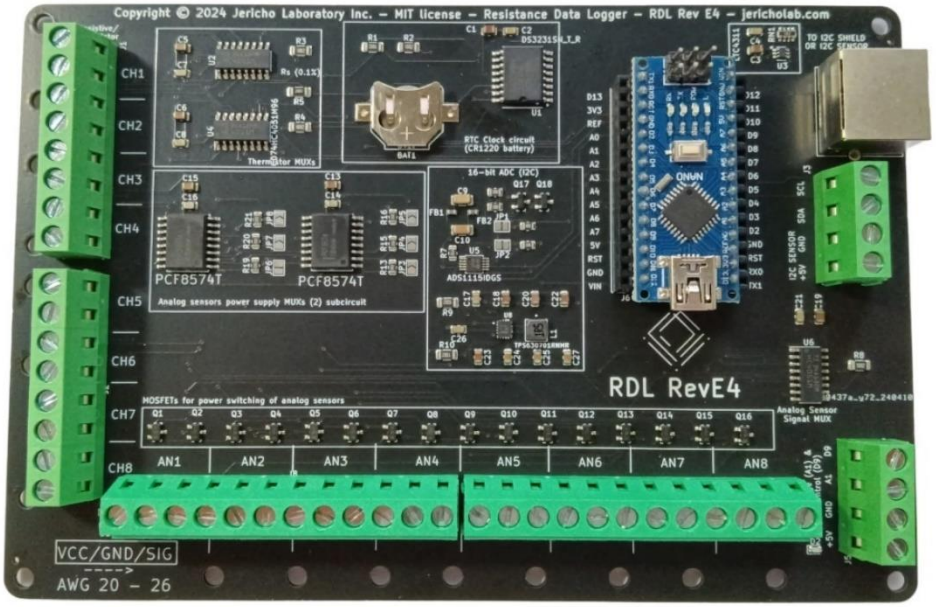
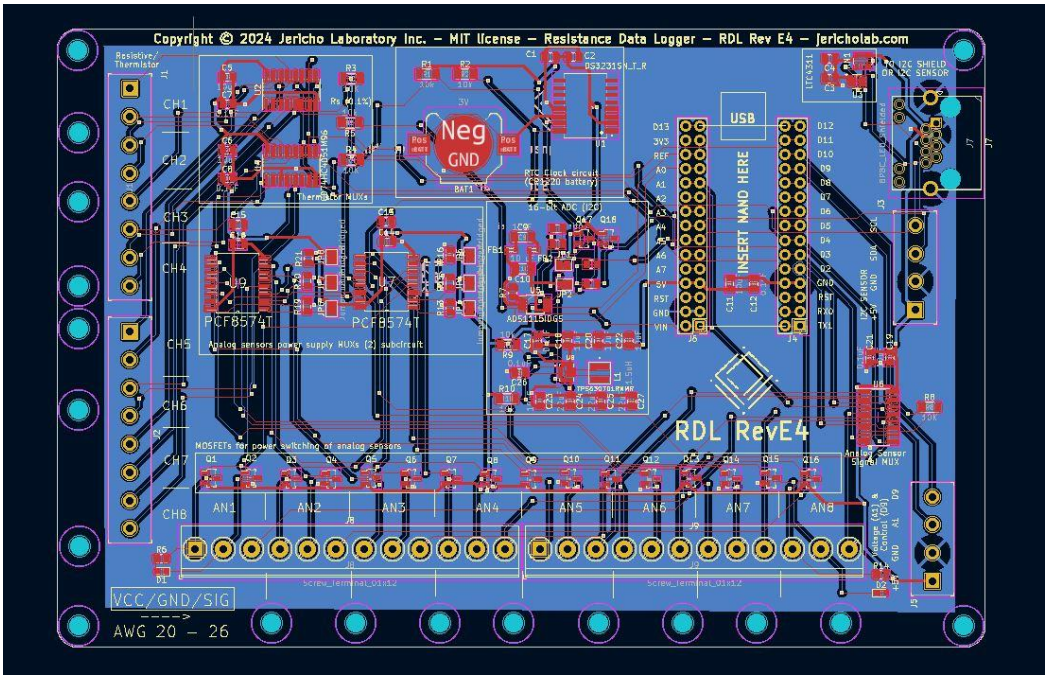
USER GUIDE – RESISTANCE DATA LOGGER (RDL) Rev E4 - WIP

JERICHO LABORATORY INC. - JERICHOLAB.COM

- -

Explore. Measure. Understand.





This page is intentionally left blank

Table of Contents

DISCLAIMER: You are responsible for your own safety. Jericho Laboratory Inc. will not be held responsible for your safety or any damage other than to the device itself. We advise the user to apply great caution while using this product. Always consult an accredited electrician before working with high-voltage electricity. This sensor operates with a low-voltage (5-12V) supply, but other components of your system might require an electrician. Do NOT install in areas prone to thunder as the system has no protection against thunder. It is recommended to NOT open the main enclosure during bad weather (rain, snow, dust) to minimize the risk of component damage and safety issues.

WARNING: This product is neither intended nor warranted for use in following equipment or devices: Special application (such as for medical devices, transportation equipment, traffic signal control equipment, fire and crime prevention equipment, aeronautics and space devices, nuclear power control, fuel control, in vehicle equipment safety devices, and so on) in which extremely high quality and high reliability is required, or if the malfunction or failures of product could be cause for loss of human life, bodily injury.

WARNING: This product contains small parts. Not for children under 5 years old.

PRODUCT OVERVIEW

Customizable, transparent and well-documented, this product is for those who want to own their data… and their tool.

The Resistance Data Logger (RDL) is a low-cost, ready-to-use, open-source data logger platform for scientists and engineers. Taking its name from the initial ability to measure resistive sensors such as thermistors, the 18-channel device can also measure humidity, luminosity, resistance, voltage, as well as do some basic control. It can communicate with most I2C sensors without additional hardware. Text-based, it focuses on data, providing only elementary graphing capabilities.

Note: For detailed information about the RDL board, please read the “RDL Design Overview” document.

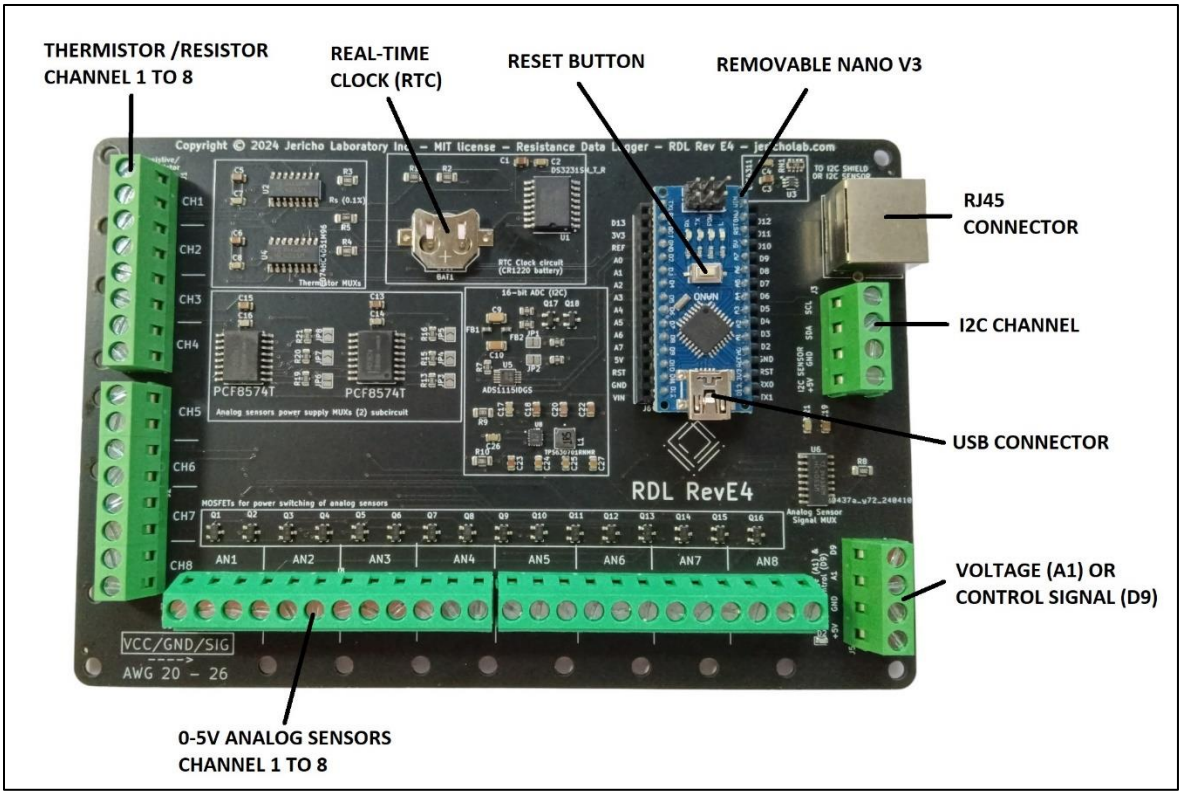


Figure 1 RDL rev E4 components layout

The RDL can measure a variety of variables over 18 channels. Exact availability of each measurement can be examined in Table 1. Humidity can be read with a digital sensor (I2C) from the SHT family (e.g. SHT4X by Jericho). Other I2C sensors (e.g., accelerometer) can be used by adding the required code. Digital control signals (0-5V, binary or PWM) can also be sent. Resistance measurements allow you to measure a range of phenomena from soil moisture to blood clot formation. Using a low-cost black globe and a wet bulb temperature, the RDL can monitor the comfort via the Wet Bulb Globe Temperature (WBGT) index.

The measured data is sent continuously to your computer, for purposes of display and data storage. The device is equipped with a temperature-adjusted Real-Time clock (RTC) and a 3V battery for accurate and resilient timestamp of the data. The device is compatible with Arduino Nano 3.0 © and its clones. There are many resources to learn about Arduino technology and coding among which the popular book by Jeremy Blum called “Exploring Arduino” (Blum 2019).

IMPORTANT: The RDL is unable to log any data if it is not properly communicating with a PC. Always make sure that the recording is active when using the product.

Table 1 RDL Channels Overview for revision E4

Zone identification	Connector type	Type of channel	Number of channels	Measurement ability	Control ability	Comments
J1, J2	Screw terminal	Voltage divider subcircuit (A0 or ADS)	8	Temperature (Resistor) Resistance Luminosity Soil humidity	N/A	Software signal filtering only. Measurements can be done with Nano ADC (pin A0, 10-bit) or ADS1115 (16-bit). Multiplexed channels.
J8, J9	Screw terminal	Arduino Analog pins (A1 or ADS)	8	Voltage (0-5V)	Digital signal PWM signal (A1 only)	Zone also provides 5V power supply via transistors. For differential voltage measurements, connect the second pin to a ground. Multiplexed channels.
J5	Screw terminal	Control signal output	2	Voltage (0-5V) (if A1 configured as input)	Digital signal PWM signal	D9 pin can output a digital (0/1) signal or PWM signal. A1 can be configured as input or output (digital, not PWM)
J7	RJ45 (Ethernet)	I2C	1	I2C sensors	I2C devices only	Allows connection to Jericho I2C shield(s) or a single I2C slave device.
J3	Screw terminal	Arduino Analog pins (A4, A5)	1	I2C sensors	I2C sensors	SDA = A4, SCL = A5. VCC, GND terminals available. Non-Multiplexed channels.
Nano	Female header	D0, D1	-	Unavailable	Unavailable	Used to control Rx/Tx
Nano	Female header	D2, D3, D4	-	Unavailable	Unavailable	Pin used to control thermistor multiplexers (2)
Nano	Female header	D5, D6, D7, D8, D9, D12, D13	7	Available	Available	Digital pins available for temporary circuits
Nano	Female header	D10	-	Unavailable	Unavailable	Pin used to enable A MUX
Nano	Female header	D11	-	Unavailable	Unavailable	Pin used to enable T MUX
Nano	Female header	A2, A3, A6, A7	4	Available	Available	4 analog pins available for temporary circuits

*All analog pins (Ax) can be defined and used as digital pin (Dx). The opposite is not possible.
*Unused pins from the Arduino female headers can be used for user custom features.

PRODUCT FEATURES

- Low-cost portable electrical multi probe data logger
- Designed for scientists, engineers and researchers
- Open-source software and hardware
- Arduino-compatible technology
- Measured variables:
    - dry bulb temperature (°C)
    - wet bulb temperature (°C), static
    - mean radiant temperature (with self-made black globe) (°C)
    - electrical resistance (ohm)
    - voltage (0 - 5 V)
    - illuminance (lux)
    - relative humidity (%)
    - Wet Bulb Globe Temperature index (WBGT) (thermal comfort index) (°C)
- 20 channels:
    - 8 input channels for resistive probes measurements only (e.g. thermistors)
    - 2 channels for I2C sensors/devices:
        - Examples of commercial I2C devices: air humidity, accelerometer, magnetometer, temperature, pressure, FM transmitter, gyroscope, IR thermal camera, potentiometer, OLED display, current/voltage sensor, gas sensor, FRAM, capacitive sensor, Analog-to-Digital (ADC) converter.
    - 8 channels for 0-5 V sensors
    - 2 channels for control signals
- Small (TH-1) or large (TH-2) temperature probes available
- Factory-calibrated probes with an overall system accuracy of 0.5°C in the 0-100°C range (3-point calibration)
- Graph capability through a third-party software (e.g., Microsoft Excel®, LibreOffice ®, Python)
- Serial communication with PC (ASCII data)
- Timestamp provided by a Real-Time Clock (RTC) subcircuit (day, month, year, hours, minutes, seconds)
- Data storage, power and communication provided by the PC or Raspberry Pi
- Acquisition rate up to 340 S/s (depending on several factors)
- Free public access to the source code, calibration procedures, application notes
- Made in Canada
- Available on GitHub
- Ready-to-use
- Customizable

DIGITAL SENSORS AND I2C SHIELD

The RDL is able to communicate with digital sensors that use the standard I2C protocol. Since all sensors use the same single Arduino I2C bus, each sensor must use a different I2C address. This implies that two units of the same sensors cannot be used simultaneously, as they will both "talk" at the same time. You should also verify that each sensor added to the main bus does not have an address that is already used by some of the onboard components. For example, the I2C address of a DS1307 RTC clock is 0x68. Therefore, this address cannot be used by any other sensor.

If the I2C shield is used – and connected – the sensors will be multiplexed both in terms of power and signal, allowing for up to 8 identical devices/sensors to be connected to the system.

The I2C shield is an expansion that connects to the RDL via RJ45 connectors and a very short CAT cable. Each shield adds 8 I2C sensors to the main bus. You must reset the RDL after connecting any I2C device. Otherwise, it will not be detected. Also, if you disconnect an I2C sensor while using the device, you risk jamming the I2C bus, therefore interrupting indefinitely the RDL code. (The chip will keep waiting for an answer from the I2C device.)

For more details, please read the I2C Shield Design Overview.

REQUIREMENTS

- Computer with minimal hardware:
  - CPU: 1 MHz; Memory: 250 MB RAM (Minimum Hardware)
  - Internet Access with Internet Explorer 9 or higher
  - 1 USB port (2.0 or 3.0)
  - Operating system: Windows®7 and newer, Mac OS X®, Linux®, Android® (See details below)
- Wire-stripper for 22 to 28 AWG wire size

DISPLAY OUTPUT COLUMNS

Order into which the data is printed to the window or text file, from left to right. Each of these columns can be activated/deactivated separately. User judgement must be applied. For example, if asked to do so, the program will calculate and display thermistor temperatures even if no thermistor is connected.

1. Reception Time (HH:MM:SS.MMM) (Arduino IDE Timestamp);
2. Date (YYYY/MM/DD) (RTC Timestamp);
3. Measurement Time (HH:MM:SS) (RTC Timestamp);
4. Data Identification number (i.e. how many measurements done since last reboot);
5. Thermistor temperature (channels 1-8);
6. Resistance values (channels 1-8);

7. Relative humidity values (channels 1-8);
8. I2C sensors readings (demonstration device is AM2301b) (channels 1-2);
9. Voltage readings (channels 1-8);
10. Soil humidity readings (channels 1-8) (Temperature compensated individually);
11. Control signal (0-5V).

QUICK START: MY FIRST EXPERIMENT USING THE RDL

This procedure will allow you to do a quick basic test of the RDL with the factory settings. It does not require modifying or compiling the source code. This will guide you through some measurements of an espresso machine brewing a cup of coffee. If you do not have access to a coffee machine, you can also use hot water.

Required time: 30 to 90 minutes.

Required material:

- RDL revE4
- 3 thermistors with lead wires
- Access to a computer, an Internet connection and a spreadsheet software (e.g. MS Office, LibreOffice).
- Access to an espresso machine, a cup and scotch tape.

1- Insert a CR1220 battery (3V) in the battery holder of the RDL. The PLUS (+) sign of the battery should be facing up. (No battery is included with the package to reduce shipping restrictions). In the absence of a battery, the clock will not behave properly and might display wrong values.

*To finely synchronize the RTC timestamp with the official time, follow the instructions in the section titled “RTC timestamp adjustment”. This is not required for a first test.

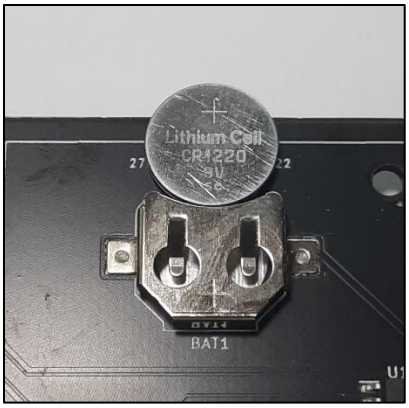


Figure 2 Inserting the CR1220 battery into the battery holder

2- Connect the USB cable from the RDL to the computer. The PWR (power) red light on the RDL should light up.

3- If this is not already the case, connect each required thermistor to its dedicated channel with the provided screwdriver. For this quick test, three probes are required. Other probes can remain connected without problems. Thermistors have no polarity, so you can invert the two wires without consequence. Make sure that the thermistor ID is coherent with the channel in the source code (see **Error! Reference source not found.**). Make sure that the terminal contact is solid by pulling slightly on each wire. If using extension wire, the wire gauge should be AWG26 or larger to ensure a solid and easy connection to screw terminals. Smaller gauge can be difficult to use. The silkscreen indicates the wire gauges that are acceptable (i.e. AWG 20-26).

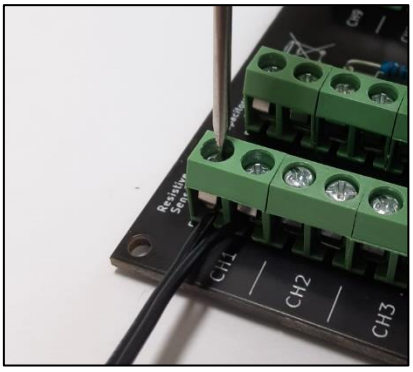
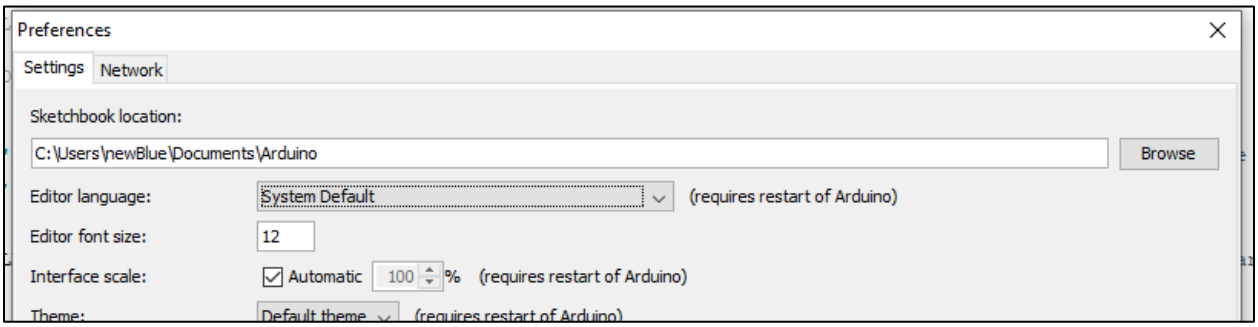


Figure 3 Connecting the thermistor probe wires to the dedicated channel C1

4- Download the free Arduino Software IDE for your operating system on the Arduino official website (www.arduino.cc). Follow the instructions provided by the website. Since the Arduino IDE 2.0 is fairly recent, we recommend using the legacy version (1.8.19) for now. It is also easier to follow along our guide.

Figure 4 Arduino official website software download section

5- Launch the IDE software. We suggest temporarily using the English version to easily follow the User Guide instructions. The software launches in your operating system default language, but you can modify this in File/Preferences/Editor Language. You will need to restart the software.



6- Modify the configuration settings. Go to Menu -> Tools.
   a. Board -> Arduino Nano
   b. Processor -> ATMega328P (Do not choose “Old Bootloader” as this will generate an uploading error)
   c. Port -> COM4. The exact port name might vary depending on your system. (e.g., ttyUSB1, ttyUSB2, COM1, etc.) You can disconnect/reconnect the USB cable to see which port name appears when you connect the RDL.

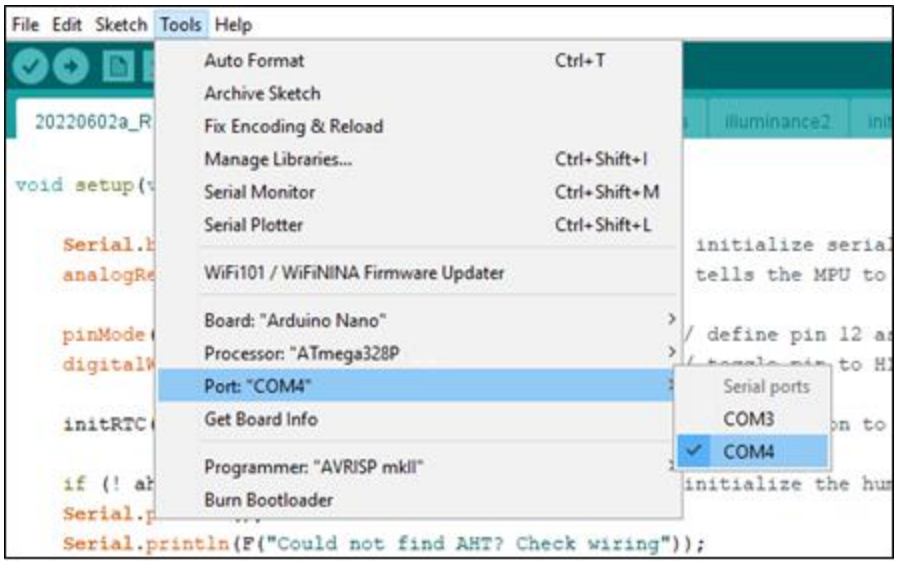


Figure 5 Serial port selection through the Arduino IDE menu

7- Go to Menu -> Tools -> Serial Monitor. A new window should pop-up. At first, the data being displayed on your screen will be gibberish, because the baud rate (i.e., communication rate) is wrong (default is 9600).

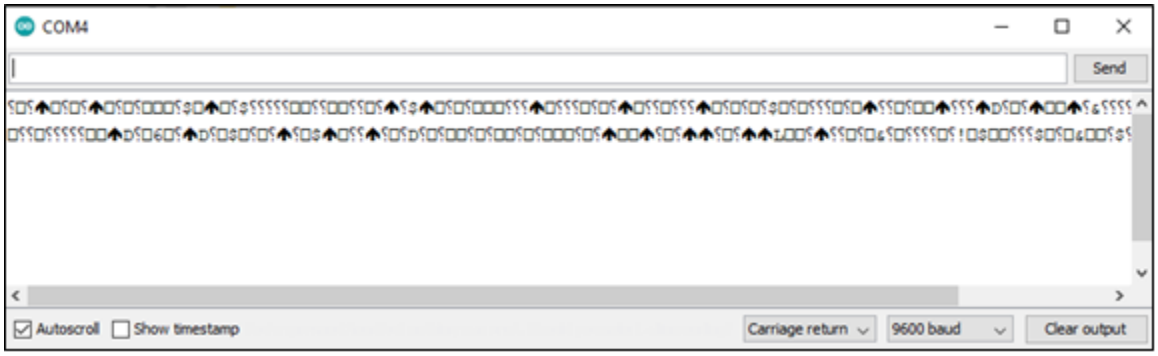


Figure 6 Example of gibberish data coming in due to a mismatch of the baud rates

8- The baud rate inside the RDL source code should match the baud rate of the computer. On the bottom right of the Serial Monitor window, set the baud rate to 57600 (program default is 9600). Once the baud rate is correct, readable data will start coming in through the Serial Monitor. It is suggested to 'CLEAR OUTPUT' with button of the same name to keep the window clean when comes the time to copy/paste.

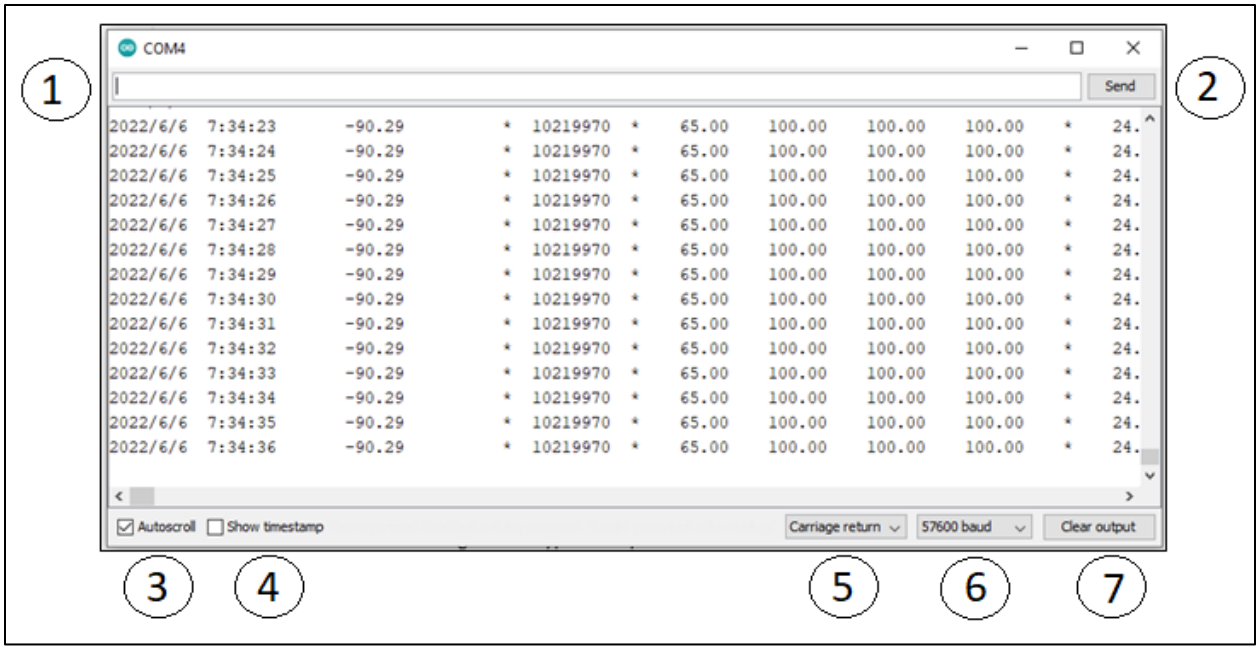


Figure 7 Serial Monitor window components. 1) Command line; 2) Send command button; 3) Activate/Deactivate autoscroll; 4) Show/Hide timestamp; 5) Selection of the end of line symbol; 6) Selection of the baud rate; 7) Clear the window.

9- Select the 'Carriage return' in the End of line options (item 5 in Figure 7). Otherwise, the command will not be recognized by the RDL.

10- The default factory setting for the acquisition rate is 1 Hz (i.e., 1000 milliseconds interval). You can modify it to 2000 ms or leave it at 1000 ms. To modify that value, you must enter the text "INTERVAL" in the command line (item 1 in Figure 7).

11- If there are too many columns, you can zoom out by using CTRL + the mouse wheel. 'Autoscroll' can be deactivated to look at the data.

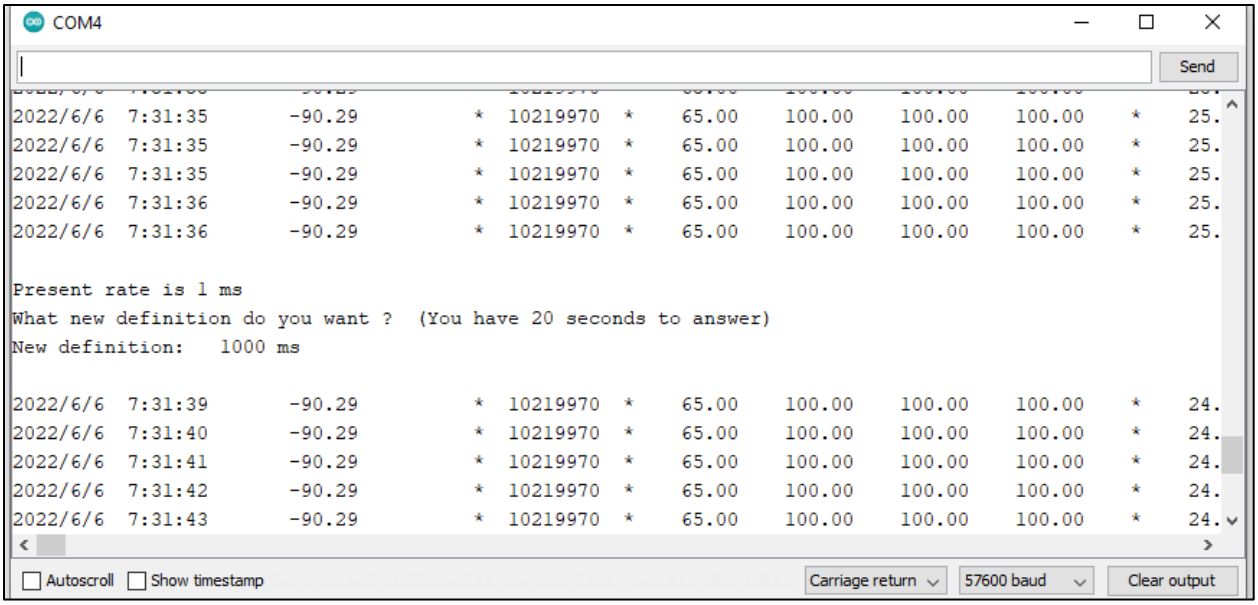


Figure 8 Example of an interval modification from 1 to 1000 ms through the command line

12- Before starting your experiment, you can clear the window by clicking on “Clear output” in the bottom right of the window (item 5 in Figure 7). Otherwise, non-relevant data can be erased later during post-treatment.

13- You can now proceed with your experiment. Meanwhile, using adhesive tape, position the three probes in the following way.

   a. Probe C1: Near the liquid exit to measure the temperature of the coffee outflow;
   b. Probe C2: Near the bottom of the cup, without touching the bottom surface, if possible. This will allow measuring the global temperature of the coffee accumulated in the cup.
   c. Probe C3: In the center of the top surface of the espresso machine. This surface becomes hot during the warm-up and operation phase since the boiler is right beneath the surface. It can be interesting to see how its temperature evolves over time.

14- Prepare the coffee as usual, and turn on the machine to preheat. Wait until the temperatures stabilize on your screen before activating the compressor (brewing phase).

Figure 9 Caption

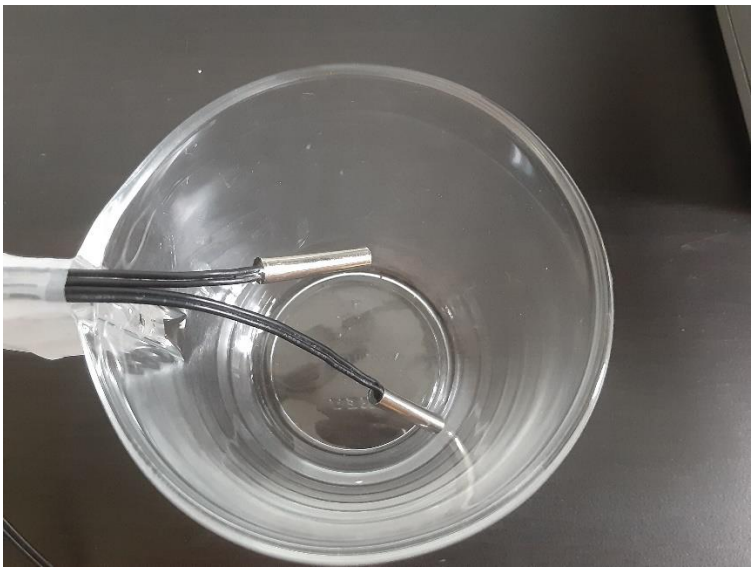


Figure 10 Position of the two probes inside the empty cup

15- As the experiment unfolds, watch the data accumulate in the Serial Monitor window. Each time, the RDL sends data, the light 'TX' on the controller should blink very quickly.
16- When you have gathered all your data, untick the case 'Autoscroll'. This will prevent the screen from moving while you select your data. Do not close your Serial Monitor window until you have saved your data in another reliable location.
17- Press "CTRL + A" to select all text. Then copy by pressing "CTRL + C".
18- Open a text editor and paste all text by pressing "CTRL + V".

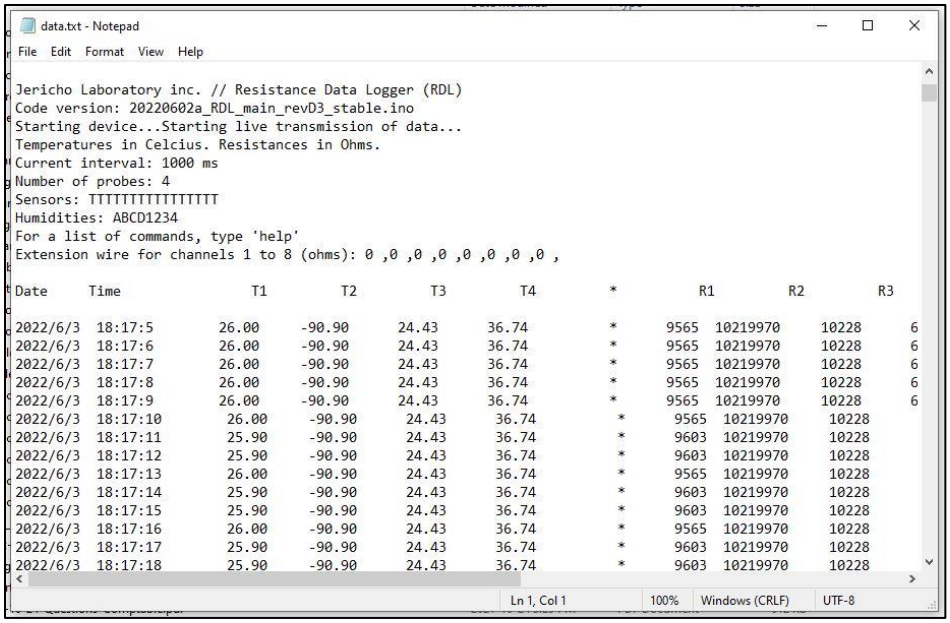


Figure 11 RDL text file output sample

19- Save the text file on your Desktop (ex: test_1.txt). This is an important step, since the data within the Serial Monitor is not recorded permanently. Data is lost when the window closes.

20- Open Microsoft Excel or, alternatively, a free spreadsheet software (e.g., LibreOffice Calc).

21- Go to Menu -> Data -> From text. Select the text file. In some recent versions of Microsoft Office, you might have to activate “Legacy Wizards” in order to have access to data import from a text file. If this is your case, follow Microsoft instructions on the topic on their forum.

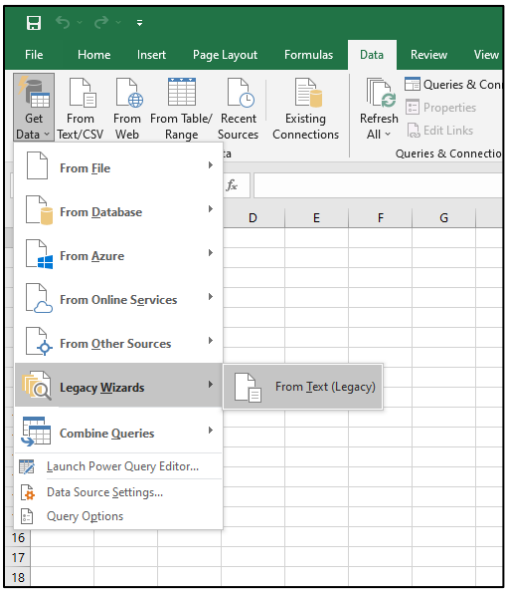
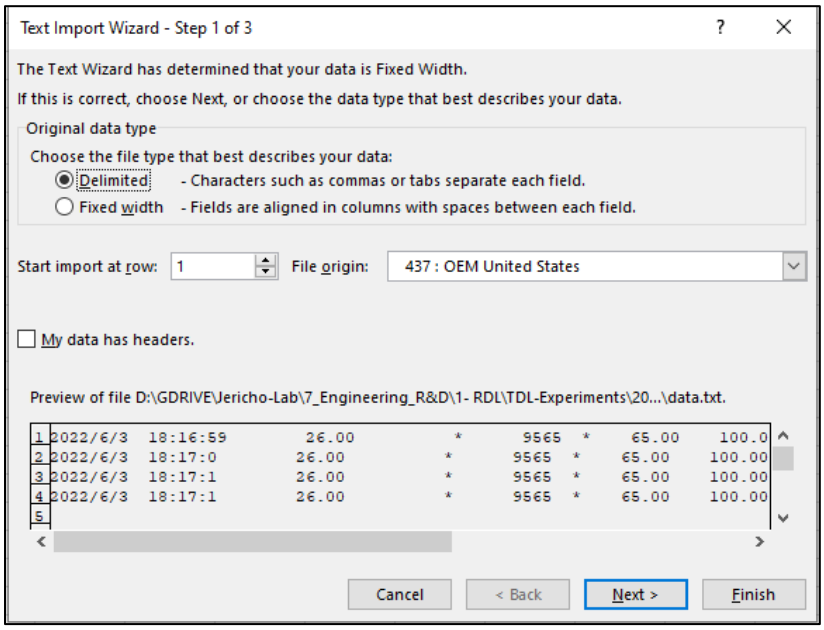
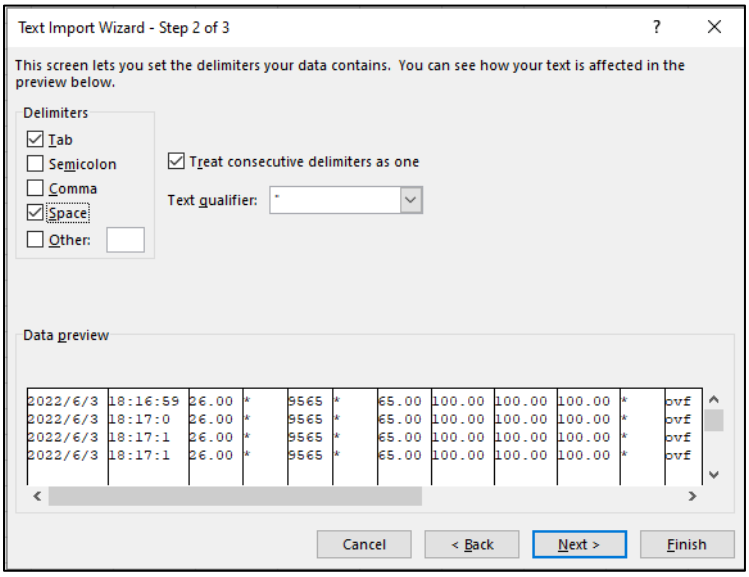


Figure 12 Microsoft Excel menu for the Text Data Import

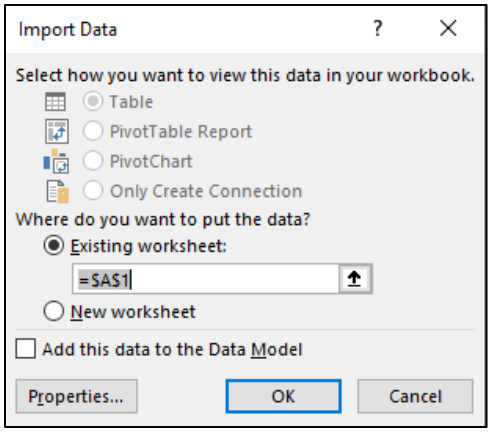
22- Select the ‘Delimited’ data type.



23- Select the ‘space’ character as the delimiter. Tick the box ‘Treat consecutive delimiters as one’. Click Finish.



24- Import data to ‘Existing Worksheet’.



25- If necessary, adjust the 'timestamp' column cell format to 'Short Date' in order to display the time properly.

26- You can now create various graphs from your data. For example, you might create a "Line graph" with the column 'Time' as the x-axis and three temperatures as the y-axis. In our case, we decided to put data from channel C4 on a secondary vertical axis.

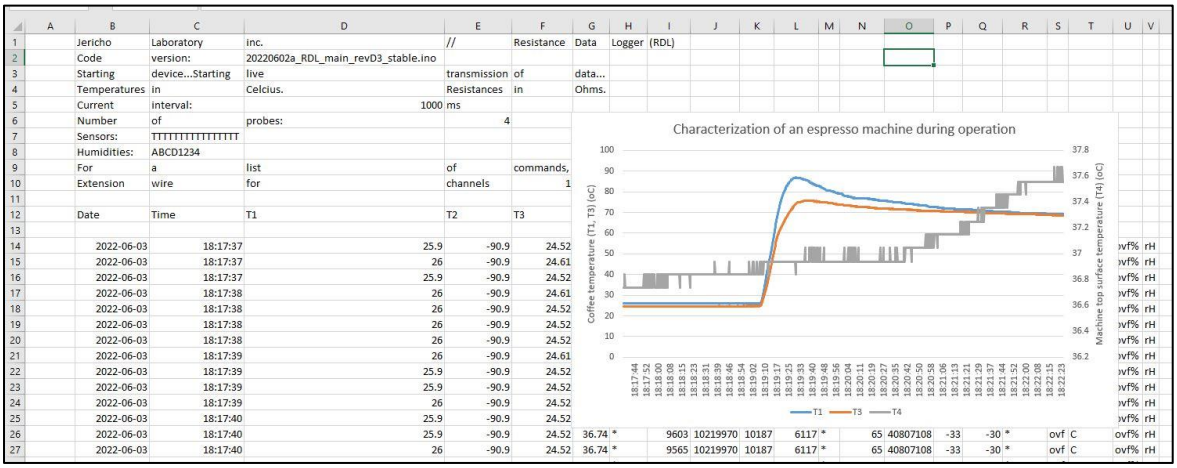


Figure 13 Data treatment inside Microsoft Excel

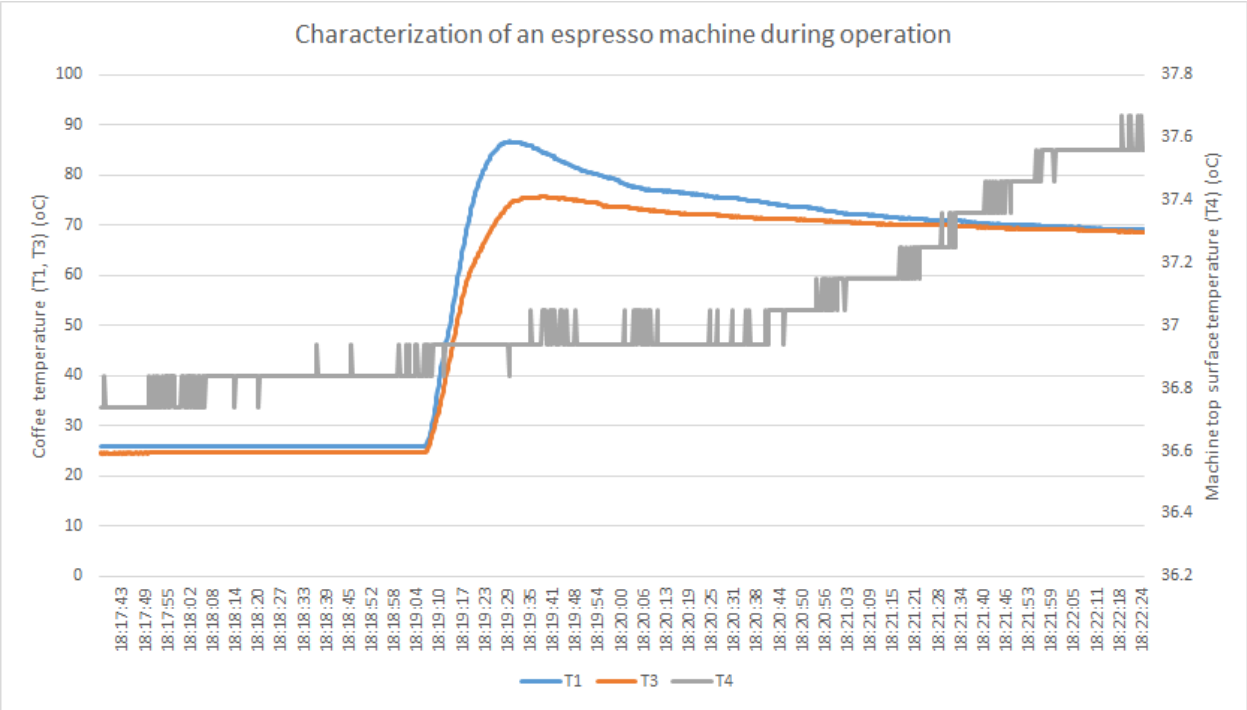


Figure 14 Example of data post-treatment with MS Excel ® for the espresso machine experiment

27- Analyzing the data is the final and most interesting part of the experiment. First, it is good practice to analyze the quality of the data. Are there any surprises, irregularities, noise in the data? Second, what are the trends that we observe in the data (peaks, valleys, plateau, oscillations, etc.)? Here are a few examples for the espresso machine graph.

d. **The temperature of the coffee outflow sensor increased very fast initially**. Due to the thermal inertia of the large sensor, it is not clear at what time the coffee temperature outflow really peaked. It would be interesting to rerun the experiment with a small probe, which has lower inertia. If the curve does not change, that would suggest that probe inertia does not play a significant role on this specific aspect.

e. **Both coffee temperature sensors peaked before the end of the brewing**. Since we know that the brewing lasted around 30 seconds, we can confirm that the heater element is not powerful enough to maintain the maximum temperature of the water. The maximum heating power [watts], the coffee grain thermal inertia and the water reservoir temperature are likely to play a role in the curve of the coffee temperature.

f. **There seems to be a slight inflexion point around 18:20:06**. This could be the time at which the brewing stopped. We could improve our observation by plotting a first order time derivative of the data for C1 and C3. It would be easier to see if our eye is being misled. That could be confirmed with a second experiment which measures the start and stop of the compressor. This could be measured in one of several ways: manually with a clock, with a current sensor, or with a digital switch.

g. **The two sensors eventually converged to a very similar temperature.** This is likely to indicate that after a few seconds without liquid inflow, the stratification became negligible due to conduction and convection within the cup.
h. **There was very little change in the temperature of the top surface (less than a degree)**. This was a surprising result as one could expect that the incoming cold water from the reservoir would have had some effect on the machine casing temperature. Also, with the help of the secondary axis, we can see that much of the temperature increase happened after the brewing. This challenges our understanding of the espresso machine.
i. **The top surface temperature had not peaked when the experiment ended.** It would be interesting to run the experiment longer for that reason.

As you can see, much can be learned from a single graph. A first experiment often leads to a series of more refined experiments.

28- We advise you to keep the original data (i.e., text file) in case you eventually want to restart your analysis from scratch for a variety of reasons.

RDL FACTORY SETTINGS

Factory settings are the settings active when you first receive your device. Most of the RDL capacities are active in order to easily demonstrate every aspect of the product. The factory settings are a combination of memory parameters and source code parameters. You can alter the factory settings by combining serial commands and small source code alterations. You can always go back to factory settings by using the command “EEPROM-ERASE”.

Factory settings (EEPROM section)

- Acquisition interval: 1000 mS
- Number of probes: 16
- Temperature unit: Celsius

Factory settings (source code section)

- Time display: Yes
- ID display: Yes
- Temperature display: Yes
- Resistance display: Yes
- I2C display: Yes
- Voltage reading display: Yes
- Soil humidity display: Yes
- Wet Bulb Globe Temperature (WBGT) display: Yes
- Steinhart-Hart thermistor coefficients (A,B,C):
    - For the Starter Kit package

- Channels C1 to C8: Custom coefficients specific to included thermistor probes
- Channels C9 to C16: Generic coefficients for 10k NTC thermistor

- For the Controller-only package
  - Channels C1 to C16: Generic coefficients for 10k NTC thermistor

AVAILABLE SERIAL COMMANDS

In order to interact with the RDL, the user must type in some of the following keywords at the top of the 'Serial Monitor' window and then click 'Send'. The option 'Carriage return' must be selected for the RDL to detect the command. The commands are not case-sensitive.

HELP: The command 'HELP' prints out the available serial commands: "CELCIUS, FAHRENHEIT, KELVIN, RESET, EEPROM-ERASE, INTERVAL, QTY-T, QTY-V, I2CSCAN". Activity will resume after a few seconds.

INTERVAL: The command 'INTERVAL' allows adjusting the measurement interval (i.e., acquisition rate). The same rate applies to all sensors. After receiving the command, the program will wait for the user to enter a value between 1 and 96 400 000 ms (i.e., 24 h). Enter the value '1' to force maximum speed, which will vary depending on the number of active probes. The default factory value is 1000 ms (1 second).

The maximum acquisition rate depends on a combination of factors:
- amount of sensors/values/data printed per poll
- baud rate (source code default is 57600, but the Nano can reach 115200)
- RTC use (DS3231MZ performance degrades above 57600)
- Speed of the PC or Raspberry Pi (buffer overload)

Note: For intervals shorter than 1000 ms, we recommend activating the serial monitor native timestamp, which includes milliseconds. Otherwise, all datapoints taken within the same second will receive the same timestamp from the RTC (hh:mm:ss), which can be problematic for plotting time series afterwards. You can keep the RTC timestamp though, as the native timestamp does not include the date.

QTY-T: The command 'QTY-T tells the device how many thermistor channels must be active. Value must be between 1 and 8. The lowest channels value are activated first (e.g. A value of 3 will activate channels 1 to 3). The parameter is stored in the permanent memory and will be remembered after a shutdown or reset of the device. A quantity of eight (8) probes is the default factory setting. No confirmation or further instructions are required. Operation will resume after a few seconds.

QTY-V: The command 'QTY-V tells the device how many voltage measurement channels must be active. Value must be between 1 and 8. The lowest channels value are activated first (e.g. A value of 3 will activate channels 1 to 3). The parameter is stored in the permanent memory and will be remembered

after a shutdown or reset of the device. A quantity of eight (8) probes is the default factory setting. No confirmation or further instructions are required. Operation will resume after a few seconds.

CELSIUS: The command 'CELSIUS' tells the device to display temperatures in Celsius (°C) units. Unit is stored in the permanent memory (EEPROM) and will be remembered after a shutdown or reset of the device. Celsius is the default factory setting. No confirmation or further instructions are required. Operation will resume after a few seconds.

FAHRENHEIT: Display temperature in Fahrenheit (°F) units. No confirmation or further instructions are required. Operation will resume after a few seconds.

KELVIN: Display temperatures in Kelvin (K) units. No confirmation or further instructions are required. Operation will resume after a few seconds.

I2CSCAN: Scan each of the eight (8) I2C channels from the I2C shield. For each channel, the command will print out the I2C addresses for which a device has been found. If a device is connected on the main hub (not multiplex), it will appear in all each channel.

EEPROM-ERASE: Erase the EEPROM memory and return to the factory settings (RATE 1000 ms, Celsius units, all 16 channels active, etc.). No confirmation or further instructions required. Operation will resume after a few seconds.

RESET: This command restarts the microcontroller. It is the equivalent of pressing the RESET button on the Arduino Nano. This is convenient to print out the header when starting a clean measurement session for logging purposes. Operation will resume after a few seconds.

- - - - - - - - - - - - - - - -

IMPORTANT USER PARAMETERS

Most variables within the source code should not be modified by the user during normal operations, but some important parameters are not available through the serial commands and need be modified within the source code before uploading it to the microcontroller. These parameters are located at the top of the main file.

Among the following list, each parameter that contains the word 'display' can be activated (1) or deactivated (0) by giving it a value of 1 or 0. Parameters that are not measured are not displayed, and vice-versa.

- **Header Display (headerDisplay)**
    - This parameter controls the display of the columns headers, which is adapted to the currently active parameters and sensors.

- **Time display (timeDisplay)**
    - The Real-Time Clock allows the RDL to keep track of the date and time with high accuracy. The RTC clock is polled at every cycle and its data is printed out in the serial monitor. This is especially useful for experiments that last more than 24 hours since most Serial Monitor Data Logging software do not keep track of the date. The RTC poses a restriction however in the maximal speed that the RDL can operate (approximately 5 Hz).
- **Identification Number display (idDisplay)**
    - By default, the source code attributes an identification number to each measurement (i.e., 1, 2, 3, etc.) The ID can supplement or substitute the timestamps in some cases, such as high-speed measurements. It is also useful during post-treatment to keep track of the data continuity (i.e., lines 2450 to 2460 are missing due to bad data)
- **Temperature display (tDisplay)**
    - Activate or deactivate the display of temperature measurements. Deactivating the temperature display can be useful when the user is not using the multiplexed channels or is not using them with thermistors or photoresistors. In that latter case, the user might only activate the display of the resistance values.
- **Resistance display (ohmDisplay)**
    - Activate or deactivate the display of the resistance value for the multiplexed channels. Deactivating the resistance display can be useful in cases where the user is not using the multiplexed channels or is only interested in temperatures. The resistance and temperature values can be displayed independently.
- **pH reading display (phDisplay)**
    - Activate or deactivate the display of pH measurements.
- **Strain measurements display (strainDisplay)**
    - Activate or deactivate the display of strain measurements.
- **Air humidity display (SHT40Display)**
    - Activate or deactivate the calculations and display of relative humidity via SHT4x sensors.
- **Current sensor display (currentDisplay)**
    - This parameter controls the display of current sensor values. By default, the current sensor value will try to add temperature compensation. However, if the parameter tDisplay is deactivated, the function will use 0°C as a temperature compensation, which will add a significant bias to the current sensor values.
- **Soil humidity sensor display (terosDisplay)**
    - This parameter controls the display of soil humidity sensor values (Teros 10).
- **Voltage display (voltDisplay)**
    - Activate or deactivate the measurement, calculation and display of the voltage input in the corresponding channels (0-5V, DC current).

Parameters for controlling the active channels

The following parameters allows the selection of channels which contain an I2C sensor for each type.

- Active channels for SHT40 measurements (i2cChannels_sht40)
- Active channels for strain measurements (i2cChannels_strain)
- Active channels for pH measurements (i2cChannels_ph)
- Active channels for electrical current measurements (channels_current)

For example, the following line “int i2cChannels_strain[] = {1,4};” will tell the RDL that channels 1 and 4 contain a strain measurement sensor (NAU7802). To deactivate the readings, you can either leave an empty list (i.e. {}) or deactivate the measurement altogether (e.g. “strainDisplay = 0”) Note that the channels are numbered from 1 to 8. Do not use the value 0. It is also possible to use the same channel with multiple sensor types. This could occur with NAU7802 (two functions) or daisy chains (two sensors on the same channel).

HOW TO MODIFY AND UPLOAD THE SOURCE CODE TO THE RDL

The RDL already comes ready-to-use, but you might want to make some changes to the source code or use the latest release.

1- If this is not already done, do steps 1 to 6 inclusively from the Quick Start section.
2- Download the latest source code on the Github repo (main branch).
3- Copy the files to your working directory. The folder name should be the same as the main file without the file extension (e.g., 20220602a_RDL_main_revD3_stable)
4- Open this code file with the Arduino IDE software. (File -> Open)
5- Make the desired modifications to the code.
6- Make sure that the code can compile. Go to Menu -> Sketch -> Compile.

Figure 15 Example of the source code opened inside the Legacy Arduino IDE software

7- Connect the RDL to the computer via the USB cable.
8- Make sure that the proper USB port is selected by the software.
9- Install the required libraries listed below. This only has to be done once on each computer. Go to Menu -> Manage Libraries. This is necessary even if you don't plan on using some of the features (e.g., I2C humidity sensors) due to their presence in the source code, even when the features are not activated. EEPROM.h and Wire.h are included in the source code but is not required to install since it is native to the IDE software.
    a. RTClib.h by Adafruit
    b. SparkFun_Qwiic_Scale_NAU7802_Arduino_Library.h
    c. Adafruit_SHT4x.h
    d. Adafruit_ADS1X15.h
    e. Adafruit_PCF8574.h

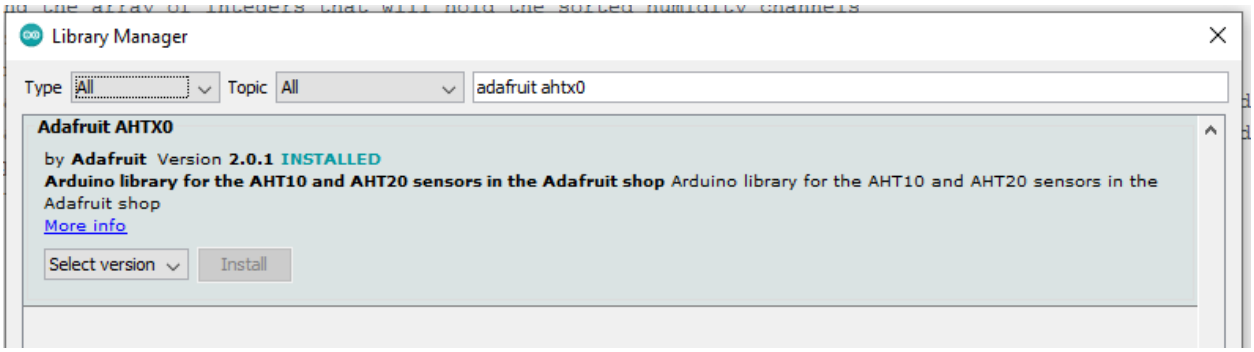


Figure 16 Searching the Adafruit AHTX0 library within the Arduino IDE Library Manager

10- Go to Menu -> Sketch -> Upload. This will compile then immediately upload the code to the RDL. The "RX" and "TX" LED light should flash quickly while the code is being uploaded. Normal upload time should be 10 seconds. Normal compile time should 10 to 15 seconds. Total time should then be 20 to 25 seconds.
11- Follow the steps from the Quick Start guide to open the Serial Monitor and see if you get the expected results.

For more help about Arduino coding and debugging, visit the official forum on arduino.cc or ask your local community: the Arduino platform is very common worldwide.

HOW TO LOG DATA TO FILE

The Arduino IDE can hold large quantities of data inside the serial monitor window. However, this data is kept in memory temporarily, and eventually the data would have to be copied manually to a text file for permanent storage. There is always a chance that the computer or program would shut down before doing that operation. It is therefore best to save data as we go to a text file. There are two preferred ways to log the data, depending on your needs.

1) Run Python scripts (Compatible with all OS)
2) Setup Services/Daemons (Compatible with Linux only for now)

Python scripts can run on all platforms that have Python 3 interpreter and the required packages installed. The following two scripts are available:

- WIRED-logging-RDL-USB0.py (Continuously logging incoming RDL data to a text file)
- WIRED-syncing.py (Continuously syncing files to Dropbox account)

Note that the syncing script is also able to sync the data from the WIRED project, which includes thermal and surveillance cameras. These two devices each have their own Python Script.

A shell script (WIRED-runAll.sh) is able to launch all Python scripts. This mode however is not resilient to failure and reboots. You must start the scripts manually each time.

Services, also called “Daemons” in the Linux environment, are the most resilient way to log data from the RDL. When set up, they will make sure that each Python script is launched at each startup and will try to execute again in case of crash or interruption.

Important: Whatever the method chosen, it is important that the computer does not sleep. When going to sleep (e.g., closing the lid) the data logging is interrupted. The USB voltage supply is maintained but the serial communication is interrupted.

For detailed information about the use of these methods, please read the WIRED Python Scripts Design Overview and the WIRED Daemons Design Overview.

NOISE CONTROL

Electromagnetic (EM) noise control is an important part of signal measurements. The amount of noise present depends on several factors among which the prevalence of EM noise in the environment and the length of the sensor lead wire. Long lead wires – several meters - tend to act as antennas and are likely to pick up strong EM noise even in low noise environment such as a typical home.

No hardware filter is used in Revision E4. Experiments have shown that hardware filters like RC filters are not well suited to multiplexed measurement channels. Since the multiplexer only polls each channel for a limited amount time, the condensator needs time to equilibrate each time. Software filters are preferred instead.

A software filtering is simpler to implement than hardware filtering: it consists of modifying the value for the parameter ‘NUMSAMPLES’ in the source code. For maximum speed, use value of 1. For heavy filtering, you can use values of 5 to 20. Since the Nano ADC takes 15 ms per reading, a value of 20 would delay by at least 300 ms per channel. For 16 channels, that would mean a delay of 4800 ms, without accounting for the processing time (i.e., arithmetic averaging). This would limit your ability to gather data at a faster interval than 5s.

OPEN-SOURCE LICENSES

Jericho Laboratory operates the RDL suite of products with the following open-source licenses:
SOFTWARE: GPL 2.0
HARDWARE: MIT
WEBSITE, DOCUMENTS: Creative Commons CC-BY-NC-SA (Non-commercial use only)

For more information about the user permissions and responsibilities associated with each license, please refer to our website or contact us via email at info@jericholab.com.

CONTACT

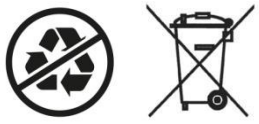
For more information, tutorials, FAQ, software updates, accessories and replacement parts, please visit us at jericholab.com. We also appreciate you sending comments and suggestions at info@jericholab.com.

ABOUT JERICHO LAB

Jericho is a company based in Montreal, Canada. Our mission is to improve access to scientific equipment by providing provide low-cost high-quality products to students, scientists and engineers. The Rose of Jericho (*Anastatica hierochuntica*) is a desert plant that can survive death by desiccation (absence of water).

ABOUT ARDUINO ©

The RDL is compatible with the Arduino language, which is based on the 'Processing 3' language, which itself is similar to the C language. Most of the C/C++ core language can be used with the RDL device and the Arduino IDE software. You can create classes, use inheritance, composition, and other functionalities, but STL and exceptions cannot be used. Arduino first appeared around 2005 and is now used worldwide by over a million users. Jericho is not endorsed nor linked in any sense with the Arduino company. Arduino clones are produced by hundreds of manufacturers around the world, with different level of qualities.



Please do not throw away this product to garbage or recycle bin as it can contain toxic substances and/or heavy metals. Please send your Jericho product to your local electronic waste drop-off site so that it can be disposed of properly.

FREQUENTLY ASKED QUESTIONS (FAQ)

- What is a thermistor?

A thermistor is a tiny passive device which has an electrical resistance that varies with the temperature of its material. It is a word play with the words ‘thermal’ and ‘resistor’.

- ***Can I use the RDL to do something else***?

Yes, you can modify the product, software and hardware to add or remove capabilities. However, any physical modification of the device will void the warranty.

- ***I dropped the RDL in the water, what should I do?***

Water will create short-circuits that will render the device unusable temporarily. Put it in dry uncooked rice with closed container for 24 to 48h to remove humidity, like you would do for a smartphone. It should work again.

- **What other software than Arduino Software can I use to control the RDL device?**

The IDE software provided by Arduino is specifically designed for Arduino and Arduino-compatible devices and provides an easy interface to beginners and advanced users. However, some users might prefer to use more professional programming tools. A free option that is often recommended is the software Atom, combined with the Platformio add-on.

- I broke some component. How do I find replacement parts?

You can consult the Bill of Material (BOM) to find the exact part number and order a new part online. If you believe this is due to a manufacturing defect and you are still covered by your warranty, contact us by email.

- Should I tin the wire endings to be used with screw terminals?

No, NASA procedures for example require to NOT thin the wires due to different thermal expansion coefficients of the wire and solder.

- How can I modify the column title that is being printed out? For example, how can I print units “g/mm3” beside the word ‘soil’?

In the source code, you can modify the column title in the function “printHeader”. You would change the String content from “soil” to “soil [g/mm3]”.

TROUBLESHOOTING

- **Error: "An error occurred while uploading the sketch. (avrdude: stk500_getsync() attempt 10 of 10: not in sync: resp=0x31)**
    - This can happen when the RDL was busy sending sensor data when the PC tried to communicate. It happens more often after a change of USB port.
    - Try reuploading the program.
    - Try to press once the RESET button on the Nano during the upload phase of the Arduino IDE.
    - Try disconnecting the USB cable and reconnecting, then reuploading.
    - Try another USB port of the PC.
    - Try to restart the program.
- **Error: "Board at COM4 is unavailable."**
    - The selected serial port is not detecting the RDL/Arduino.
    - Verify that the USB cable is connected to both the PC and the RDL.
    - Verify that the corresponding port is selected in the Arduino menu.
- **The printed output of a channel makes no sense. Example, the temperature of the channel C2 is -273.15°C.**
    - Negative, random-looking values are usually associated with empty or opened channels (false contact, broken probe). Verify that there really is a sensor connected to the channel and that the connection is solid.
- **Error: "Variable X was not declared in this scope"**
    - Two very common reasons for this error is 1) a function was indeed declared at the wrong place 2) a missing bracket or semi-colon has created a complete fuckup of the entire code. It can be hard to find the exact spot. Undo the latest change or go back to your previous working saved version.
- **"My Arduino has stopped responding to 'uploads' via the Arduino IDE"**
    - In case the Nano stops responding, you can try to reinstall the bootloader. This kind of error can happen if a code upload has been interrupted. The bootloader is a very small program that allows the Nano to communicate and receive a script. To reinstall the bootloader, follow the instructions here: https://docs.arduino.cc/built-in-examples/arduino-isp/ArduinoISP.

USEFUL REFERENCES

- Arduino Uno Rev3, Digikey, https://media.digikey.com/pdf/Data%20Sheets/Arduino%20PDFs/A000066_Web.pdf
- ATmega 328p datasheet: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
- Free Arduino tutorials, https://www.tutorialspoint.com/arduino/
- Blum, Jeremy. 2019. *Exploring Arduino: Tools and Techniques for Engineering Wizardry*. John Wiley & Sons.
- Learn: Basic knowledge about principles and techniques behind the Arduino ecosystem. https://docs.arduino.cc/learn
- Build your own data logger (free course with videos). FreakLabs: Hardware for conservation. https://freaklabs.org/learn/build-your-own-data-logger/overview/