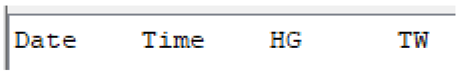
Now with the serial monitor open, we can send both of these commands. In this example (with the PT12 sensor attached) the live reading command should return the water level (HG) and the water temperature at the correct time:



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

Figure 30 Live reading from the serial monitor.

To view the logged data, send the proper command. If it has not past the next time to log data then the output will only be the heading of the file. If is has (it is past the timestamp step in the setup of the datalogger sketch), then there will be a datapoint logged and outputted to the monitor:



```
Date      Time      HG        TW
```

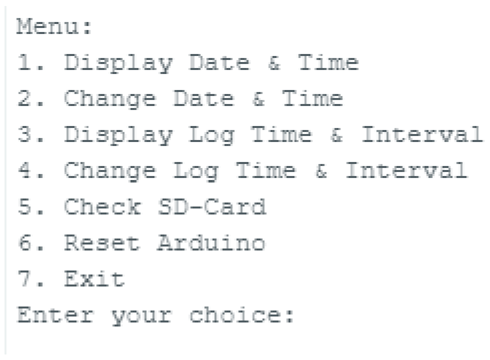
Figure 31 Data from SD card on the serial monitor.

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

If there are missing values and or values are not reading correctly please start at the begging 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.

5.4 Using the Menu

To use the user interface programmed into the datalogger use the command "menu" to open the menu which will open the menu:



```
Menu:
1. Display Date & Time
2. Change Date & Time
3. Display Log Time & Interval
4. Change Log Time & Interval
5. Check SD-Card
6. Reset Arduino
7. Exit
Enter your choice:
```

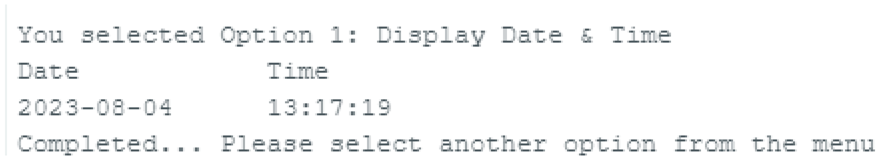
Figure 32 The menu displayed when using the user interface.

There are seven options programmed into the datalogger. To select an option, type the same number as the desired option into the serial monitor. Note that when the datalogger needs to log data it will display a message and exit the menu:

```
DATA LOGGING IN PROCESS... EXITING MENU
```

To re-enter the menu, wait until the data logging has been complete (10 seconds is enough) and re-enter the "menu" command.

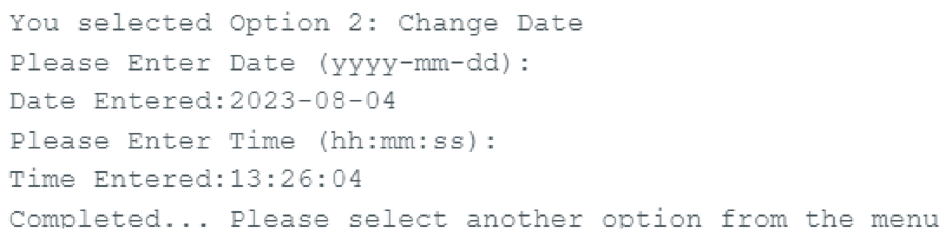
Option 1: This is used to display the date and time:



```
You selected Option 1: Display Date & Time
Date            Time
2023-08-04      13:17:19
Completed... Please select another option from the menu
```

Figure 33 Option 1.

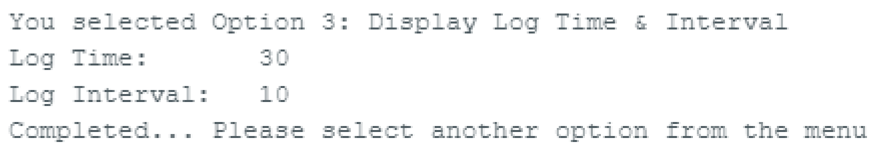
Option 2: This is used to alter the date and time of the datalogger. Once selected it will prompt the user to enter a date in the format “yyyy-mm-dd”. After entering the date it will ask for the time in the format “hh:mm:ss”. Note the clock is 24 hr (1:26:04 pm = 13:26:04). Only use eastern standard and not daylight savings time:



```
You selected Option 2: Change Date
Please Enter Date (yyyy-mm-dd):
Date Entered:2023-08-04
Please Enter Time (hh:mm:ss):
Time Entered:13:26:04
Completed... Please select another option from the menu
```

Figure 34 Option 2

Option 3: This is used to check what time the next datalogging will occur as well as the current interval datalogging occurs on:

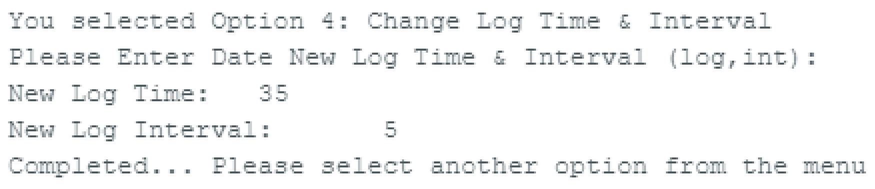


```
You selected Option 3: Display Log Time & Interval
Log Time:       30
Log Interval:   10
Completed... Please select another option from the menu
```

Figure 35 Option 3.

We can see that the next logging occurs at the 30th minute and it is logging every 10 minutes.

Option 4: This is used to change the logging time and interval. To change the times input the desired log time and interval in the format (LOGTIME, INTERVAL):

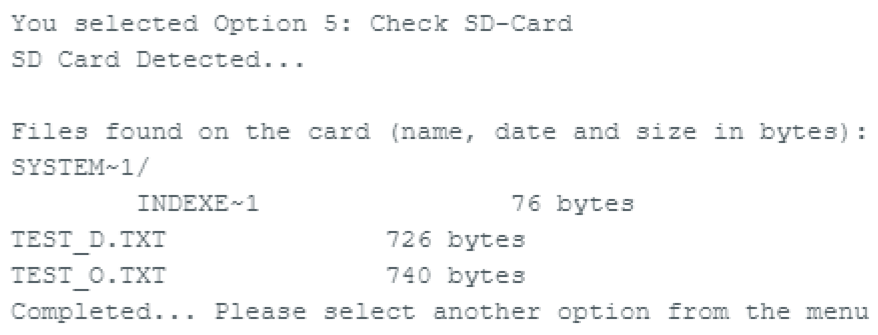


```
You selected Option 4: Change Log Time & Interval
Please Enter Date New Log Time & Interval (log,int):
New Log Time:   35
New Log Interval:       5
Completed... Please select another option from the menu
```

Figure 36 Option 4.

We can see that entering (35,5) changed the log time to 35th minute and will now log every 5 minutes.

Option 5: This is used to check to see if the SD card is accessible and view the files and their sizes on the SD card:

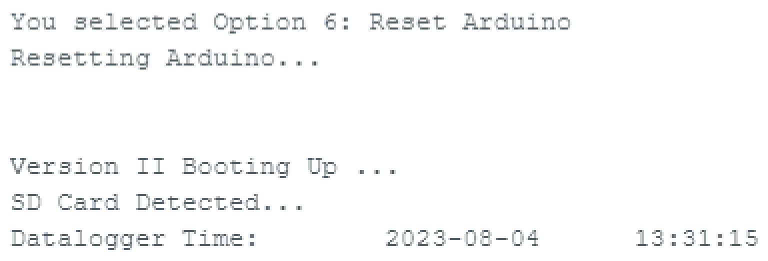


```
You selected Option 5: Check SD-Card
SD Card Detected...

Files found on the card (name, date and size in bytes):
SYSTEM~1/
        INDEXE~1                76 bytes
TEST_D.TXT              726 bytes
TEST_O.TXT              740 bytes
Completed... Please select another option from the menu
```

Figure 37 Option 5.

Option 6: This is used to reset the Arduino and is the same as hitting the reset button on the Arduino:



```
You selected Option 6: Reset Arduino
Resetting Arduino...


Version II Booting Up ...
SD Card Detected...
Datalogger Time:        2023-08-04      13:31:15
```

Figure 38 Option 6.

Option 7: This is used to exit the menu which returns to the main loop of the datalogger code. This makes the other commands “live” and “log” as well as data collection usable.

6.0 Remote Data Acquisition

Now that the datalogger is set up to take data and log it to the SD card with the correct parameters, we can now connect the Arduino to the modem to allow for telemetry communication. Note that this section also expects familiarity with Solar-PuTTY. Please see this video for a tutorial: Solar-PuTTY Free Tool Overview - YouTube

Setup any modem so that is can be accessed using Solar-PuTTY via the IP address attached to the Sim card being used with the modem. Attach the modem using an RS-232 serial cable. Login to the modem using Solar-Putty and try the “live” command. If the Arduino commands do not work try altering the RS232 cable to have connections according to figure 40 and try again. If this does not work contact: info@mvc.on.ca and ask for the *Full Stack Developer*.

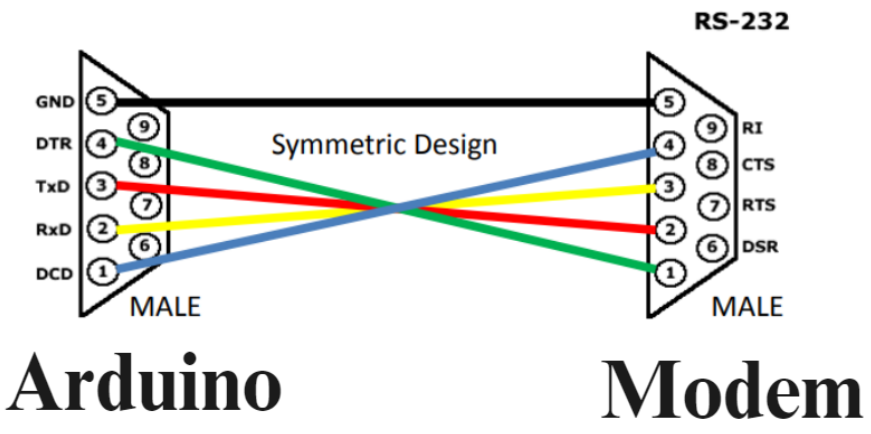


Figure 39 Serial cable wiring to connect Arduino to a Modem.

Once the modem and Arduino are connected use the “9999” command to acquire the new data from the Arduino. The “9999” command prints all the data from the last time the command was used. From here the data can be copied manually or have a script setup to automatically connect and download the data from the Arduino.

References

[1] T. A. Team, “What is Arduino?,” *Arduino*. [Online]. Available: https://www.arduino.cc/en/Guide/Introduction. [Accessed: 04-Aug-2022].

[2] “Arduino Mega 2560 REV3,” Arduino Official Store, https://store.arduino.cc/products/arduino-mega-2560-rev3 [accessed Jul. 6, 2023].

[3] “All-in-one, simply powerful remote monitoring,” *sdi12*. [Online]. Available: https://sdi12.com/. [Accessed: 04-Aug-2022].

[4] “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 [4]

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.

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. aI! 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.