# Arduino Programming: Reading data from devices

Measure information from your environment using simple digital devices and use the data to control hardware

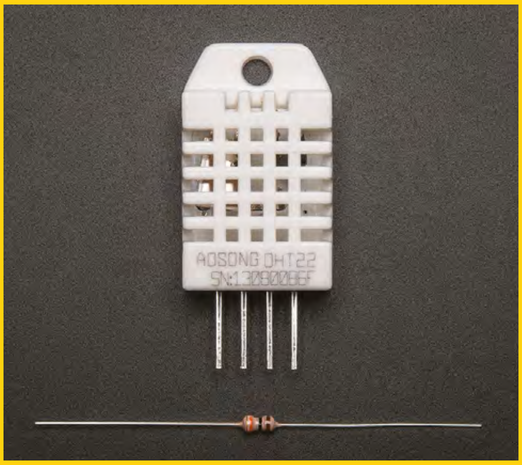


**Figure 1** The DHT22 is available from sellers including Adafruit, who package it with the resistor needed to build the circuit hsmag.cc/OBhmYH

**Do you want to read information about your environment using a digital temperature and humidity sensor, and drive the status of an LED based on the current temperature?** Well, let's go! This project expands on an existing sample application available online using the simple, inexpensive DHT22 sensor.

In previous articles, we've shown how to read both analogue and digital data values from connected devices. Most digital devices transmit more than a simple on/off value, streaming numeric or string values as a series of digital bits. In this scenario, the sending device varies its output in a pre-defined pattern that both the Arduino and the DHT22 understand. The device sends each data bit one by one until it has sent everything. It sends the appropriate value (high or low, depending on the bit value), then waits before sending the next. On the receiving end, the Arduino uses special code that reads those values.

To read this data, your Arduino needs to know the particular pattern of digital values the device sends. Fortunately, there are many libraries available, so most of the time, you can communicate with devices without having to do the low-level code yourself.

We'll use the DHT22, a widely available, inexpensive temperature and humidity sensor. Arduino sketches retrieve environment data values from the device using a library provided by the folks at Adafruit. The device provides four electrical connections (only three of which are used); its connections, by pin number from left to right in the figure, are:

1. **V+**
2. **Data out**
3. **Not connected**
4. **Ground**

When you connect the DHT22 to an Arduino, wire the DHT22's pin 1 to a 3V or 5V power source, pin 2 to a digital input on the Arduino, and pin 4 to the Arduino's ground. For this project (and associated source code), we'll connect the data output (pin 2 on the DHT22) to the Arduino's digital input 2.

With those connections in place, it's time to start looking at the code that reads sensor values. Before you use the sensor, install additional libraries in the Arduino IDE. The folks at Adafruit provide two libraries that you'll need in order to talk with the sensor.

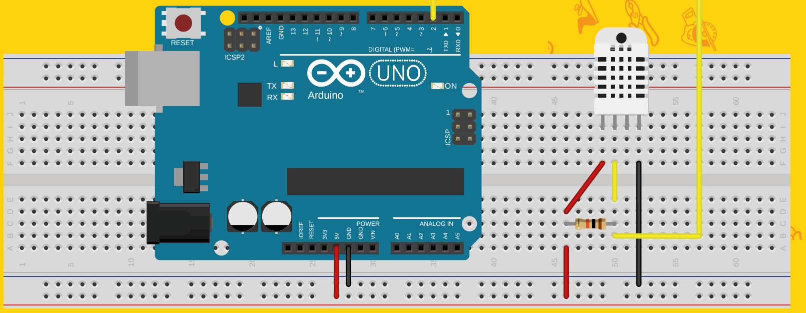
## ADDITIONAL CAPABILITIES

Let's do the manual installation first. Adafruit's DHT sensor library and example files are stored in a GitHub repository at **hsmag.cc/wyGYGY**. The Arduino IDE installs all libraries in a common folder; any Arduino libraries you place in folders in that location are automatically loaded by the IDE on startup. To determine the location where the IDE stores its libraries, open the IDE's preferences dialog by opening the application's File menu and selecting Preferences from the menu. The IDE will open the preferences pane shown in **Figure 3.** At the top of the Settings tab in the dialog is an input field labelled Sketchbook Location; this is where the IDE stores its libraries folder.

John Wargo

@johnwargo

John is a professional software developer, writer, presenter, father, husband, and geek. He is currently a Program Manager at Microsoft, working on Visual Studio Mobile Center. You can find him at **johnwargo.com**



On my system (which is the example shown below in **Figure 3**) sketchbooks are stored in the **d:\dev\hardware\arduino** folder. Arduino libraries go in the libraries folder in the sketchbook storage location.

Note: The Arduino IDE loads libraries at startup, so before continuing, close the Arduino IDE now, or it won't recognise the library installed in the next step.

## INSTALL LIBRARIES MANUALLY

To install the library manually, download the library from the GitHub page or use the git command to clone the repository to your local system. To install the library files via download, click the green Clone or Download button on the repository page to download the files, then extract the files into the IDE's libraries folder for your system. After you've extracted the files, rename the extracted folder to DHT.

If you have git installed on your system, there's an easier way to extract the library files. Open a terminal window (or Windows command prompt), navigate to the IDE's libraries folder, and then simply execute the following command:

```
git clone https://github.com/adafruit/DHT-sensor-
library DHT
```

git will connect to the repository, download the files, then store them in a new DHT folder in the current location. When you're done, you should be able to execute the dir command and see the resulting DHT folder as shown in the figure.

The Adafruit DHT sensor library uses a library provided by Adafruit; this library is published through the Arduino library catalogue, so it's easy to install. Open the Arduino IDE then go into the Sketch menu, select Include Library, and then Manage Libraries…. In the Library Manager dialog that opens, enter Adafruit_Sensor in the search field, then locate the Adafruit Unified Sensor library highlighted in the figure. Click the Install button to install the library: **hsmag.cc/gpKgkR**.

With the libraries installed, load a sample application to measure temperature and humidity through the sensor. In the Arduino IDE, open the File menu, select Examples, then DHT Sensor Library. In the menu that appears, select the DHTtester sample application. Follow the instructions in the code to configure the sketch for your hardware configuration. If you used the DHT22 sensor connected to Arduino digital input 2, you're all set. If you have different hardware, you'll need to make some changes to the code.

## THE DHT EXAMPLE SKETCHES INSTALL WITH THE DHT LIBRARY

Before executing the sketch, open the IDE's Serial Monitor; to do this, open the Tools menu, and then select Serial Monitor. Now, upload the sketch to the Arduino and look for the temperature and humidity data displayed in the Serial Monitor window as shown in **Figure 4** on the next spread.

Now, it's time to do something with the temperature data we're collecting.

Imagine you're monitoring the temperature of a particular device and you want some sort of visual notification that the temperature's exceeded a threshold. You could do all sorts of things in this case: light an indicator light, sound an alarm, even send an email or text message to your phone. As we're just starting out with Arduino programming, let's do the easy one: lighting an indicator light.

To add an LED to the circuit, you'll need both an LED and a resistor; the resistor acts a current limiter, reducing the amount of current that passes through the LED. You could connect the LED directly, and eliminate the resistor, but the LED would burn out more quickly. In general, a 220 Ohm resistor is sufficience to protect the LED, but other values may be more appropriate. →

**Figure 2**
Complete Arduino Circuit using the DHT22

### YOU'LL NEED

- **An Arduino Uno or compatible device**
- **DHT22 Temperature and Humidity Sensor** adafruit.com/product/385
- **A 10 kΩ resistor**
- **One or more 5 V colour LEDs** and the appropriate resistors
- **Breadboard**
- **Breadboard jumper wires**

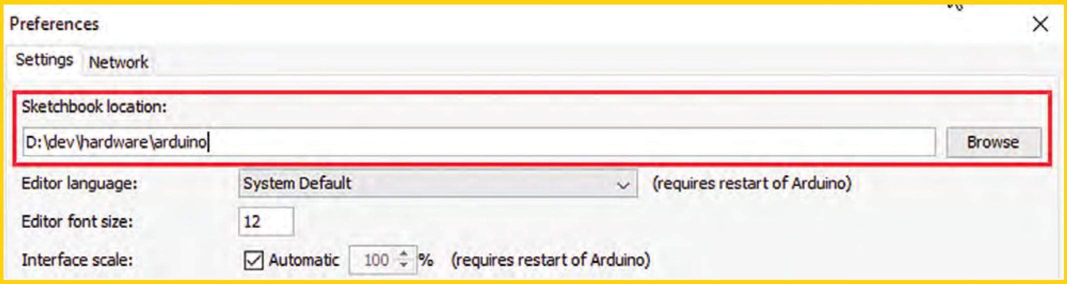
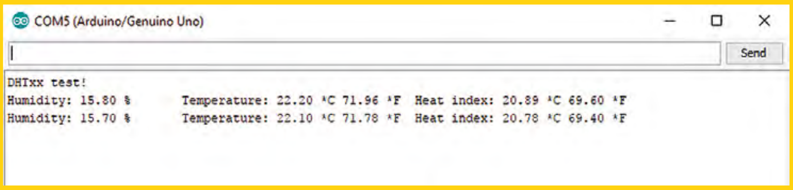

Preferences
Settings | Network
Sketchbook location:
D:\dev\hardware\arduino
Browse
Editor language: System Default (requires restart of Arduino)
Editor font size: 12
Interface scale: ☑ Automatic 100 % (requires restart of Arduino)

**Figure 3**
Arduino IDE Preferences



```
DHTxx test!
Humidity: 15.80 %       Temperature: 22.20 *C 71.96 *F  Heat index: 20.89 *C 69.60 *F
Humidity: 15.70 %       Temperature: 22.10 *C 71.78 *F  Heat index: 20.78 *C 69.40 *F
```

**Figure 4**
Serial Monitor output

When you buy LEDs in bulk from Amazon, they usually come with the appropriate resistor for the LED, otherwise you'll need to figure it out based on the LED voltage, your voltage source (5V for most Arduino devices). There are a lot of great articles online that walk you through the process of determining the best resistor; here's a good example: **hsmag.cc/fTXDmC**.

## USE A RESISTOR WITH YOUR LED CIRCUIT

Add an LED to the circuit, connecting the Arduino's digital output pin 3 to one leg of the resistor. Connect the other leg of the resistor to the LED's anode (the longer wire on the LED). Finally, connect the other LED connector (the negative, or cathode connector) to ground. **Figure 5** shows the updated circuit diagram, and final assembly is shown in **Figure 6**.

The code for this version of the project is based on the demo code from the libaray; We've removed the output to the serial monitor and added code that checks the current temperature against a threshold and sets the LED mode (on or off), appropriately.

At the beginning of the sketch, the code defines a new value called LEDPIN – this defines the digital output pin used to power the LED.

For this example, I've used pin 3, if your hardware has a different configuration, change the value for LEDPIN to match your hardware configuration.

Next, the sketch defines two constants: LESS_THAN and GREATER_THAN. These are numeric values that help the sketch understand how to determine if the temperature has exceeded a threshold. The threshold is defined in the TRIGGER_TEMP constant; set this constant to the temperature value you want to use as the temperature threshold. The TRIGGER_DIRECTION constant specifies whether the LED should turn on when the temperature is higher (greater than) or lower (less than) the trigger temperature. When TRIGGER_TEMP is 72, and TRIGGER_DIRECTION is GREATER_THAN, the LED will illuminate when the temperature is higher than 72 degrees.

```
#define LEDPIN 3 // the pin the LED is connected to

// constants used to determine the trigger direction
// greater or less than the trigger value
const int LESS_THAN = 0;
const int GREATER_THAN = 1;

// set this value to the temperature you want
// to trigger the LED on/off
const int TRIGGER_TEMP = 72; // Degrees F

// Trigger direction
const int TRIGGER_DIRECTION = GREATER_THAN;
//const int TRIGGER_DIRECTION = LESS_THAN;
```

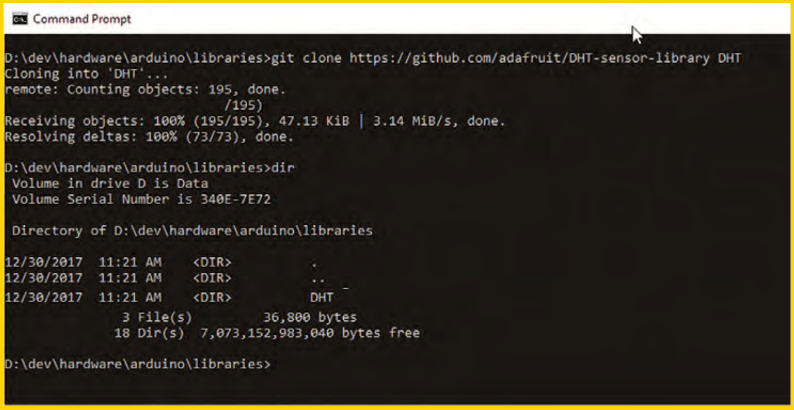
In the setup function, the sketch performs the same setup from the previous sketch; the only difference is that it validates that TRIGGER_DIRECTION is a valid value before continuing. The function also sets the pin mode for the LED to output and flashes the LED twice to let you know (visually) that the sketch is running.

```
void setup() {
 // initialize the serial communication link between
 // the Arduino device and the computer system
 // running the Serial Monitor
 Serial.begin(9600);
 Serial.println("DHT Temperature Monitor");

 Serial.println("Validating sketch configuration");
 // check to make sure trigger direction is
 // a valid value
 if ((TRIGGER_DIRECTION < LESS_THAN) ||
 (TRIGGER_DIRECTION > GREATER_THAN)) {
 Serial.println("Invalid value for TRIGGER_
DIRECTION, please fix the sketch and try again");
 // The code is broken, so loop infinitely
 while (true);
 // the sketch gets stuck here,
 // and never blinks the LED.
 }
 Serial.println("Configuration validated");

 // configure the LED pin for output mode
```

**Figure 5**
Cloning the Adafruit DHT Library



```
D:\dev\hardware\arduino\libraries>git clone https://github.com/adafruit/DHT-sensor-library DHT
Cloning into 'DHT'...
remote: Counting objects: 195, done.
                            /195)
Receiving objects: 100% (195/195), 47.13 KiB | 3.14 MiB/s, done.
Resolving deltas: 100% (73/73), done.

D:\dev\hardware\arduino\libraries>dir
 Volume in drive D is Data
 Volume Serial Number is 340E-7E72

 Directory of D:\dev\hardware\arduino\libraries

12/30/2017  11:21 AM    <DIR>          .
12/30/2017  11:21 AM    <DIR>          ..
12/30/2017  11:21 AM    <DIR>          DHT
               3 File(s)         36,800 bytes
              18 Dir(s)  7,073,152,983,040 bytes free

D:\dev\hardware\arduino\libraries>
```

```
 pinMode(LEDPIN, OUTPUT);

 // Initialize the dht object
 dht.begin();

 // blink the LED twice so you can tell the
 // sketch is working.
 blinkLED(2, 250);
}
```

Finally, in the loop function, the sketch reads the current temperature and figures out if the LED should be on or off. The sketch first checks to see if it's doing a less than or greater than comparison, then sets the LEDPIN output based on the results.

```
void loop() {
 // Wait two seconds between measurements
 delay(2000);

 // Read temperature in degrees Fahrenheit
 float currTemp = dht.readTemperature(true);
 // To work in degrees Celsius, do the following:
 //float currTemp = dht.readTemperature(false);

 // Make sure the sketch was able to read values
 if (isnan(currTemp)) {
 Serial.println("Failed to read from DHT sensor!");
 Return;
 }

 if (TRIGGER_DIRECTION < GREATER_THAN) {
 // Then we're doing a less than option,
 // so check to see if the current temp
 // is less than the trigger temp and write
 // the appropriate value to the output pin
 digitalWrite(LEDPIN, (currTemp < TRIGGER_TEMP) ?
HIGH : LOW);
 } else {
 // Otherwise, we're doing a greater than option,
 // so check to see if the current temp is greater
 // than the trigger temp and write the appropriate
 // value to the output pin
 digitalWrite(LEDPIN, (currTemp > TRIGGER_TEMP) ?
HIGH : LOW);
 }
}
```

To turn the LED on or off, the sketch simply writes either a HIGH or LOW value to the digital pin. In the past, you've seen it written like this:

```
digitalWrite(LEDPIN, HIGH);
```

In the sketch, we've dramatically reduced the amount of code needed by using what's known as a ternary expression, which is basically a three-part expression that looks like this:

```
currTemp < TRIGGER_TEMP ? HIGH : LOW
```

The first part is an expression that calculates to a true or false result, in this case, whether the current temperature is less than the trigger temperature. For a true result, the expression returns the value immediately following the question mark (in this case, HIGH). For a false result, the code returns the value after the colon (in this case, **LOW**). The call to **digitalWrite** writes a HIGH or LOW value to the output pin depending on the result of **currTemp < TRIGGER_TEMP**.

```
digitalWrite(LEDPIN, (currTemp < TRIGGER_TEMP) ?
HIGH : LOW);
```

Upload this code to the Arduino and your DHT22 temperature sensor circuit should start monitoring the local environment. □

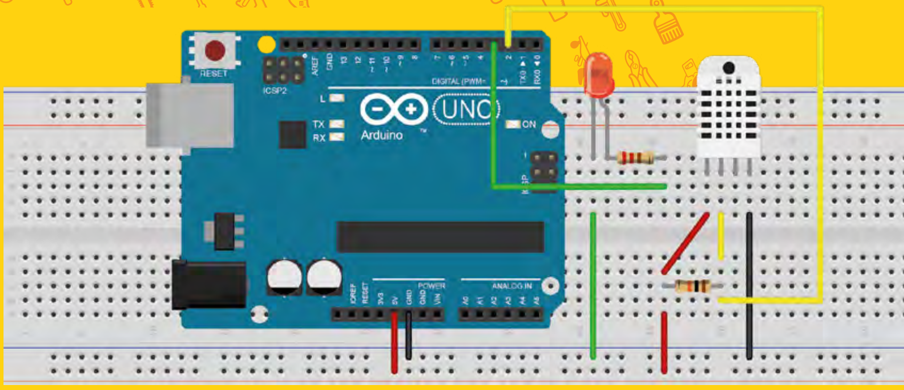


**Figure 5** Adding an LED to the circuit

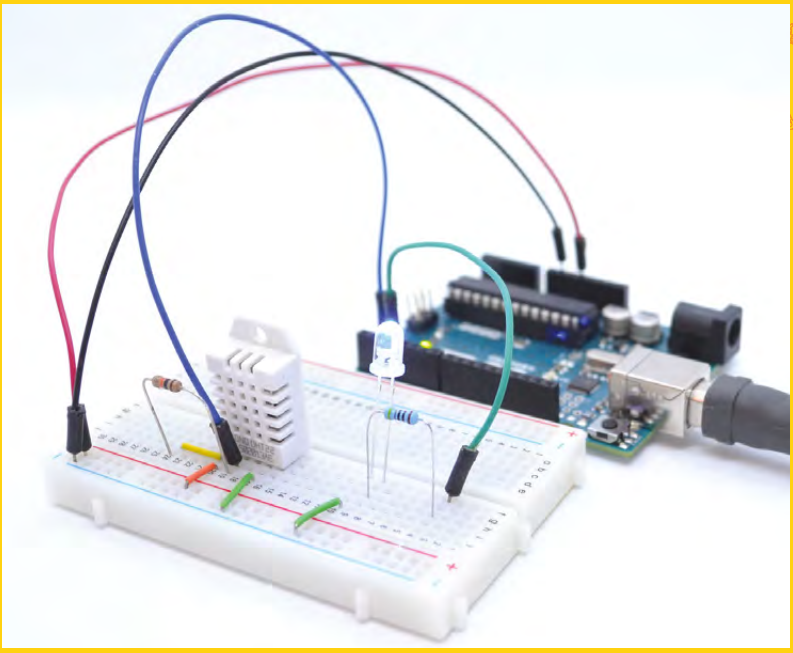


**Figure 6** Completed hardware with LED