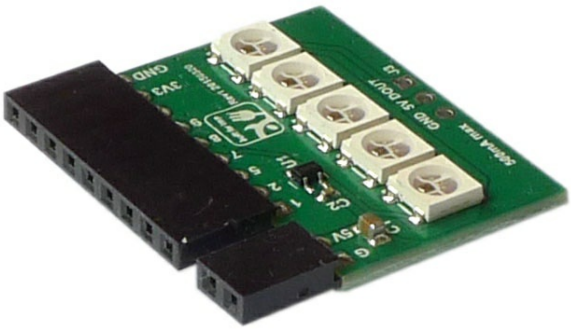
Electric Imp Tails Project: RGBWeather

This project uses the RGF LED Tail's five WS2812 LEDs as a display to show ambient 'weather effects'. The result is a handy gadget to have by your front door or by your desk, especially if you don't have a window. The code includes distinct animations are included for:

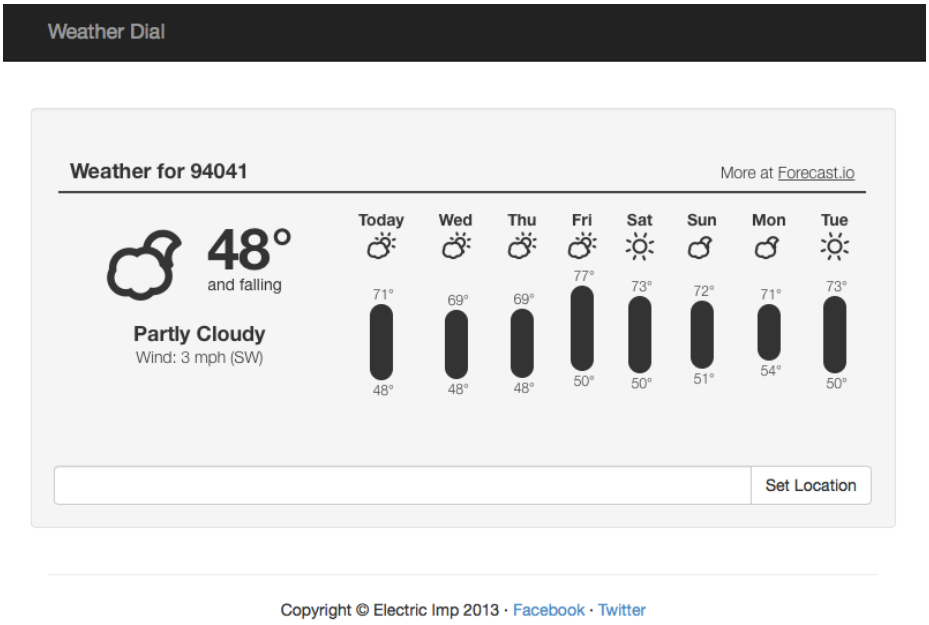
- Drizzle / Rain
- Thunderstorms
- Snow
- Mist
- Ice
- Fog / Haze
- Clear / Overcast conditions

When the weather is dry, the color of the display indicates the current temperature – its brightness is set by the cloud conditions. Colors range from dark blue at -10°C to warm yellow/green at around 15°C to bright red at 30°C.



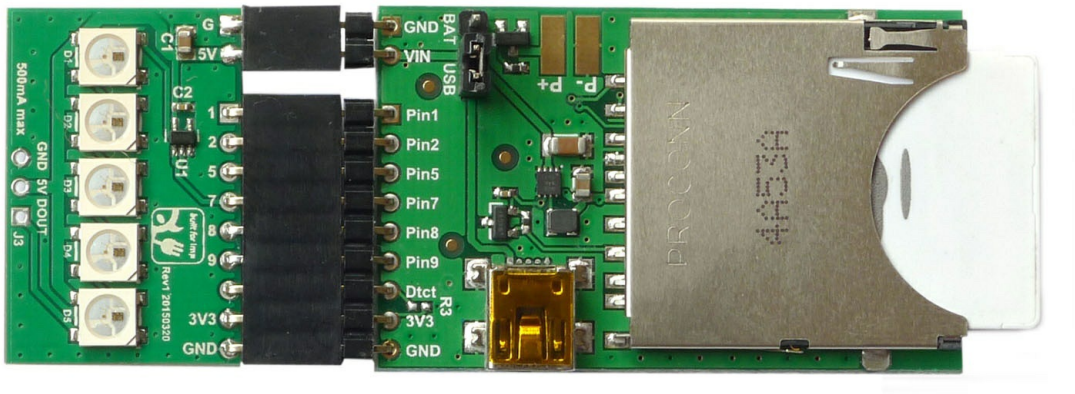
The weather data is obtained from Weather Underground, which has a free and feature-filled API.

The agent in this example serves a small web page to allow the user to change the forecast location and view the five-day forecast. The five-day forecast is sourced from forecast.io, another very useful service with free developer tools.



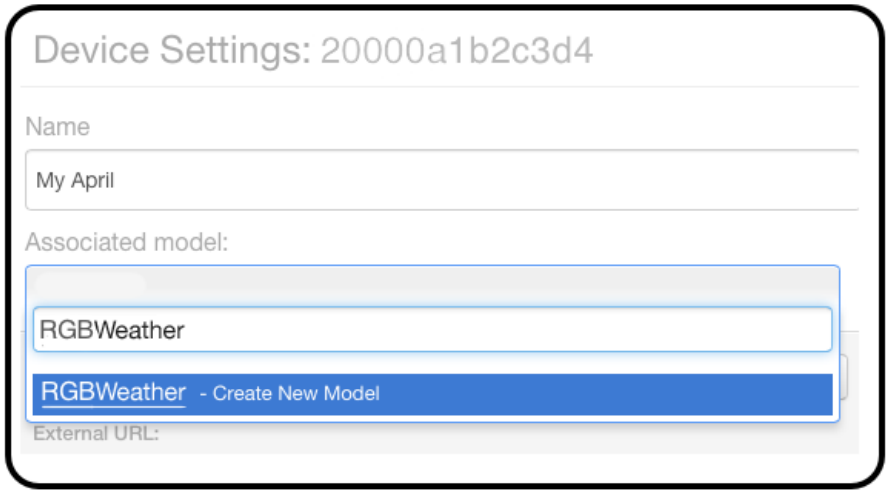
Step 1: Assemble the Hardware

If you haven't done so already, clip the RGB LED 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 'RGBWeather' (without the single quotes). When you've done, click on 'Save Changes'.



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

Agent Code

```
server.log("Weather Agent Running");
local AGENTRELOADED = true;

const UPDATEINTERVAL = 900; // fetch forecast every 10 minutes
updatehandle <- null;
WEBPAGE <- null;

// Add your own wunderground API Key here.
// Register for free at http://api.wunderground.com/weather/api/

const WUNDERGROUND_KEY = "YOUR WEATHERUNDERGROUND KEY";
local WUNDERGROUND_URL = "http://api.wunderground.com/api/";

local LOCATIONSTR = "94041";
```

```
savedata <- server.load();

if ("locationstr" in savedata) {
    LOCATIONSTR = savedata.locationstr;
    server.log("Restored Location String: " + LOCATIONSTR);
}

local LAT = null;
local LON = null;

// This function just assigns a big string to a global variable.
// The string happens to be a webpage, allowing the agent to serve
// a web UI to the user. The webpage must be stored as a verbatim
// multiline string, and therefore must not contain double quotes (")

function prepWebpage() {
    WEBPAGE = @"<!DOCTYPE html>
    <html lang='en'>
      <head>
        <meta charset='utf-8'>
        <meta name='viewport' content='width=device-width, initial-scale=1.0'>
        <meta name='description' content=''>
        <meta name='author' content=''>

        <title>Weather</title>
        <link href='
        <link href='https://netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap.min.css' rel='stylesheet'>
      </head>
      <body>

        <nav id='top' class='navbar navbar-static-top navbar-inverse' role='navigation'>
          <div class='container'>
            <div class='navbar-header'>
              <button type='button' class='navbar-toggle' data-toggle='collapse' data-target='.navbar-ex1-collapse'>
                <span class='sr-only'>Toggle navigation</span>
                <span class='icon-bar'></span>
                <span class='icon-bar'></span>
                <span class='icon-bar'></span>
              </button>
              <a class='navbar-brand'>Weather Dial</a>
            </div>

            <!-- Collect the nav links, forms, and other content for toggling -->
            <div class='collapse navbar-collapse navbar-ex1-collapse'>
              <ul class='nav navbar-nav'>
              </ul>
            </div><!-- /.navbar-collapse -->
          </div><!-- /.container -->
        </nav>

    <div class='container'>
            <div class='row' style='margin-top: 20px'>
                <div class='col-md-offset-2 col-md-8 well'>
                    <div class='row' style = 'margin-top: 20px; margin-bottom: 20px; margin-left:5px; margin-right: 5p
                        <iframe id='forecast_embed' type='text/html' frameborder='0' height='245' width='100%' src='ht
                    </div>
                    <div class='input-group'>
                        <input type='text' class='form-control' id='newLocation'>
                        <span class='input-group-btn'>
                            <button class='btn btn-default' type='button' onClick='setLocation()'>Set Location</button
                        </span>
                    </div>
                </div>
            </div>
        <hr>

        <footer>
            <div class='row'>
```

```
                <div class='col-lg-12'>
                    <p class='text-center'>Copyright &copy; Electric Imp 2013 &middot; <a href='http://facebook.com/el
                </div>
            </div>
        </footer>
    </div><!-- /.container -->

    <!-- javascript -->
    <script src='https://cdnjs.cloudflare.com/ajax/libs/jquery/2.0.3/jquery.min.js'></script>
    <script src='https://netdna.bootstrapcdn.com/bootstrap/3.0.2/js/bootstrap.min.js'></script>
    <script>

        var setLocationURL = document.URL+'/setLocation';
        var getLocationURL = document.URL+'/getLocation';
        var forecastBaseURL = 'https://forecast.io/embed/#';

        function setLocation() {
            var location = $('#newLocation').val();
            $.ajax({
                type: 'POST',
                url: setLocationURL,
                data: location,
                success: function(dataString) {
                    var data = $.parseJSON(dataString);
                    console.log(dataString);
                    var p = $('#forecast_embed').parent();
                    $('#forecast_embed').remove();
                    p.prepend('<iframe id=\'forecast_embed\' type=\'text/html\' frameborder=\'0\' height=\'245\' width=
                    $('#newLocation').val('');
                }
            });
        }

        $(document).ready(function() {
            $.ajax({
                type: 'GET',
                url: getLocationURL,
                success: function(dataString) {
                    console.log(dataString);
                    var data = $.parseJSON(dataString);
                    $('#forecast_embed').attr('src', forecastBaseURL+'&lat='+data.lat + '&lon='+data.lon+'&name='+data
                }
            });
        });
    </script>
    </body>
    </html>";
}

// Use weatherunderground to get the conditions, latitude and longitude given a location string.
// Location can be:
//   Country/City ("Australia/Sydney")
//   US State/City ("CA/Los_Altos")
//   Lat,Lon ("37.776289,-122.395234")
//   Zipcode ("94022")
//   Airport code ("SFO")

function getConditions() {
    // Prevent double-scheduled updates (in case both device and agent restart at some point)
    if (updatehandle) { imp.cancelwakeup(updatehandle); }

    // Schedule next update
    updatehandle = imp.wakeup(UPDATEINTERVAL, getConditions);

    // Use http.urlencode to URL-safe the human-readable location string,
    // then use string.split to remove "location=" from the result.
    local safelocationstr = split(http.urlencode({location = LOCATIONSTR}), "=")[1];
    local url = format("%s/%s/conditions/q/%s.json", WUNDERGROUND_URL, WUNDERGROUND_KEY, safelocationstr);
    local req = http.get(url, {}).sendsync();
```

```
    local res = http.get(url, {}).sendsync();
    if (res.statuscode != 200) {
        server.log("Wunderground error: " + res.statuscode + " => " + res.body);
    } else {
        try {
            local response = http.jsondecode(res.body);
            local weather = response.current_observation;
            LAT = weather.observation_location.latitude.tofloat();
            LON = weather.observation_location.longitude.tofloat();
            local forecastString = "";

            // Chunk together our forecast into a printable string
            forecastString += ("Forecast for " + weather.display_location.city + ", " + weather.display_location.state
            forecastString += (weather.weather + ", ");
            forecastString += ("Temperature " + weather.temp_f + "F, ");
            forecastString += (weather.temp_c + "C, ");
            forecastString += ("Humidity " + weather.relative_humidity + ", ");
            forecastString += ("Pressure " + weather.pressure_in + " in. ");

            if (weather.pressure_trend == "+") {
                forecastString += "and rising, ";
            } else if (weather.pressure_trend == "-") {
                forecastString += "and falling, ";
            } else {
                forecastString += "and steady, ";
            }

            forecastString += ("Wind " + weather.wind_mph + ". ");
            forecastString += weather.observation_time;
            device.send("seteffect", {conditions = weather.weather, temperature = weather.temp_c});
        } catch (err) {
            server.error("Wunderground error: " + err);
        }
    }
}

http.onrequest(function(request, response) {
    response.header("Access-Control-Allow-Origin", "*");
    local path = request.path.tolower();
    if (path == "/getlocation" || path == "/getlocation/") {
        if (LAT == null || LON == null) getConditions();
        response.send(200, http.jsonencode( { "lat":LAT, "lon":LON, "name":LOCATIONSTR } ));
    } else if (path == "/setlocation" || path == "/setlocation/") {
        LOCATIONSTR = request.body;
        getConditions();
        response.send(200, http.jsonencode( { "lat":LAT, "lon":LON, "name":LOCATIONSTR } ));

        // Keep the latest user-set location through agent restarts
        savedata.locationstr <- LOCATIONSTR;
        server.save(savedata);
    } else {
        // Serve the UI page
        response.send(200, WEBPAGE);
    }
});

// Handle device restarts while agent carries on running
device.on("start", function(dummyValue) {
    getConditions();
    AGENTRELOADED = false;
});

// Set up the web page
prepWebpage();

// Handle agent restarts while device carries on running
imp.wakeup(5, function() {
    if (AGENTRELOADED) { getConditions(); }
});
```

```
219 });
```

rgbWeather.agent.nut hosted with ❤ by GitHub view raw

Device Code

```
#require "WS2812.class.nut:1.0.0"

class RGBWeather extends WS2812 {
    // Control parameter for raindrop and thunder effects

    REFRESHPERIOD       = 0.05;  // normal effects refresh 20 times per second
    SLOWREFRESHPERIOD   = 0.2 ;  // slow effects refresh 5 times per second
    NEWPIXELFACTOR      = 1000;  // 1/100 pixels will show a new "drop" for a factor 1 effect
    LIGHTNINGFACTOR     = 5000;  // factor/5000 refreshes will yield lightning
    SCALE               = 100;   // NEWPIXELFACTOR / maximum "factor" value provided to an effect
                                 // this class uses factor 0-10 to set intensity
    MAXNEWDROP          = 500;    // max percent chance a new drop will occur on an empty pixel
    MAXLIGHTNING        = 10;    // max percentage chance lightning will occur on an frame
    LTBRTSCALE          = 3.1;   // amount to scale lightning brightness with intensity factor
    DIMPIXELPERCENT     = 0.8;   // percent of previous value to dim a pixel to when fading
    MAXBRIGHTNESS       = 24;   // maximum sum of channels to fade up to for ice, fog, and mist effects

    // Control parameters for temperature color effect
    TEMPFACTORDIV   = 4.0;
    TEMPRANGE       = 40;   // 40 degrees C of range
    TEMPMIN         = -10;
    TEMPRBOFFSET    = 10;   // red and green stay out of the middle by 10 degrees each to avoid white

    // Default color values
    RED         = [16,0,0];
    GREEN       = [0,16,0];
    BLUE        = [0,0,16];
    YELLOW      = [8,8,0];
    CYAN        = [0,8,8];
    MAGENTA     = [8,0,8];
    ORANGE      = [16,8,0];
    WHITE       = [8,8,8];

    // An array of [r,g,b] arrays to describe the next frame to be displayed
    pixelvalues = [];
    wakehandle = 0;  // keep track of the next imp.wakeup handle, so we can cancel if changing effects

    constructor(_spi, _frameSize) {
        base.constructor(_spi, _frameSize);
        pixelvalues = [];
        for (local x = 0 ; x < _frameSize ; x++) { pixelvalues.push([0,0,0]); }
    }

    // Stop all effects from displaying and blank out all the pixels.
    //  Input: (none)
    //  Return: (none)

    function stop() {
        // Cancel any previous effect currently running
        if (wakehandle) { imp.cancelwakeup(wakehandle); }
        dialvalues = array(_frameSize, [0,0,0]);
        clearFrame();
        writeFrame();
    }

    // Blue and Purple fading dots effect.
    // Factor is 1 to 10 and scales the number of new raindrops per refresh.

    function rain(factor) {
        local NUMCOLORS = 2;

        // Cancel any previous effect currently running
```

```
        if (wakehandle) { imp.cancelwakeup(wakehandle); }

        // Schedule refresh
        wakehandle = imp.wakeup((REFRESHPERIOD), function() {rain(factor)}.bindenv(this));
        local newdrop = 0;
        local threshold = (factor * SCALE);
        if (threshold < NUMCOLORS) threshold = NUMCOLORS;
        if (threshold > MAXNEWDROP) threshold = MAXNEWDROP;
        local next = false;
        clearFrame();
        for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
            // If there's any color data in this pixel, fade it down
            next = false;
            if (pixelvalues[pixel][0]) { pixelvalues[pixel][0] = math.floor(pixelvalues[pixel][0] * DIMPIXELPERCENT);
            if (pixelvalues[pixel][1]) { pixelvalues[pixel][1] = math.floor(pixelvalues[pixel][1] * DIMPIXELPERCENT);
            if (pixelvalues[pixel][2]) { pixelvalues[pixel][2] = math.floor(pixelvalues[pixel][2] * DIMPIXELPERCENT);

            // Skip random number generation if we just dimmed
            if (!next) {
                newdrop = math.rand() % NEWPIXELFACTOR;
                if (newdrop <= threshold) {
                    switch (newdrop % NUMCOLORS) {
                        case 0:
                            for (local channel = 0; channel < 3; channel++) {
                                pixelvalues[pixel][channel] = BLUE[channel];
                            }
                            break;

                        default:
                            for (local channel = 0; channel < 3; channel++) {
                                pixelvalues[pixel][channel] = MAGENTA[channel];
                            }
                    }
                }
            }

            writePixel(pixel, pixelvalues[pixel]);
        }

        writeFrame();
    }

    // White fading dots effect.
    // Factor is 1 to 10 and scales the number of new raindrops per refresh.

    function snow(factor) {
        local NUMCOLORS = 1;

        // Cancel any previous effect currently running
        if (wakehandle) { imp.cancelwakeup(wakehandle); }

        // Schedule refresh
        wakehandle = imp.wakeup((REFRESHPERIOD), function() {snow(factor)}.bindenv(this));
        local newdrop = 0;
        local threshold = (factor * SCALE);
        if (threshold < NUMCOLORS) threshold = NUMCOLORS;
        if (threshold > MAXNEWDROP) threshold = MAXNEWDROP;
        local next = false;
        clearFrame();

        for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
            // If there's any color data in this pixel, fade it down
            next = false;
            if (pixelvalues[pixel][0]) { pixelvalues[pixel][0] = math.floor(pixelvalues[pixel][0] * DIMPIXELPERCENT);
            if (pixelvalues[pixel][1]) { pixelvalues[pixel][1] = math.floor(pixelvalues[pixel][1] * DIMPIXELPERCENT);
            if (pixelvalues[pixel][2]) { pixelvalues[pixel][2] = math.floor(pixelvalues[pixel][2] * DIMPIXELPERCENT);

            // Skip random number generation if we just dimmed
```

```
            if (!next) {
                newdrop = math.rand() % NEWPIXELFACTOR;
                if (newdrop <= threshold) {
                    for (local channel = 0 ; channel < 3 ; channel++) {
                        pixelvalues[pixel][channel] = WHITE[channel];
                    }
                }
            }

            writePixel(pixel, pixelvalues[pixel]);
        }

        writeFrame();
    }

    // Blue and White fading dots effect.
    // Factor is 1 to 10 and scales the number of new raindrops per refresh.

    function hail(factor) {
        local NUMCOLORS = 3;

        // Cancel any previous effect currently running
        if (wakehandle) { imp.cancelwakeup(wakehandle); }

        // Schedule refresh
        wakehandle = imp.wakeup((REFRESHPERIOD), function() {hail(factor)}.bindenv(this));
        local newdrop = 0;
        local threshold = (factor * SCALE);
        if (threshold < NUMCOLORS) threshold = NUMCOLORS;
        if (threshold > MAXNEWDROP) threshold = MAXNEWDROP;
        local next = false;
        clearFrame();

        for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
            // If there's any color data in this pixel, fade it down
            next = false;
            if (pixelvalues[pixel][0]) { pixelvalues[pixel][0] = math.floor(pixelvalues[pixel][0] * DIMPIXELPERCENT);
            if (pixelvalues[pixel][1]) { pixelvalues[pixel][1] = math.floor(pixelvalues[pixel][1] * DIMPIXELPERCENT);
            if (pixelvalues[pixel][2]) { pixelvalues[pixel][2] = math.floor(pixelvalues[pixel][2] * DIMPIXELPERCENT);

            // Skip random number generation if we just dimmed
            if (!next) {
                newdrop = math.rand() % NEWPIXELFACTOR;
                if (newdrop <= threshold) {
                    switch (newdrop % NUMCOLORS) {
                        case 0:
                            for (local channel = 0 ; channel < 3 ; channel++) {
                                pixelvalues[pixel][channel] = CYAN[channel];
                            }
                            break;

                        case 1:
                            for (local channel = 0 ; channel < 3 ; channel++) {
                                pixelvalues[pixel][channel] = MAGENTA[channel];
                            }
                            break;

                        default:
                            for (local channel = 0 ; channel < 3 ; channel++) {
                                pixelvalues[pixel][channel] = WHITE[channel];
                            }
                    }
                }
            }

            writePixel(pixel, pixelvalues[pixel]);
        }

        writeFrame();
```

```
        writeFrame();
    }

    // Blue and Purple fading dots effect with yellow "lightning strikes".
    // Factor is 0 to 10 and scales the number of new raindrops per refresh,
    // as well as frequency of lightning.

    function thunder(factor) {
        local NUMCOLORS = 2;

        // Cancel any previous effect currently running
        if (wakehandle) { imp.cancelwakeup(wakehandle); }

        // Schedule refresh
        wakehandle = imp.wakeup((REFRESHPERIOD), function() {thunder(factor)}.bindenv(this));
        local newdrop = 0;
        local threshold = (factor * SCALE);
        if (threshold < NUMCOLORS) threshold = NUMCOLORS;
        if (threshold > MAXNEWDROP) threshold = MAXNEWDROP;

        local lightningthreshold = factor;
        if (lightningthreshold > MAXLIGHTNING) threshold = MAXLIGHTNING;

        local lightningcheck = math.rand() % LIGHTNINGFACTOR;
        local next = false;
        clearFrame();
        if (lightningcheck <= lightningthreshold) {
            local lightningbrightness = math.floor(factor * LTBRTSCALE);
            for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
                for (local channel = 0 ; channel < 3 ; channel++) {
                    pixelvalues[pixel][channel] = lightningbrightness * YELLOW[channel];
                }
            }
        } else {
            for (local pixel = 0; pixel < pixelvalues.len(); pixel++) {
                // If there's any color data in this pixel, fade it down
                next = false;
                if (pixelvalues[pixel][0]) { pixelvalues[pixel][0] = math.floor(pixelvalues[pixel][0] * DIMPIXELPERCEN
                if (pixelvalues[pixel][1]) { pixelvalues[pixel][1] = math.floor(pixelvalues[pixel][1] * DIMPIXELPERCEN
                if (pixelvalues[pixel][2]) { pixelvalues[pixel][2] = math.floor(pixelvalues[pixel][2] * DIMPIXELPERCEN

                // Skip random number generation if we just dimmed
                if (!next) {
                    newdrop = math.rand() % NEWPIXELFACTOR;
                    if (newdrop <= threshold) {
                        switch (newdrop % NUMCOLORS) {
                            case 0:
                                for (local channel = 0 ; channel < 3 ; channel++) {
                                    pixelvalues[pixel][channel] = BLUE[channel];
                                }
                                break;

                            default:
                                for (local channel = 0 ; channel < 3 ; channel++) {
                                    pixelvalues[pixel][channel] = MAGENTA[channel];
                                }
                        }
                    }
                }

                writePixel(pixel, pixelvalues[pixel]);
            }
        }

        writeFrame();
    }

    // Rotate pixelvalues array
    // this is used to animate the ice, mist, and fog effects
```

```
    // this is used to animate the ice, mist, and fog effects.

    function rotate_gradient() {
        // Schedule refresh
        wakehandle = imp.wakeup((REFRESHPERIOD), function() {rotate_gradient()}.bindenv(this));

        // Wrap around the end of the array
        for (local ch = 0 ; ch < 3 ; ch++) {
            pixelvalues[frameSize - 1][ch] = pixelvalues[0][ch];
        }

        writePixel(frameSize - 1, pixelvalues[frameSize - 1]);

        //Shift each pixel over by one
        for (local pixel = 0 ; pixel < (frameSize - 1) ; pixel++) {
            for (local ch = 0 ; ch < 3 ; ch++) {
                pixelvalues[pixel][ch] = pixelvalues[pixel + 1][ch];
            }

            writePixel(pixel, pixelvalues[pixel]);
        }

        writeFrame();
    }

    // Blue / white gradient circles the display
    // No input parameters

    function ice() {
        // Cancel any previous effect currently running
        if (wakehandle) { imp.cancelwakeup(wakehandle); }

        // Schedule refresh
        wakehandle = imp.wakeup((REFRESHPERIOD), function() {rotate_gradient()}.bindenv(this));
        local opposite = frameSize / 2;
        local step = (1.0 * WHITE[0]) / ((1.0 * opposite));
        for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
            if (pixel < opposite) {
                pixelvalues[pixel][0] = math.floor((opposite - pixel) * step);
                pixelvalues[pixel][1] = math.floor((opposite - pixel) * step);
                pixelvalues[pixel][2] = math.floor(((opposite - pixel) * step) + (pixel * step));
            } else {
                pixelvalues[pixel][0] = math.floor((pixel - opposite) * step);
                pixelvalues[pixel][1] = math.floor((pixel - opposite) * step);
                pixelvalues[pixel][2] = math.floor(((pixel - opposite) * step) + (frameSize - pixel) * step);
            }

            writePixel(pixel, pixelvalues[pixel]);
        }

        writeFrame();
    }

    // Cyan / white gradient circles the display
    // No input parameters

    function mist() {
        // Cancel any previous effect currently running
        if (wakehandle) { imp.cancelwakeup(wakehandle); }

        // Schedule refresh
        wakehandle = imp.wakeup((REFRESHPERIOD), function() {rotate_gradient()}.bindenv(this));
        local opposite = frameSize / 2;
        local step = (1.0 * WHITE[0]) / ((1.0 * opposite));
        for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
            if (pixel < opposite) {
                pixelvalues[pixel][0] = math.floor((opposite - pixel) * step);
                pixelvalues[pixel][1] = math.floor(((opposite - pixel) * step) + (pixel * step));
                pixelvalues[pixel][2] = math.floor(((opposite - pixel) * step) + (pixel * step));
```

```
            } else {
                pixelvalues[pixel][0] = math.floor((pixel - opposite) * step);
                pixelvalues[pixel][1] = math.floor(((pixel - opposite) * step) + (frameSize - pixel) * step);
                pixelvalues[pixel][2] = math.floor(((pixel - opposite) * step) + (frameSize - pixel) * step);
            }

            writePixel(pixel, pixelvalues[pixel]);
        }

        writeFrame();
    }

    // White gradient circles the display
    // No input parameters

    function fog() {
        local baseval = 4;

        // Cancel any previous effect currently running
        if (wakehandle) { imp.cancelwakeup(wakehandle); }

        // Schedule refresh
        wakehandle = imp.wakeup((REFRESHPERIOD), function() {rotate_gradient()}.bindenv(this));
        local opposite = frameSize / 2;
        local step = (2.0 * WHITE[0]) / ((1.0 * opposite));
        for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
            if (pixel < opposite) {
                pixelvalues[pixel][0] = math.floor((opposite - pixel) * step) + baseval;
                pixelvalues[pixel][1] = math.floor((opposite - pixel) * step) + baseval;
                pixelvalues[pixel][2] = math.floor((opposite - pixel) * step) + baseval;
            } else {
                pixelvalues[pixel][0] = math.floor((pixel - opposite) * step) + baseval;
                pixelvalues[pixel][1] = math.floor((pixel - opposite) * step) + baseval;
                pixelvalues[pixel][2] = math.floor((pixel - opposite) * step) + baseval;
            }

            writePixel(pixel, pixelvalues[pixel]);
        }

        writeFrame();
    }

    // Set the color based on the temperature, and the brightness based on the
    // factor. Temperature range is -10 to 30 C, where -10 is all-blue, and
    // 30 is all-red.

    function temp(val, factor) {
        // Cancel any previous effect currently running
        if (wakehandle) { imp.cancelwakeup(wakehandle); }
        factor = factor / TEMPFACTORDIV;

        // min temp is -10 C (full-on blue)
        // max temp is 30 C (full-on red)
        // scale temp up by 10 so we're dealing only with positive numbers
        if (TEMPMIN < 0) val += (-1 * TEMPMIN) ;
        if (val < 0) val = 0;
        if (val > TEMPRANGE) val = TEMPRANGE;

        // Scale red proportionally to temp, from -10 to 20 C
        local r_scale = (RED[0] * 1.0) / (TEMPRANGE - TEMPRBOFFSET);
        local r = (factor * r_scale * (val - TEMPRBOFFSET));
        if (val < TEMPRBOFFSET) r = 0;

        // Scale green proportionally to temp from 0 to 10, inversely to temp from 10 to 20
        local g_range = (TEMPRANGE - (2 * TEMPRBOFFSET)) / 2; // green shifts over a 10-degree range
        local g_max = (2 * TEMPRBOFFSET);                     // green max occurs at val = 20 (10 degrees)
        local g_scale = 2 * ((GREEN[1] * 1.0) / g_range);
        local g = 0;
```

```
        if ((val < TEMPRBOFFSET) || (val > (TEMPRANGE - TEMPRBOFFSET))) {
            g = 0;
        } else {
            g = factor * g_scale * (g_range - math.abs(g_max - val));
        }

        // Scale blue inverse to temp, from -10 to 20 C
        local b_scale = (BLUE[2] * 1.0) / (TEMPRANGE - TEMPRBOFFSET);
        local b = (factor * b_scale * ((TEMPRANGE - TEMPRBOFFSET) - val));
        if (val > (TEMPRANGE - TEMPRBOFFSET)) b = 0;
        for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
            pixelvalues[pixel][0] = r;
            pixelvalues[pixel][1] = g;
            pixelvalues[pixel][2] = b;
            writePixel(pixel, pixelvalues[pixel]);
        }

        writeFrame();
    }
}

// AGENT CALLBACKS

agent.on("seteffect", function(val) {
    local cond = null;
    local temp = null;

    try {
        cond = val.conditions;
        temp = val.temperature;
    } catch (err) {
        server.log("Invalid Request from Agent: " + err);
        return;
    }

    if (cond.find("Thunderstorm") != null) {
        if (cond.find("Light") != null) {
            display.thunder(0);
        } else if (cond.find("Heavy") != null) {
            display.thunder(4);
        } else {
            display.thunder(2);
        }
    } else if (cond.find("Hail") != null) {
        if (cond.find("Light") != null) {
            display.hail(0);
        } else if (cond.find("Heavy") != null) {
            display.hail(4);
        } else {
            display.hail(2);
        }
    } else if (cond.find("Drizzle") != null) {
        if (cond.find("Light") != null) {
            display.rain(0);
        } else if (cond.find("Heavy") != null) {
            display.rain(2);
        } else {
            display.rain(1);
        }
    } else if (cond.find("Rain") != null) {
        if (cond.find("Light") != null) {
            display.rain(3);
        } else if (cond.find("Heavy") != null) {
            display.rain(5);
        } else {
            display.rain(4);
        }
    } else if (cond.find("Snow") != null) {
```

```
        if (cond.find("Light") != null) {
            display.snow(2);
        } else if (cond.find("Heavy") != null) {
            display.snow(8);
        } else {
            display.snow(4);
        }
    } else if (cond.find("Ice") != null) {
        display.ice();
    } else if ((cond.find("Fog") != null)|| (cond.find("Haze") != null) || (cond.find("Dust") != null) || (cond.find("
        display.fog();
    } else if ((cond.find("Mist") != null) || (cond.find("Spray") != null)) {
        display.mist();
    } else if (cond.find("Clear") != null) {
        display.temp(temp, 3);
    } else if (cond.find("Cloud") != null) {
        display.temp(temp, 2);
    } else if (cond.find("Overcast") != null) {
        display.temp(temp, 1);
    } else {
        display.temp(temp, 1);
    }
});

// RUNTIME BEGINS HERE

// The number of pixels in your chain
const NUMPIXELS = 5;

spi <- hardware.spi257;
spi.configure(MSB_FIRST, 7500);
display <- RGBWeather(spi, NUMPIXELS);

// Let the agent know we've just booted, which will trigger a weather update.
agent.send("start", 0);
```

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

Step 3: Customize the Code

The code you've just paste into the IDE isn't yet ready to run – you need to tweak it to add your own Weather Underground API key, which grants the agent permission to access this web service and retrieve the data that ultimately controls what color patterns are displayed on the Tail.

You can obtain your API key by signing on for a free account. You'll need to verify the email address you entered during sign-up, and then 'purchase' an API key. For developers, the key is free, but you still have to 'buy' it. Don't worry, no credit card details are required.

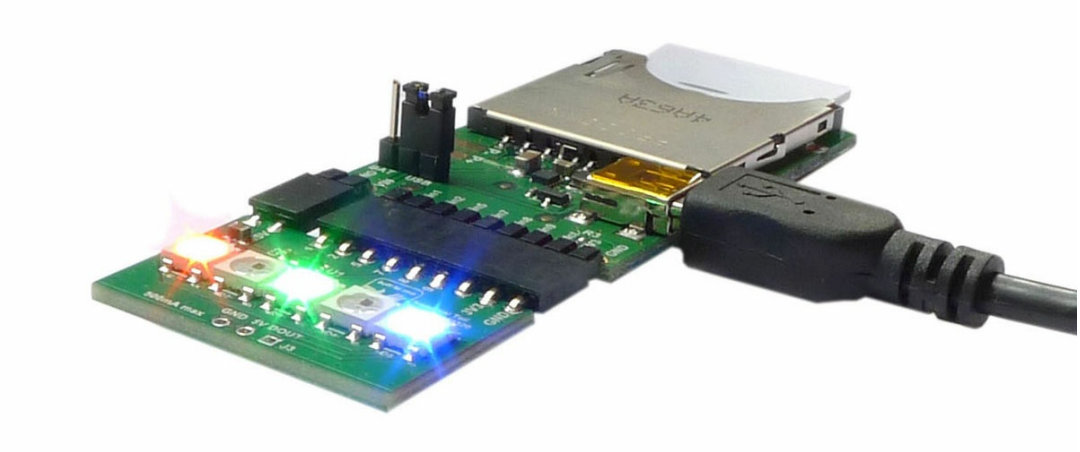
When you have your API key, look for this line in your agent code, up towards the top:

```
    const WUNDERGROUND_KEY = "YOUR WEATHERUNDERGROUND KEY"
```

Replace the text between the double quote marks with the API key you just obtained.

Step 4: Run the Code

Click on the 'Build and Run' button.



Step 5: Where Next?

There are a number of ways you can improve or customize the RGBWeather:

- Come up with your own weather display lighting effects
- Try some of the other Tails projects
    - Visit the RGB LED Tail page for more applications you can explore.

PLATFORM

BUSINESS SOLUTIONS

CUSTOMERS

DEV CENTER
Dev Kits
Getting Started
API Reference
Developer Guides
Hardware Reference
Manufacturing

ABOUT US
Jobs
FAQ
Media Coverage
Blog
Contact

electric imp

© 2015 Electric Imp, Inc.

Privacy Policy, Terms of Service