Are you ok? Widget

Tags: Wireless, Sensors, Project, Internet of Things, Electric Imp

When Elizabeth and I got to talking about friends, neighbors, parents, and children we worry about, there was a theme: if we knew they were ok, it would be enough. The hard part is in not knowing if they came home last night or got up this morning. Elizabeth and I worked out what we wanted it to do on my [embedded software podcast]( <http://embedded.fm/episodes/2013/9/3/17-facebook-status-maybe-not-dead>).

We have the technology to alleviate such worries. The are-you-ok widget contains an [Electric Imp] (<https://www.sparkfun.com/products/11395>), an [accelerometer] (<https://www.sparkfun.com/products/10955>), and an [RGB LED] (<https://www.sparkfun.com/products/11120>).

If you put it in a stuffed animal, the person you care about can pat it every day. If they fail to do so, the Imp will tweet at you, send you email, and/or text your phone. Of course, you could put it on their refrigerator so you’d know they were ok *and* eating.

This is definitely an advanced tutorial. I’ll show you how I put together the hardware and the software but if you haven’t tried Electric Imp before, this may not be the place to start.

## Recommended Reading

* [Electric Imp Hook Up Guide] (<https://learn.sparkfun.com/tutorials/electric-imp-breakout-hookup-guide>)
* [How to Solder] (<https://learn.sparkfun.com/tutorials/how-to-solder---through-hole-soldering>)
* [Battery technologies] (<https://learn.sparkfun.com/tutorials/battery-technologies>)
* [I2C] (<https://learn.sparkfun.com/tutorials/i2c>)

# Tools and Materials

The items you will need for this are:

<script src="https://www.sparkfun.com/wish\_lists/69936.js"></script>

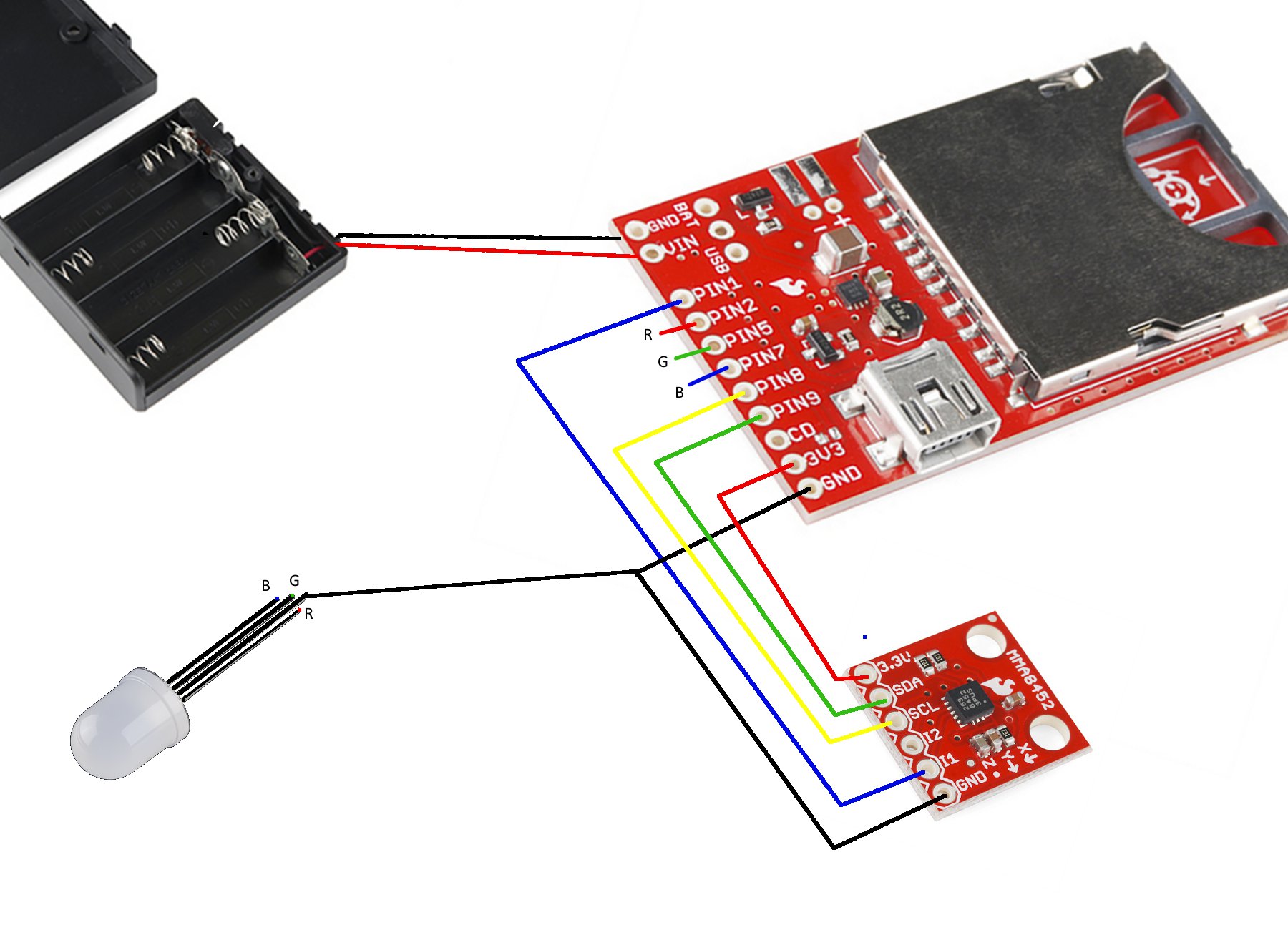
I chose right angle connectors in the kit because it goes better into a box or stuffed animal. Your form factor may vary. I also *really* like jumper wires over direct soldering but if you are more confident on the hardware side, I won’t stop you from direct soldering. Note, if you are building a plush, you may want to locate the LED and accelerometer away from the main board and batteries.

Note on the list, you will also need:

* 4AA batteries
* Soldering iron, solder
* Hot glue gun (ok, you don’t *need* one for this project but how can you live without it?)

# Building the hardware

[Connection plan.jpg]



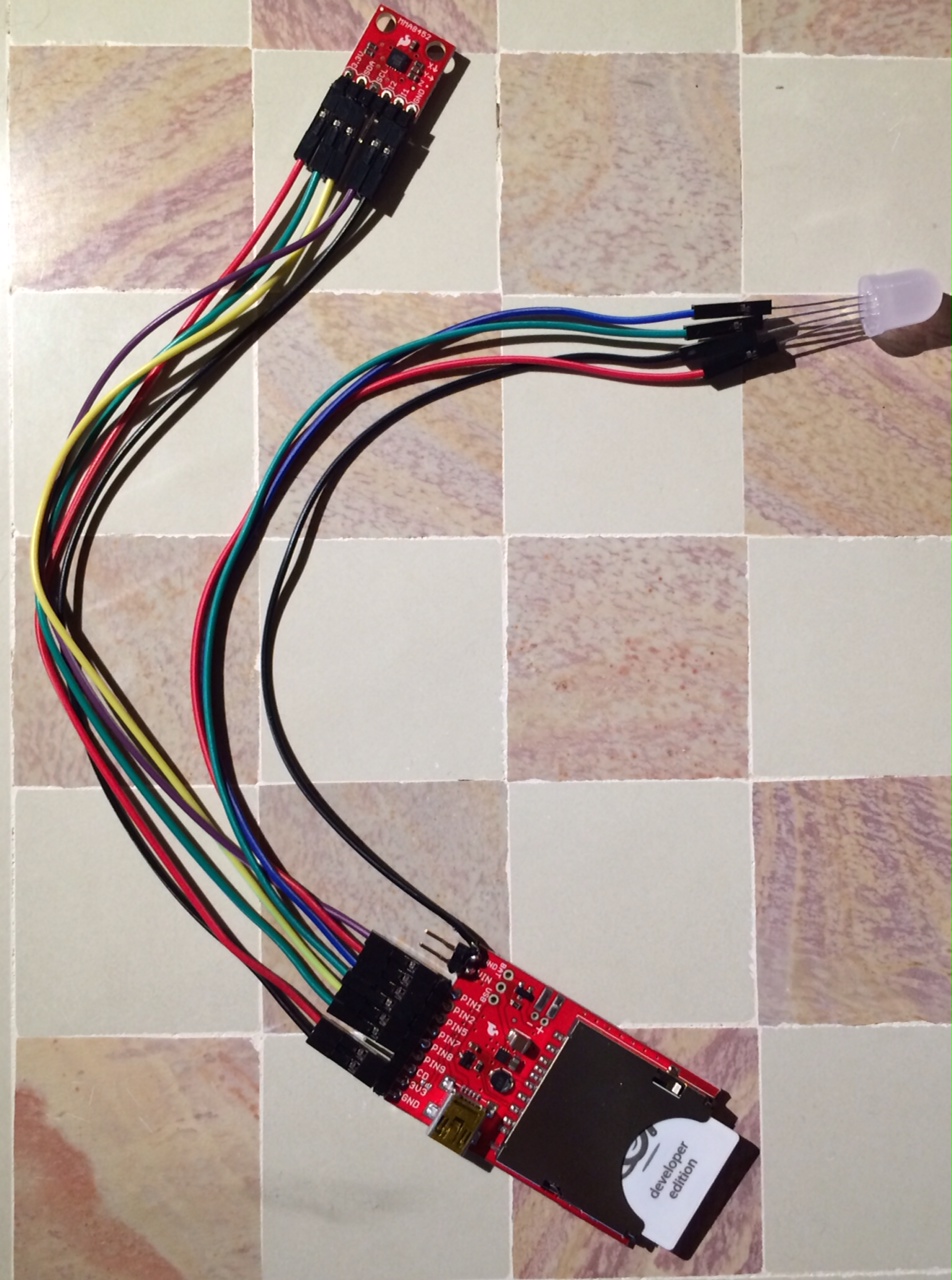
The build is fairly straight forward. We start with soldering:

1. Attach a two pin header to the Electric Imp breakout board for ground and VIN. Solder the battery wires on. (Then hot glue them or something to make sure they don’t break.)
2. Breakoff 9 pins on the header, solder those on to the Imp breakout board.
3. Since you’ll need two ground wires (one for the accel, one for then LED), cut the end off of one jumper wire, strip the coating, and solder it to the Imp breakout board. (We can wait to hot glue until the jumper wires are in place.)
4. Solder 6 pin header on to the accelerometer.

Now wire up the system:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name on**  **Imp breakout** | **Wire color** | **Goes to** | **Use** |
| PIN1 | Blue | Accel’s I1 | Interrupt to wake up the Imp. |
| PIN2 | Red | LED’s red leg | PWM red |
| PIN5 | Blue | LED’s blue leg | PWM blue |
| PIN7 | Green | LED’s green leg | PWM green |
| PIN8 | Yellow | Accel’s SCL | I2C clock signal |
| PIN9 | Green | Accel’s SDA | I2C data signal |
| 3V3 | Red | Accel’s 3.3V | Power |
| GND | Black | Accel’s GND  **AND**  Longest leg on LED | Ground |

After you are done, the result should look like this.

[are-you-ok wiring.jpg]

Note: the extra ground wire is soldered on and this is ready for a battery connection to be soldered on.

# What the software does (device)

If you haven’t used the Electric Imp, go see the [Electric Imp Hook Up Guide] (<https://learn.sparkfun.com/tutorials/electric-imp-breakout-hookup-guide>. You can use that code with this system set up as described. It will blink red if you change all instances of hardware.pin1 to hardware.pin2 in the device code.

However, once your have your Electric Imp talking to your network and downloading code, we can do a lot more. Our device is going to be a bit more complex as:

* Controls an RGB LED via PWM to do dimming
* Configures the accelerometer to interrupt when movement exceeds a threshold
* Goes to deep sleep to conserve battery
* On regular wakeups, reads the battery voltage

## Constants

At the top of the file are parameters for you to change.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* User parameters \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Imp's hardware.voltage reads the output of the regulator so we don't see the

// whole range of the batteries, do the best with the info available

const MAX\_EXPECTED\_VOLTAGE = 3.3; // max readable

const MIN\_EXPECTED\_VOLTAGE = 2.6; // dying

const MIN\_GOOD\_STATE\_OF\_CHARGE = 25; // percent

// when there is movement, how much movement does there have to be

// to get the accelerometer to wake up the device

const ACCEL\_TAP\_THRESHOLD = 10; // experimentally derived threshold

const ACCEL\_TRANSIENT\_THRESHOLD = 0x02; // experimentally derived threshold

// the LED ramps up to a color, holds for a bit, then ramps down

const LED\_HOLD\_TIME = 5.0; // seconds

const LED\_RAMP\_STEP\_TIME = 0.05; // seconds per ramp step (0.05 = 200Mhz)

const LED\_RAMP\_STEPS = 20; // steps in ramp at timing above

## Setup

First, set up the hardware to match the hookup instructions.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Hardware \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Pin Assignments according to silkscreen

\* Pin 1 = Input: wakeup interrupt from accelerometer

\* Pin 2 = PWM Red

\* Pin 5 = PWM Blue

\* Pin 7 = PWM Green

\* Pin 8 = I2C SCL (yellow wire for me)

\* Pin 9 = I2C SDA (green wire for me)

\*/

wakeupPin <- hardware.pin1;

redHWPin <- hardware.pin2;

greenHWPin <- hardware.pin5;

blueHWPin <- hardware.pin7;

i2c <- hardware.i2c89;

i2c.configure(CLOCK\_SPEED\_400\_KHZ);

## LED ramping

The ramp up and down, is done via the class LEDColor. Its functions are

* constructor(redPin, greenPin, bluePin) – initial creation, this is called with
  + local rgbLed = LEDColor(redHWPin, greenHWPin, blueHWPin);
* function setGoalColor (red, green, blue) – sets the state variable goalLED to the values passed in
* function update() – writes to the LED with the values in currentLED
* function off() – turns the LED off

This is used in conjunction with ledRamp function to make the LED come up softly, hold for a few seconds, then dim softly. The parameters can be reconfigured if you want a different profile.

## Battery monitoring

The battery monitoring subsystem is straightforward for AA batteries: read the Imp’s voltage and compare it against expected.

* FuelGaugeResetFromBoot – no initialization is needed for this monitoring
* FuelGaugeReadSoC – does a bit of math to move from reading to %, depends on the constants MIN\_EXPECTED\_VOLTAGE and MAX\_EXPECTED\_VOLTAGE set at the top of the file.

In the Going Further section, there is the option of using rechargeable LiPo batteries and monitoring them with a Fuel Gauge board. In that instance, these functions are replaced with more complex ones that speak to another chip via I2C.

## Accelerometer

Adding an MMA8452Q to an Electric Imp is pretty common so I used the [code available from duppypro] (https://gist.github.com/duppypro/7225636), making minor modification changes to the configuration (and some of the code).

The stock Sparkfun MMA8452Q has the address line pulled high so the I2C address is

const ACCEL\_ADDR = 0x3A // 0x1D << 1

If your accelerometer has the SAO line pulled down (the resistor in place on the back of on the Sparkfun board), change the address to

const ACCEL\_ADDR = 0×38 // 0x1C << 1

The accelerometer has a few functions you probably won’t need to call directly (they are internal to the subsystem):

* writeReg(addressToWrite, dataToWrite) – writes to accel’s address via I2C
* readSequentialRegs(addressToRead, numBytes) – reads accel’s address from I2C
* readReg(addressToRead) – calls the above function but for one byte at a time
* AccelerometerSetActive(mode) – sets the accelerometer into register modification mode or normal (active) mode.

The ones you may want to look at further are

* AccelerometerResetFromBoot() – verifies accel’s existence and configures interrupts and thresholds for the system
* AccelerometerIRQ() – called after the accelerometer

There in one unused function, there for debugging and amusement.

* readAccelData() – get the data from accelerometer

## Sidebar: Accelerometer fun

If you haven’t played with an accelerometer before, they can be fun. Change IndicateGoodInteraction() to something like this:

// Get values that are absolute value, 0 to 255, and proportional to 1G

local data = [0,0,0];

local rawData = readAccelData()

foreach (i, val in rawData) {

val = (val < 128 ? val : val - 256); // set val range -128 to 128

val = (val < 0 ? -val : val); // positive only

val = val \* 2; // brighter light

data [i] = (val > 256 ? 255 : val);

}

setGoalColor (data[0], data[1], data[2])

Now, the LED will indicate the orientation of the accelerometer every time you move or tap it.

## AYOK code

While the code far has been for different subsystems, this is the new code for making the are-you-ok widget do its thing.

* GetReadyToSleep() – sends the Imp to deepsleep. When it restarts, it will be from the top of the program (so it is different than the ledRamp’s imp.wakeup).
* CheckBatteryAndGoToSleep() – every hour the Imp wakes up to check battery status and send a note to the server
* IndicateGoodInteraction() – sets the LED to white and starts a ramp
* IndicateLowBattery() – sets LED to yellow, starts a ramp
* IndicateNoWiFi() – sets LED to red, starts a ramp
* HandleReasonForWakeup(unused = null) – the state machine of the device, it looks at the reason for wakeup and acts appropriately

When the Imp comes back from deepsleep, it runs this code:

imp.setpowersave(true);

Before we get very far into the code, we want the Imp to send this code an error if it has trouble connecting

server.setsendtimeoutpolicy(RETURN\_ON\_ERROR, WAIT\_TIL\_SENT, 30);

The important part there is the RETURN\_ON\_ERROR portion. . Without this line, the Imp will try to connect but won’t let the device code run. We’ll turn the LED to red if we can’t get connected.

Next, we need to make sure the accelerometer will wake the system up after it goes to sleep:

// Configure pin1 for wakeup. Connect MMA8452Q INT1 pin to imp pin1.

wakeupPin.configure(DIGITAL\_IN\_WAKEUP, AccelerometerIRQ);

Note that since the Imp is usually in deep sleep, the AccelerometerIRQ seldom gets called directly, instead HandleReasonForWakeup notes that wakeup reason is WAKEREASON\_PIN1.

If the system is not, then this code is responsible for calling HandleReasonForWakeup. It starts by verifying the system is connected to WiFi. If not, then it tries for 3s before calling HandleReasonForWakeup. (If the imp is connected, then it immediately calls HandleReasonForWakeup.)

if (!server.isconnected()) {

// we probably can't get to the internet, try for

// a little while (3 seconds), then get pushed to

// HandleReasonForWakeup where IndicateNoWiFi will be called

server.connect(HandleReasonForWakeup, 3)

} else {

HandleReasonForWakeup();

}

Much of the complexity here is due to the goal of having low power usage, ideally to make the system last for 6 months (or more) on four AAs.

# What the software does (agent)

The goal of the agent software is to send user messages if the device fails in some way. For debugging, it is often easier to get messages when something *does* happen. Thus, there are two output message methodologies.

## Settings

When debugging, you don’t necessarily want to flood your debug mechanism. If you keep tapping on the unit because you are fine tuning the color of the LED, you don’t want to get 1000 emails or twitter messages.

These next settings control how often you get messages and prevent message flurries.

// debug output frequency: these prevent twitter flurries where you

// get the same message 10 times because you are tapping the device

const dtDebugMessageMotionDetected = 80; // seconds

const dtDebugMessageBatteryUpdateDetected = 600; // seconds

The next setting is the most important.

// This is how long the device will go without an update from the

// user before it cries for help

const dtNoMotionDetected = 129600;

These should be set according to how often your device needs attention before it sends you email. If you set it to be too frequent, the person you are monitoring will find it intrusive. If you set it too long, you may not be able to alleviate your worry. Here are some settings I like:

|  |  |  |
| --- | --- | --- |
| User needs to move it | Hours | Setting |
| Three times a day | 12 | 43200 |
| Every day | 36 | 129600 |
| Every couple days | 60 | 216000 |

With the most frequent setting, remember the user needs to sleep so 12 hours represents a couple hours on either side of night’s rest. As for “every day”, it isn’t 24 hours because that would require the user to interact with the device every day at the same time (or a minute earlier). Using 36 hours instead means that one day can be 8am but the next can be 8pm. The “every couple days” has a similar buffer.

The next setting configures how long the device can fail to communicate before you get an error message. This is likely because the device has lost power or its WiFi is not available.

const dtNoBatteryUpdate = 21600; // seconds (21600 ==> 6 hours)

Peace of mind sometimes means knowing that everything is still working. I updated my system to send a note every few days to say everything is ok.

const dtEverythingFineUpdate = 432000; // seconds (432000 ==> 5 days)

Actually, everything may be working fine but you should know the device’s batteries are getting low. To that end, we have to decide when the “everything is fine” messages should include “but my battery is getting low”:

const MIN\_GOOD\_STATE\_OF\_CHARGE = 25; // percent

## Communicating with the caregiver

The different ways to communicate with the device require permissions. Electric Imp has example code for a number of web services: Xively, Twillio, Twitter, etc. I used the agent code from [their repository] (<https://github.com/electricimp/reference/tree/master/webservices/twitter>).

For twitter, you’ll need to use the [Twitter Developer Center] (<https://dev.twitter.com/apps/new>). Then you’ll need the keys to work:

\_CONSUMER\_KEY **<-** "YOUR KEY"

\_CONSUMER\_SECRET **<-** "YOUR SECRET"

\_ACCESS\_TOKEN **<-** "YOUR TOKEN"

\_ACCESS\_SECRET **<-** "YOUR SECRET"

These are at the top of my agent.nut but the code is a little further down (marked with a section break). It’s relevant function is twitter.Tweet(string).

Note: if you have trouble with twitter, modify the twitterDebug function to output messages to the Electric Imp server.log. Those messages get tedious to look at but are nice when starting out.

## Monitoring multiple devices

Once you have one of these devices, you may want another. Right after the Settings sections, there is an area to help with monitoring multiple devices:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Handle setting the device's name \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

If you only want one unit, and you want to skip the next steps, edit the code directly and fix the name of the unit and where to send caregiver messages.

settings <- {

name = "Maxwell", // name of the unit

attn = "@logicalelegance" // who to send messages to

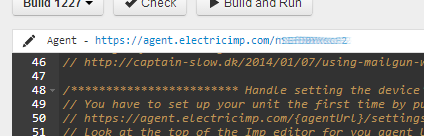
};

The other way to set these server stored variables, set up your unit the first time by putting the information in a URL of the form:

https://agent.electricimp.com/{agentUrl}/settings?name={nameValue}&attn={attnValue}

If you look at the top of the Imp editor for you agent URL, you'll see something like

[agentURL.jpg]



Copy the URL from the editor and add the necessary information so it looks like:

https://agent.electricimp.com/abcd1234/settings?name={Maxwell}&attn={@logicalelegance}

Again, Maxwell is the name of the unit and @logicalelegance is where I want messages to be sent.

Once the agent code is running, put that URL into a browser and it will write the information into server storage.

The next block of code handles the http interaction. After that is the twitter handling code.

## Messages to the user

As noted above, there are two types of output: debugMessage and userMessage. You may turn off debugMessage entirely once you are happy with how it is working.

For me, debug messages go to the server log and to my general account at twitter. User messages are called out in the server log (prefaced with !!!!) and go to the attn in twitter.

Since these functions are separated, it is straightforward to modify them to email, text, or use another media to inform you of events.

## Device handling

There are three asynchronous timers, each monitoring for things to happen (or not):

* Battery update – if this times out, we received no battery update: something is wrong since the device isn’t communicating
* Motion update – if this times out, the user has not moved the device: send a message to the caregiver to check on their loved one
* Everything is fine – if this times out, no errors occurred: send a reassurance message to the caregiver that all is working normally

These all have the same elements:

* modifiable timeout from the settings section on the top of the file (dtNoMotionDetected)
* timer variable used to start and stop the timer (motionUpdateFromDeviceTimer)
* function that is called when something happens that resets the timer, such as a motion is received from the device (motionOnDevice), this often causes a debug message
* callback when the timer expires (noMotionFromDevice), this causes a user message
* timer creation (motionUpdateFromDeviceTimer = imp.wakeup(dtNoMotionDetected, noMotionFromDevice);)

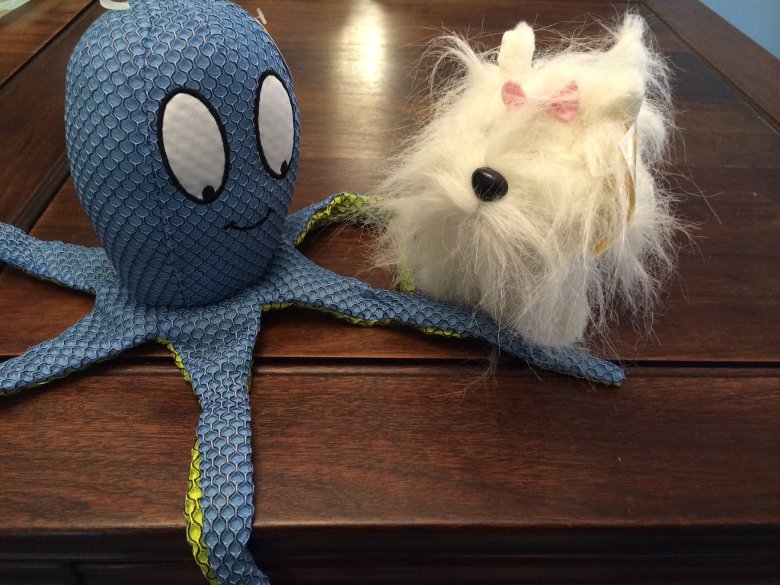
Once you see the symmetry between them, it is less of an impenetrable wall of code

# Making it look good

The technology is interesting, lots of pieces hooked together to make a whole that is greater than its parts. However, the social engineering on this device is critical to its real world use.

I like stuffed animals for the cuteness factor. But putting this in a box on a refrigerator requires less interaction from the person being monitored. In the end, I went for the stuffed animal, modifying an off the shelf plush octopus.

[MaxandDoggie.jpg]

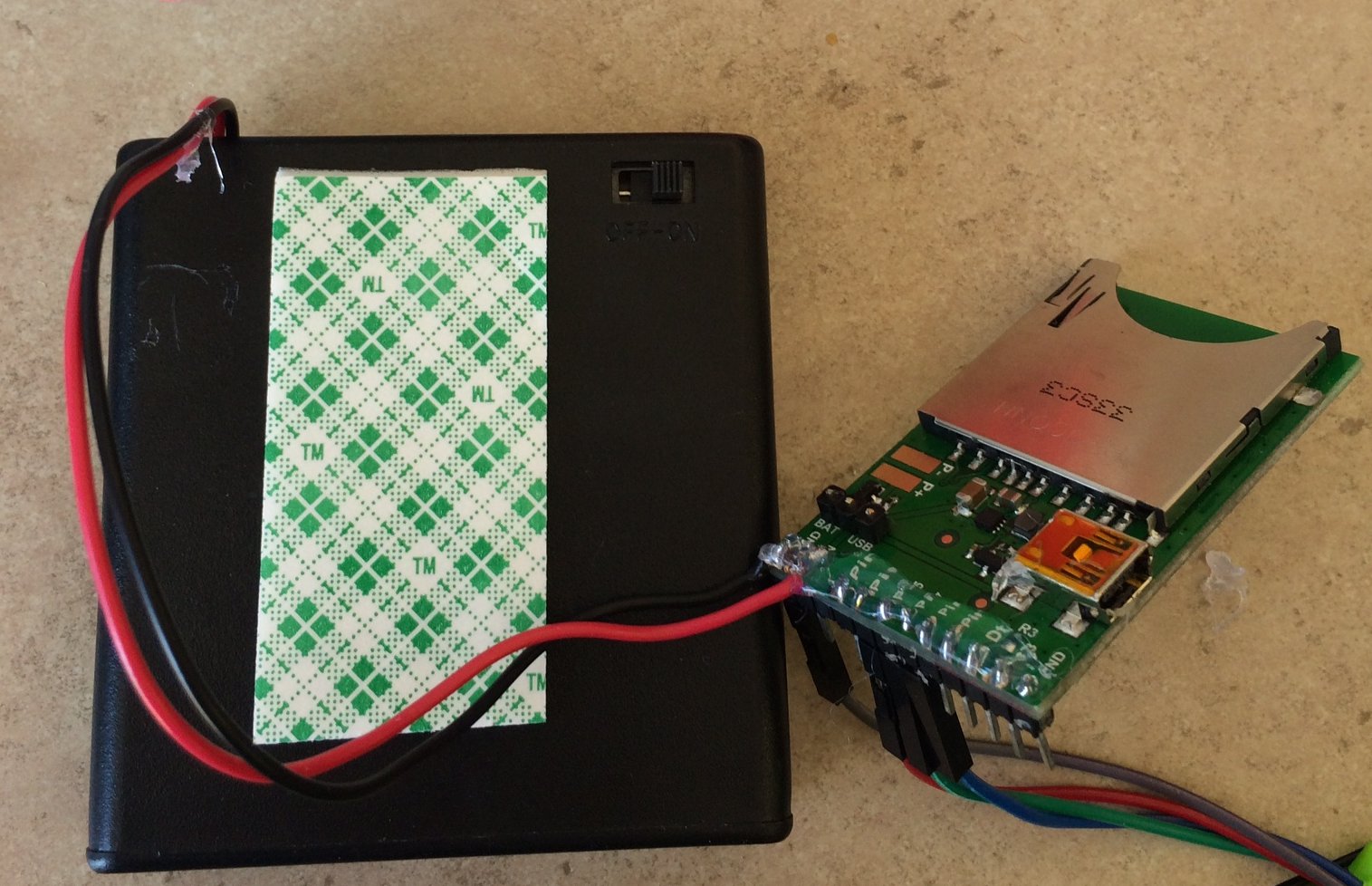


I clipped the LED leads then glued the jumper cables to the leads. I used fishing line to hang the LED in the octopus head.

[affixingTheLED.jpg]



A little more fishing line attached the accelerometer (which has mounting holes) to the very top of the head. The next step is to attach all of that to the Electric Imp to the battery compartment.

[doubleStickTapeAndImp.jpg]

Note that the Imp has to be accessible to the caregiver to set the WiFi information (via the Electric Imp Blink Up method with a smart phone). The batteries themselves should be changeable by the user (i.e. elderly neighbor). Organizing those two goals and getting all the cables to go in correct direction can be tricky. It is a matter of trying it before taping and hotgluing it together. Of course, I recommend testing it before putting it all together.

[glowingWithGuts2.jpg]



In the end, I’m pleased with how it turned out. But this is my first one (ahem, this is my first time atting electronics to a plush).

I have plans for the little dog stuffed animal. The first step is to choose a battery module that works for the end device:

* [Cube] (<https://www.sparkfun.com/products/550>)
* [Rectangle] (<https://www.sparkfun.com/products/551>)
* [Flat] (<https://www.sparkfun.com/products/12083>) (with cover switch!)

# Going further

The Electric Imp is so flexible that staying on topic can be tough. Once you’ve built one thing, it is easy to think of more feature.

## Make it rechargeable

An easy modification is to switch to [rechargeable batteries](<https://learn.sparkfun.com/tutorials/lipo-usb-charger-hookup-guide>).

A [2000mAh LiPo battery]( <https://www.sparkfun.com/products/8483>) will last almost a year, assuming you don’t update the code a lot.

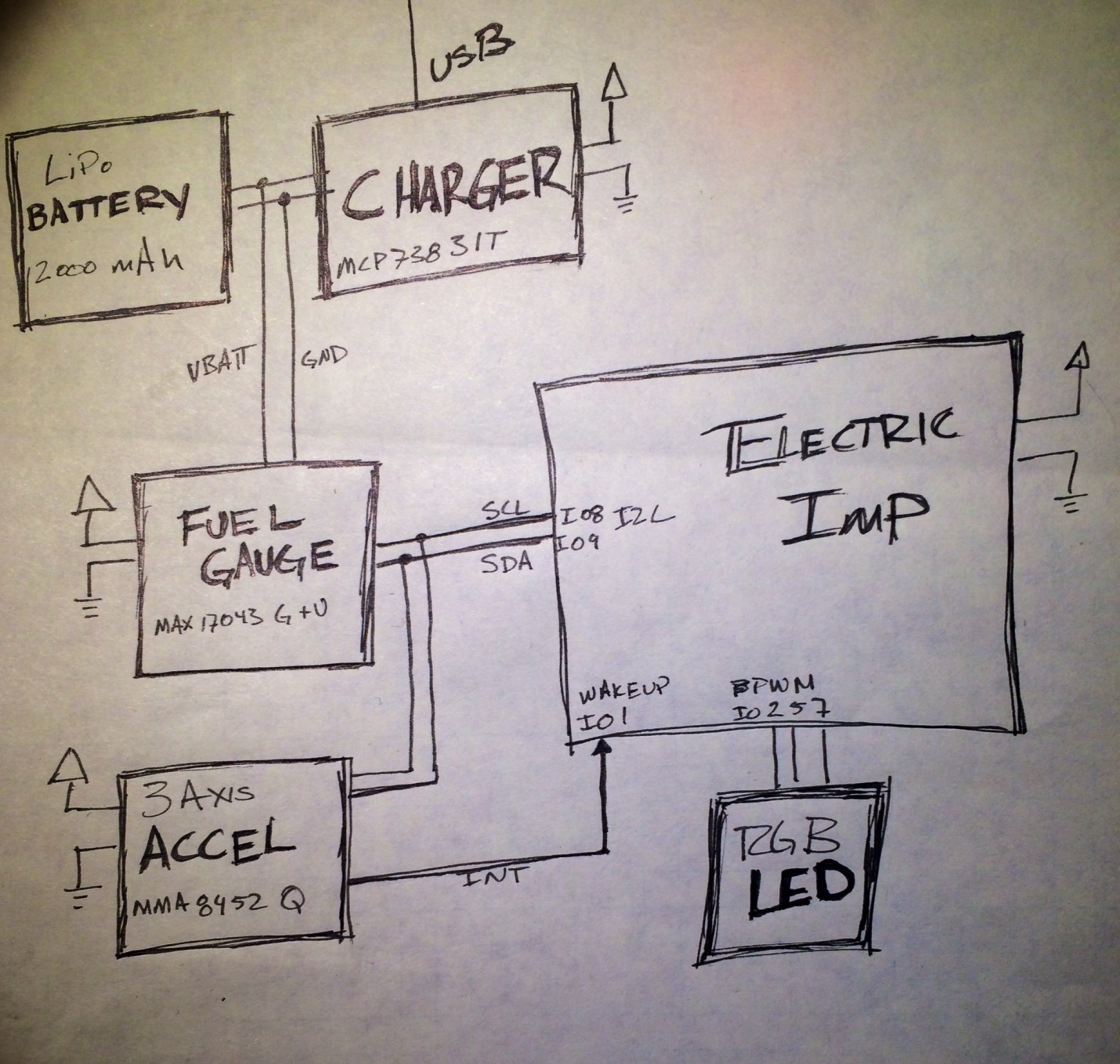
Get a LiPo charger with [mini USB](<https://www.sparkfun.com/products/10401>) or [micro USB]( <https://www.sparkfun.com/products/10217>).

Since LiPo batteries are tough to monitor based on voltage along, use a [fuel gauge](<https://www.sparkfun.com/products/10617>) board to do the monitoring.

I created [code](https://github.com/eleciawhite/ayok/blob/master/fw/rechargeable_device.nut) to make that work with the are-you-ok system, it drops in to replace the battery monitoring on the device.

The hard part is splitting the I2C wires so they go to both the accelerometer or and the fuel gauge. Here is the hook up sketch.

[prototype connection plan.jpg]



## Make it cheaper

If the price is too high, you can make the system cheaper by replace the accelerometer with a switch: either something the [user presses](<https://www.sparkfun.com/products/97>) or a [motion based switch] (<https://www.sparkfun.com/products/10289>). You will be able to remove the accelerometer code, only using the wakeup (pin 1). However, make sure you wire this so that the interrupt doesn’t happen (and the LED doesn’t show) until after the user has released the button.

Another cost cutting area is to replace the beautiful, diffuse LED with individual LEDs and control them directly. Of course, the diffuse one is only $0.95 so to make it cheaper, you may need to buy LEDs in bulk. I’ve ignored the intricacies of LED resistors but you might want to check out the [LED tutorial]( <https://learn.sparkfun.com/tutorials/light-emitting-diodes-leds/all>).

## Texting and Emailing

Right now the system uses twitter but that lacks privacy (and many people don’t use twitter). However, the Electric Imp already has [information]( <http://forums.electricimp.com/discussion/comment/4736>) and [code]( <https://github.com/electricimp/reference/tree/master/webservices/twilio>) available for texting via Twilio.

Email is also a good solution and there is [a great description]( <http://captain-slow.dk/2014/01/07/using-mailgun-with-electric-imp/>) on how to set that up.

## Configuration webpage

While creating a device specific http URL is great for setting the only two parameters, there are others we may want to set which means doing configuration via webpage

It is pretty silly to set up the frequency of interaction in the code. Also, it would be pretty easy to set the caregiver information and contact methodology in a webpage.

Here, I’ve made a sketch of what I want. Let me know when you are finished, all right?

[FutureWebsite.jpg]

