Electric Imp Tails Project: Temperature Logger

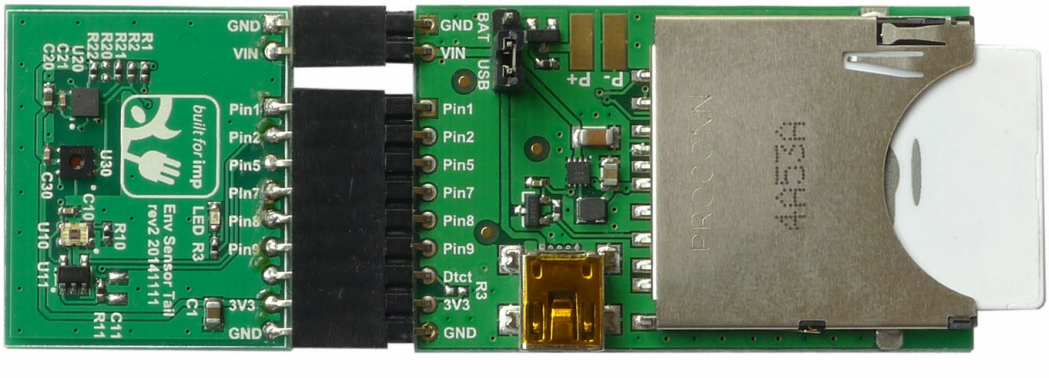
A fundamental feature of any Internet of Things device is the ability to make sense of the environment around it. The Env Tail includes sensors to read the key environmental qualities your own device might need to record, among them air temperature. So let's put the Tail's on-board thermometer to use.

We'll set the Tail up to take regular temperature readings – and to post them to the Internet. That lets you look back over your measurements, perhaps to find the patterns you can employ to make your home less wasteful of energy. That's beyond the scope of this project, but we will show you the basics of linking your app to the broader Internet.

The code provided will show you not only how to interact with sensors but also let you explore the basic bi-partite nature of Electric Imp IoT applications – device and agent – and how they communicate. It will also show you how you app's agent can reach out to third-party web services like Dweet.io. That data can, in turn, be presented with another web service, Freeboard.

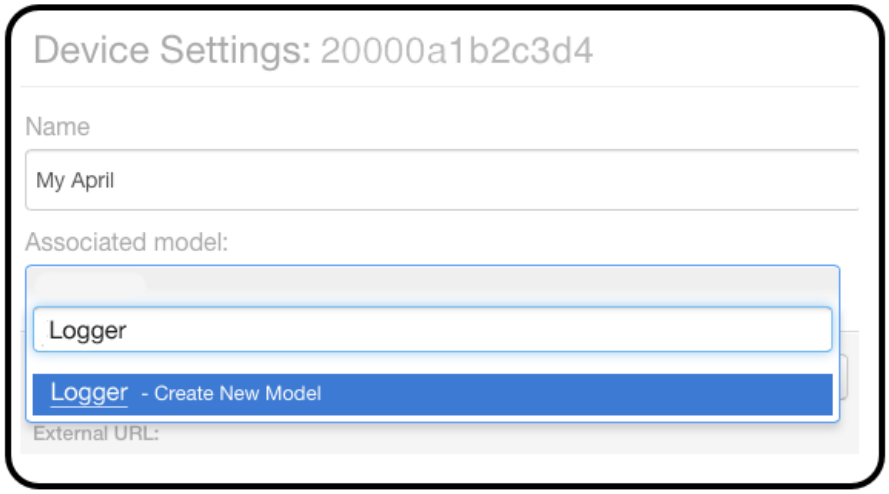
Step 1: Assemble the Hardware

If you haven't done so already, clip the Env Tail onto your April dev board. Slip in the imp001 card too, and connect the mini USB cable to a power supply and then to the April.



Step 2: Program the Project

Open the Electric Imp IDE in a web browser. You'll see your device listed on the left-hand side under 'Unassigned Devices'. Click on the gearwheel icon to the right of this to display the 'Device Settings' window. Here you can give the device a more friendly name, such as 'My April'. Click on the pop-up menu under 'Associated Model:' and in the empty space that appears, type in 'Logger' (without the single quotes). When you've done, click on 'Save Changes'.



Your device should now disappear from 'Unassigned Devices' and reappear under 'Logger' in the 'Active Models' section. If you can't see your device, just click on the disclosure triangle to the left of 'Logger' to reveal it. Can't see 'Logger'? Click on the disclosure triangle to the left of 'Active Models'.

Click on 'My April' and you'll see 'Agent', 'Device' and 'Device Logs' panels appear in the space on the right-side of the screen. This is where you enter your programs: one for the device, another for its online agent. Both blocks of code together comprise a model – Electric Imp terminology for an Internet of Things app.

The code you need is listed below; copy and paste it into the IDE's agent and device code panels. Make sure you paste it correctly. The agent code's first line should read:

```
    // Agent Code
```

The Agent Code

```
1  // Agent Code
2  #require "Dweetio.class.nut:1.0.0"
3
4  // Create a Dweet instance
5  local client = DweetIO();
6
7  // Add a function to post data from the device to your stream
8
9  function postReading(reading) {
10     // Note: reading is the data passed from the device, ie.
11     // a Squirrel table with the key 'temp'
12     client.dweet("templogger", reading, null);
13 }
14
15 // Register the function to handle data messages from the device
16 device.on("reading", postReading);
```

tempLogger.agent.nut hosted with ❤ by **GitHub** **view raw**

The Device Code

```
// Device Code
#require "Si702x.class.nut:1.0.0"

// Instance the Si702x and save a reference in tempHumidSensor
hardware.i2c89.configure(CLOCK_SPEED_400_KHZ);
local tempHumidSensor = Si702x(hardware.i2c89);

// Configure the LED (on pin 2) as digital out with 0 start state
local led = hardware.pin2;
led.configure(DIGITAL_OUT, 0);

// This function will be called regularly to take the temperature
// and log it to the device's agent

function takeTemp() {
    tempHumidSensor.read(function(reading) {
        // The read() method is passed a function which will be
        // called when the temperature data has been gathered.
        // This 'callback' function also needs to handle our
        // housekeeping: flash the LED to show a reading has
        // been taken; send the data to the agent;
        // put the device to sleep

        // Create a Squirrel table to hold the data - handy if we
        // later want to package up other data from other sensors
        local data = {};

        // Get the temperature using the Si7020 object's readTemp() method
        // Add the temperature using Squirrel's 'new key' operator
        data.temp <- reading.temperature;

        // Send the packaged data to the agent
        agent.send("reading", data);

        // Flash the LED to show we've taken a reading
        flashLed();

        // All done, we can put the device to sleep for 5 minutes,
        // but make sure impOS has no work of its own to do first –
        // ie. wait until it goes idle then call the packaged inline
        // function
        imp.onidle(function() {
            server.sleepfor(300);
        });
    });
}

function flashLed() {
    // Turn the LED on (write a HIGH value)
    led.write(1);

    // Pause for half a second
    imp.sleep(0.5);

    // Turn the LED off
    led.write(0);
}

// Take a temperature reading as soon as the device starts up
// Note: when the device wakes from sleep (caused by line 32)
// it runs its device code afresh - ie. it does a warm boot
takeTemp();
```

tempLogger.device.nut hosted with ❤ by **GitHub** **view raw**

Step 3: Customize the Code

To post your temperature data to Dweet.io, you just need to integrate the service's library into your code. Look for the line

```
    client.dweet("YOUR_TAIL_NAME", reading, null)
```

Delete the text between the double quote marks and add a name by which Dweet.io will label your data. It's a good idea to use your agent ID. This is the unique code that you'll see in the IDE at the head of the agent code pane making up part of the agent's URL. It'll look something like this:

```
https://agent.electricimp.com/AbCd0123eF
```

We've used `AbCd0123eF` in this example.

Step 4: Run the Code

The code you pasted into the IDE is now ready to run, so click on the 'Build and Run' button. You should see the Tail flash its LED and then do so every five minutes. Every time it does, it signals that the imp has taken a temperature reading from the Tail's Si7020 thermal sensor. The reading is packaged up and sent to the device's agent – its own personal gopher in the Cloud. The agent stays awake even when the device sleeps – which it does until it's time to take a fresh reading – and sends any data it receives to Dweet.io.

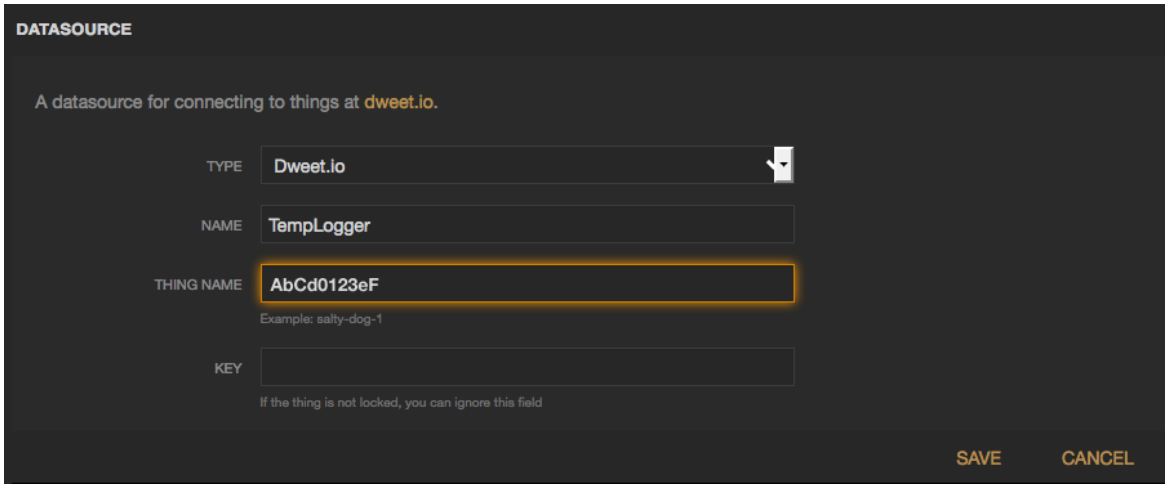
How does the agent know how to communicate with Dweet.io? Again, it makes use of an Electric Imp library, loaded at the start of the agent code. The code instantiates a new Dweet object. When data arrives from the device, the agent calls the Dweet object's *dweet()* method to relay the data. *dweet()* takes three parameters: the name of your Tail, which Dweet.io uses to identify your data; the data itself; and, optionally, a function that will be called when the Dweet.io server responds (we don't need this functionality here).

You can read about the Dweet.io library's full set of features here.

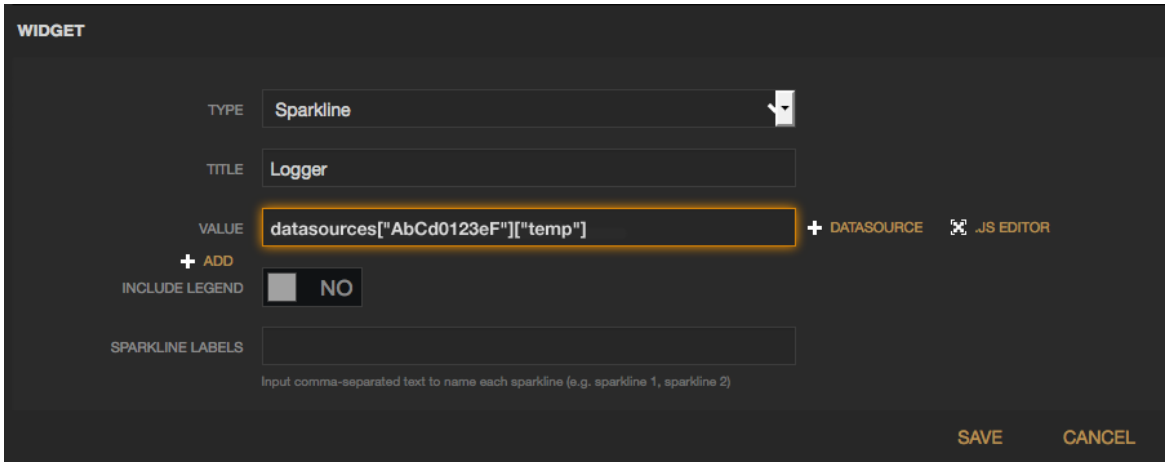
Step 5: Create a Data Display

To turn the data into an interesting display, you'll need sign up for a free Freeboard account here. When you've created your account, you'll be taken to a list of your Freeboards. You don't have one yet, so enter a name in the box provided and click on 'Create New'. We've used 'TempLogger' for our board; choose a different name for yours.

You'll be taken to the main Freeboard dashboard in which you add data sources and display widgets. First, set up the data source: click on the 'Add' button under 'Datasource'. In the dialog that appears, select 'Dweet.io' as your data source type, then enter the name you gave your Tail in Step 3, above. Click on 'Save'.

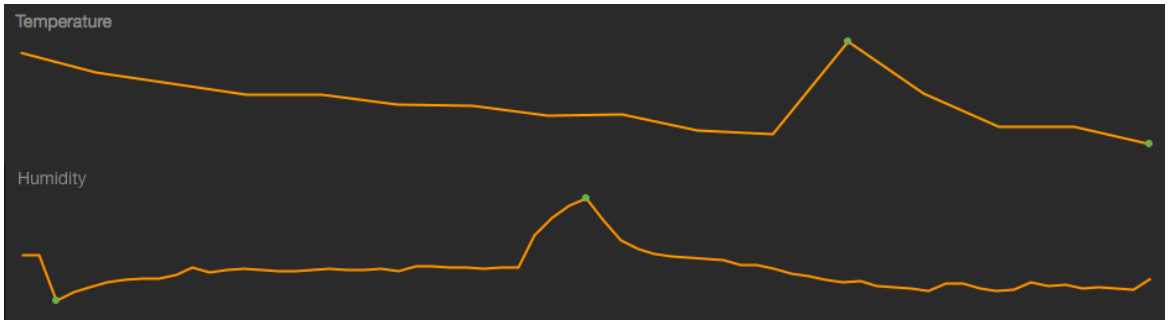


Now click on 'Add Pane'. A blank widget will appear in the lower half of the screen: click on its '+' icon to add a widget to the pane. Select 'Sparkline' as its type, and then click on the 'Add Datasource' button. Now pick the data source you just set up, clicking first on its name and then on the data available – you'll see 'temp', so select that. Click on 'Save' and Freeboard will begin charting the temperature data as it arrives.



Keep an eye on your Tail. When the blue LED flashes, signalling that a reading has been taken and sent to Dweet.io, you'll see the chart on

Freeboard update almost immediately afterward.



Step 6: What Next?

Well done – you've built a simple temperature logger, and got it taking temperature readings and posting them to a web service. That's pretty basic so here are some ideas to help you extend its functionality:

- Add humidity readings to your stream
  - You can get the humidity reading using the Si702x object's *readHumidity()* method.
  - Add a 'humidity' reading to the device code, and save the value in the data table that will be sent to the agent.
  - Edit your Freeboard widget. Click on the + icon next to the name you gave your pane, and add a new Sparkline widget that uses the humidity value now provided by your existing data source (you'll need to get your Tail posting data for this new value to show up here).
- Add other readings to your stream
  - The Env Tail page will show you how to address the Tail's other sensors in your code.
  - Add the LPS25H library to your code, create and configure a new instance of the class, and add code to take a reading and relay it in the data table to the agent.
  - Update your dashboard accordingly.
- Try some of the other Tails projects
  - Visit the Env Tail page for more applications you can explore.