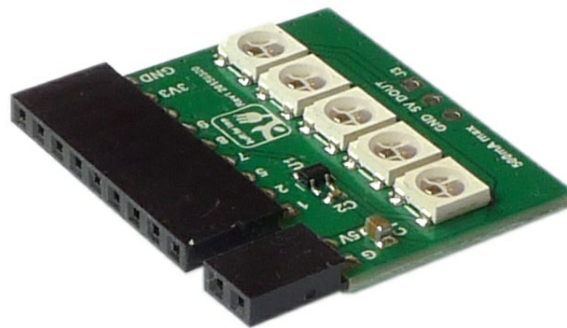


Electric Imp Tails Project: RGBWeather

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

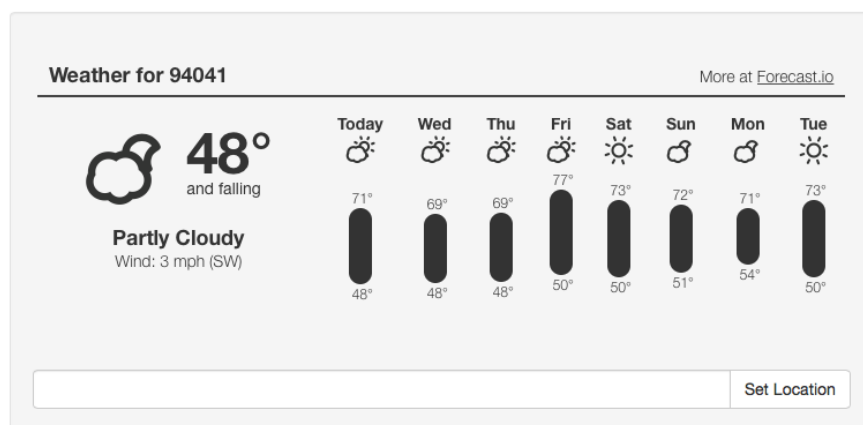
When the weather is dry, the color of the display indicates the current temperature – its brightness is set by the cloud conditions. Colors range from dark blue at -10°C to warm yellow/green at around 15°C to bright red at 30°C.



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.

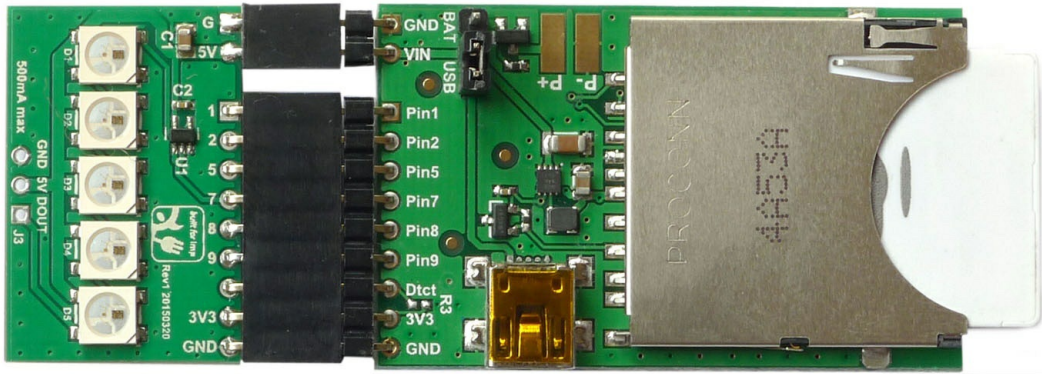
Weather Dial



Copyright © Electric Imp 2013 · [Facebook](#) · [Twitter](#)

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'.

Device Settings: 20000a1b2c3d4

Name

My April

Associated model:

RGBWeather

RGBWeather - Create New Model

External URL:

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

```

1 server.log("Weather Agent Running");
2 local AGENTRELOADED = true;
3
4 const UPDATEINTERVAL = 900; // fetch forecast every 10 minutes
5 updatehandle <- null;
6 WEBPAGE <- null;
7
8 // Add your own wunderground API Key here.
9 // Register for free at http://api.wunderground.com/weather/api/
10
11 const WUNDERGROUND_KEY = "YOUR WEATHERUNDERGROUND KEY";
12 local WUNDERGROUND_URL = "http://api.wunderground.com/api/";
13
14 local LOCATIONSTR = "94041";

```

```
15 savedata <- server.load();
16
17 if ("locationstr" in savedata) {
18     LOCATIONSTR = savedata.locationstr;
19     server.log("Restored Location String: " + LOCATIONSTR);
20 }
21
22 local LAT = null;
23 local LON = null;
24
25 // This function just assigns a big string to a global variable.
26 // The string happens to be a webpage, allowing the agent to serve
27 // a web UI to the user. The webpage must be stored as a verbatim
28 // multiline string, and therefore must not contain double quotes (")
29
30 function prepWebpage() {
31     WEBPAGE = @"<!DOCTYPE html>
32     <html lang='en'>
33     <head>
34         <meta charset='utf-8'>
35         <meta name='viewport' content='width=device-width, initial-scale=1.0'>
36         <meta name='description' content=''>
37         <meta name='author' content=''>
38
39         <title>Weather</title>
40         <link href='data:image/x-icon;base64,AAABAAEBAQAQAAAAAAoQAAFgAAACgAAAAQAAAAIAAAAAEABAAAAAAAgAAAAAAAAAAAAAAE
41         <link href='https://netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap.min.css' rel='stylesheet'>
42     </head>
43     <body>
44
45     <nav id='top' class='navbar navbar-static-top navbar-inverse' role='navigation'>
46         <div class='container'>
47             <div class='navbar-header'>
48                 <button type='button' class='navbar-toggle' data-toggle='collapse' data-target='.navbar-ex1-collapse'>
49                     <span class='sr-only'>Toggle navigation</span>
50                     <span class='icon-bar'></span>
51                     <span class='icon-bar'></span>
52                     <span class='icon-bar'></span>
53                 </button>
54                 <a class='navbar-brand'>Weather Dial</a>
55             </div>
56
57             <!-- Collect the nav links, forms, and other content for toggling -->
58             <div class='collapse navbar-collapse navbar-ex1-collapse'>
59                 <ul class='nav navbar-nav'>
60                     </ul>
61             </div><!-- /.navbar-collapse -->
62         </div><!-- /.container -->
63     </nav>
64
65     <div class='container'>
66         <div class='row' style='margin-top: 20px'>
67             <div class='col-md-offset-2 col-md-8 well'>
68                 <div class='row' style = 'margin-top: 20px; margin-bottom: 20px; margin-left:5px; margin-right: 5p
69                     <iframe id='forecast_embed' type='text/html' frameborder='0' height='245' width='100%' src='ht
70                 </div>
71                 <div class='input-group'>
72                     <input type='text' class='form-control' id='newLocation'>
73                     <span class='input-group-btn'>
74                         <button class='btn btn-default' type='button' onClick='setLocation()'>Set Location</button
75                     </span>
76                 </div>
77             </div>
78         </div>
79         <hr>
80
81         <footer>
82             <div class='row'>
```

```

83         <div class='col-lg-12'>
84             <p class='text-center'>Copyright &copy; Electric Imp 2013 &mdot; <a href='http://facebook.com/el
85         </div>
86     </div>
87 </footer>
88 </div><!-- /.container -->
89
90 <!-- javascript -->
91 <script src='https://cdnjs.cloudflare.com/ajax/libs/jquery/2.0.3/jquery.min.js'></script>
92 <script src='https://netdna.bootstrapcdn.com/bootstrap/3.0.2/js/bootstrap.min.js'></script>
93 <script>
94
95     var setLocationURL = document.URL+'/setLocation';
96     var getLocationURL = document.URL+'/getLocation';
97     var forecastBaseURL = 'https://forecast.io/embed/#';
98
99     function setLocation() {
100         var location = $('#newLocation').val();
101         $.ajax({
102             type: 'POST',
103             url: setLocationURL,
104             data: location,
105             success: function(dataString) {
106                 var data = $.parseJSON(dataString);
107                 console.log(dataString);
108                 var p = $('#forecast_embed').parent();
109                 $('#forecast_embed').remove();
110                 p.prepend('<iframe id=\'forecast_embed\' type=\'text/html\' frameborder=\'0\' height=\'245\' width=
111                 $('#newLocation').val('');
112             }
113         });
114     }
115
116     $(document).ready(function() {
117         $.ajax({
118             type: 'GET',
119             url: getLocationURL,
120             success: function(dataString) {
121                 console.log(dataString);
122                 var data = $.parseJSON(dataString);
123                 $('#forecast_embed').attr('src', forecastBaseURL+'&lat='+data.lat + '&lon='+data.lon+'&name='+data
124             }
125         });
126     });
127 </script>
128 </body>
129 </html>";
130 }
131
132 // Use weatherunderground to get the conditions, latitude and longitude given a location string.
133 // Location can be:
134 //   Country/City ("Australia/Sydney")
135 //   US State/City ("CA/Los_Altos")
136 //   Lat,Lon ("37.776289,-122.395234")
137 //   Zipcode ("94022")
138 //   Airport code ("SFO")
139
140 function getConditions() {
141     // Prevent double-scheduled updates (in case both device and agent restart at some point)
142     if (updatehandle) { imp.cancelwakeup(updatehandle); }
143
144     // Schedule next update
145     updatehandle = imp.wakeup(UPDATEINTERVAL, getConditions);
146
147     // Use http.urlencode to URL-safe the human-readable location string,
148     // then use string.split to remove "location=" from the result.
149     local safelocationstr = split(http.urlencode({location = LOCATIONSTR}), "=")[1];
150     local url = format("%s/%s/conditions/q/%s.json", WUNDERGROUND_URL, WUNDERGROUND_KEY, safelocationstr);
151     local resp = http.get(url, { headers: {

```

```

151     local res = http.get(url, {}).sendSync();
152     if (res.statuscode != 200) {
153         server.log("Wunderground error: " + res.statuscode + " => " + res.body);
154     } else {
155         try {
156             local response = http.jsondecode(res.body);
157             local weather = response.current_observation;
158             LAT = weather.observation_location.latitude.toFloat();
159             LON = weather.observation_location.longitude.toFloat();
160             local forecastString = "";
161
162             // Chunk together our forecast into a printable string
163             forecastString += ("Forecast for " + weather.display_location.city + ", " + weather.display_location.state
164             forecastString += (weather.weather + ", ");
165             forecastString += ("Temperature " + weather.temp_f + "F, ");
166             forecastString += (weather.temp_c + "C, ");
167             forecastString += ("Humidity " + weather.relative_humidity + ", ");
168             forecastString += ("Pressure " + weather.pressure_in + " in. ");
169
170             if (weather.pressure_trend == "+") {
171                 forecastString += "and rising, ";
172             } else if (weather.pressure_trend == "-") {
173                 forecastString += "and falling, ";
174             } else {
175                 forecastString += "and steady, ";
176             }
177
178             forecastString += ("Wind " + weather.wind_mph + ". ");
179             forecastString += weather.observation_time;
180             device.send("seteffect", {conditions = weather.weather, temperature = weather.temp_c});
181         } catch (err) {
182             server.error("Wunderground error: " + err);
183         }
184     }
185 }
186
187 http.onrequest(function(request, response) {
188     response.header("Access-Control-Allow-Origin", "*");
189     local path = request.path.toLowerCase();
190     if (path == "/getlocation" || path == "/getlocation/") {
191         if (LAT == null || LON == null) getConditions();
192         response.send(200, http.jsonencode( { "lat":LAT, "lon":LON, "name":LOCATIONSTR } ));
193     } else if (path == "/setlocation" || path == "/setlocation/") {
194         LOCATIONSTR = request.body;
195         getConditions();
196         response.send(200, http.jsonencode( { "lat":LAT, "lon":LON, "name":LOCATIONSTR } ));
197
198         // Keep the latest user-set location through agent restarts
199         savedata.locationstr <- LOCATIONSTR;
200         server.save(savedata);
201     } else {
202         // Serve the UI page
203         response.send(200, WEBPAGE);
204     }
205 });
206
207 // Handle device restarts while agent carries on running
208 device.on("start", function(dummyValue) {
209     getConditions();
210     AGENTRELOADED = false;
211 });
212
213 // Set up the web page
214 prepWebpage();
215
216 // Handle agent restarts while device carries on running
217 imp.wakeup(5, function() {
218     if (AGENTRELOADED) { getConditions(); }
219     ...

```

Device Code

```
1  #require "WS2812.class.nut:1.0.0"
2
3  class RGBWeather extends WS2812 {
4      // Control parameter for raindrop and thunder effects
5
6      REFRESHPERIOD      = 0.05; // normal effects refresh 20 times per second
7      SLOWREFRESHPERIOD  = 0.2 ; // slow effects refresh 5 times per second
8      NEWPIXELFACTOR     = 1000; // 1/100 pixels will show a new "drop" for a factor 1 effect
9      LIGHTNINGFACTOR    = 5000; // factor/5000 refreshes will yield lightning
10     SCALE               = 100; // NEWPIXELFACTOR / maximum "factor" value provided to an effect
11                          // this class uses factor 0-10 to set intensity
12     MAXNEWDROP          = 500; // max percent chance a new drop will occur on an empty pixel
13     MAXLIGHTNING        = 10;  // max percentage chance lightning will occur on an frame
14     LTBRTSCALE          = 3.1; // amount to scale lightning brightness with intensity factor
15     DIMPIXELPERCENT     = 0.8; // percent of previous value to dim a pixel to when fading
16     MAXBRIGHTNESS      = 24;  // maximum sum of channels to fade up to for ice, fog, and mist effects
17
18     // Control parameters for temperature color effect
19     TEMPFACORDIV        = 4.0;
20     TEMPRANGE           = 40;  // 40 degrees C of range
21     TEMPMIN             = -10;
22     TEMPRBOFFSET        = 10;  // red and green stay out of the middle by 10 degrees each to avoid white
23
24     // Default color values
25     RED                 = [16,0,0];
26     GREEN               = [0,16,0];
27     BLUE                = [0,0,16];
28     YELLOW              = [8,8,0];
29     CYAN                = [0,8,8];
30     MAGENTA             = [8,0,8];
31     ORANGE              = [16,8,0];
32     WHITE               = [8,8,8];
33
34     // An array of [r,g,b] arrays to describe the next frame to be displayed
35     pixelvalues = [];
36     wakehandle = 0; // keep track of the next imp.wakeup handle, so we can cancel if changing effects
37
38     constructor(_spi, _frameSize) {
39         base.constructor(_spi, _frameSize);
40         pixelvalues = [];
41         for (local x = 0 ; x < _frameSize ; x++) { pixelvalues.push([0,0,0]); }
42     }
43
44     // Stop all effects from displaying and blank out all the pixels.
45     // Input: (none)
46     // Return: (none)
47
48     function stop() {
49         // Cancel any previous effect currently running
50         if (wakehandle) { imp.cancelwakeup(wakehandle); }
51         dialvalues = array(_frameSize, [0,0,0]);
52         clearFrame();
53         writeFrame();
54     }
55
56     // Blue and Purple fading dots effect.
57     // Factor is 1 to 10 and scales the number of new raindrops per refresh.
58
59     function rain(factor) {
60         local NUMCOLORS = 2;
61
62         // Cancel any previous effect currently running
```

```

63     if (wakehandle) { imp.cancelwakeup(wakehandle); }
64
65     // Schedule refresh
66     wakehandle = imp.wakeup((REFRESHPERIOD), function() {rain(factor)}.bindenv(this));
67     local newdrop = 0;
68     local threshold = (factor * SCALE);
69     if (threshold < NUMCOLORS) threshold = NUMCOLORS;
70     if (threshold > MAXNEWDROP) threshold = MAXNEWDROP;
71     local next = false;
72     clearFrame();
73     for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
74         // If there's any color data in this pixel, fade it down
75         next = false;
76         if (pixelvalues[pixel][0]) { pixelvalues[pixel][0] = math.floor(pixelvalues[pixel][0] * DIMPIXELPERCENT);
77         if (pixelvalues[pixel][1]) { pixelvalues[pixel][1] = math.floor(pixelvalues[pixel][1] * DIMPIXELPERCENT);
78         if (pixelvalues[pixel][2]) { pixelvalues[pixel][2] = math.floor(pixelvalues[pixel][2] * DIMPIXELPERCENT);
79
80         // Skip random number generation if we just dimmed
81         if (!next) {
82             newdrop = math.rand() % NEWPIXELFACTOR;
83             if (newdrop <= threshold) {
84                 switch (newdrop % NUMCOLORS) {
85                     case 0:
86                         for (local channel = 0; channel < 3; channel++) {
87                             pixelvalues[pixel][channel] = BLUE[channel];
88                         }
89                         break;
90
91                     default:
92                         for (local channel = 0; channel < 3; channel++) {
93                             pixelvalues[pixel][channel] = MAGENTA[channel];
94                         }
95                 }
96             }
97         }
98
99         writePixel(pixel, pixelvalues[pixel]);
100     }
101
102     writeFrame();
103 }
104
105 // White fading dots effect.
106 // Factor is 1 to 10 and scales the number of new raindrops per refresh.
107
108 function snow(factor) {
109     local NUMCOLORS = 1;
110
111     // Cancel any previous effect currently running
112     if (wakehandle) { imp.cancelwakeup(wakehandle); }
113
114     // Schedule refresh
115     wakehandle = imp.wakeup((REFRESHPERIOD), function() {snow(factor)}.bindenv(this));
116     local newdrop = 0;
117     local threshold = (factor * SCALE);
118     if (threshold < NUMCOLORS) threshold = NUMCOLORS;
119     if (threshold > MAXNEWDROP) threshold = MAXNEWDROP;
120     local next = false;
121     clearFrame();
122
123     for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
124         // If there's any color data in this pixel, fade it down
125         next = false;
126         if (pixelvalues[pixel][0]) { pixelvalues[pixel][0] = math.floor(pixelvalues[pixel][0] * DIMPIXELPERCENT);
127         if (pixelvalues[pixel][1]) { pixelvalues[pixel][1] = math.floor(pixelvalues[pixel][1] * DIMPIXELPERCENT);
128         if (pixelvalues[pixel][2]) { pixelvalues[pixel][2] = math.floor(pixelvalues[pixel][2] * DIMPIXELPERCENT);
129
130         // Skip random number generation if we just dimmed

```

```

131         if (!next) {
132             newdrop = math.rand() % NEWPIXELFACTOR;
133             if (newdrop <= threshold) {
134                 for (local channel = 0 ; channel < 3 ; channel++) {
135                     pixelvalues[pixel][channel] = WHITE[channel];
136                 }
137             }
138         }
139
140         writePixel(pixel, pixelvalues[pixel]);
141     }
142
143     writeFrame();
144 }
145
146 // Blue and White fading dots effect.
147 // Factor is 1 to 10 and scales the number of new raindrops per refresh.
148
149 function hail(factor) {
150     local NUMCOLORS = 3;
151
152     // Cancel any previous effect currently running
153     if (wakehandle) { imp.cancelwakeup(wakehandle); }
154
155     // Schedule refresh
156     wakehandle = imp.wakeup((REFRESHPERIOD), function() {hail(factor)}.bindenv(this));
157     local newdrop = 0;
158     local threshold = (factor * SCALE);
159     if (threshold < NUMCOLORS) threshold = NUMCOLORS;
160     if (threshold > MAXNEWDROP) threshold = MAXNEWDROP;
161     local next = false;
162     clearFrame();
163
164     for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
165         // If there's any color data in this pixel, fade it down
166         next = false;
167         if (pixelvalues[pixel][0]) { pixelvalues[pixel][0] = math.floor(pixelvalues[pixel][0] * DIMPIXELPERCENT);
168         if (pixelvalues[pixel][1]) { pixelvalues[pixel][1] = math.floor(pixelvalues[pixel][1] * DIMPIXELPERCENT);
169         if (pixelvalues[pixel][2]) { pixelvalues[pixel][2] = math.floor(pixelvalues[pixel][2] * DIMPIXELPERCENT);
170
171         // Skip random number generation if we just dimmed
172         if (!next) {
173             newdrop = math.rand() % NEWPIXELFACTOR;
174             if (newdrop <= threshold) {
175                 switch (newdrop % NUMCOLORS) {
176                     case 0:
177                         for (local channel = 0 ; channel < 3 ; channel++) {
178                             pixelvalues[pixel][channel] = CYAN[channel];
179                         }
180                         break;
181
182                     case 1:
183                         for (local channel = 0 ; channel < 3 ; channel++) {
184                             pixelvalues[pixel][channel] = MAGENTA[channel];
185                         }
186                         break;
187
188                     default:
189                         for (local channel = 0 ; channel < 3 ; channel++) {
190                             pixelvalues[pixel][channel] = WHITE[channel];
191                         }
192                 }
193             }
194         }
195
196         writePixel(pixel, pixelvalues[pixel]);
197     }
198
199     writeFrame();

```



```

199     writeFrame();
200 }
201
202 // Blue and Purple fading dots effect with yellow "lightning strikes".
203 // Factor is 0 to 10 and scales the number of new raindrops per refresh,
204 // as well as frequency of lightning.
205
206 function thunder(factor) {
207     local NUMCOLORS = 2;
208
209     // Cancel any previous effect currently running
210     if (wakehandle) { imp.cancelwakeup(wakehandle); }
211
212     // Schedule refresh
213     wakehandle = imp.wakeup((REFRESHPERIOD), function() {thunder(factor)}.bindenv(this));
214     local newdrop = 0;
215     local threshold = (factor * SCALE);
216     if (threshold < NUMCOLORS) threshold = NUMCOLORS;
217     if (threshold > MAXNEWDROP) threshold = MAXNEWDROP;
218
219     local lightningthreshold = factor;
220     if (lightningthreshold > MAXLIGHTNING) threshold = MAXLIGHTNING;
221
222     local lightningcheck = math.rand() % LIGHTNINGFACTOR;
223     local next = false;
224     clearFrame();
225     if (lightningcheck <= lightningthreshold) {
226         local lightningbrightness = math.floor(factor * LTBRTSCALE);
227         for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
228             for (local channel = 0 ; channel < 3 ; channel++) {
229                 pixelvalues[pixel][channel] = lightningbrightness * YELLOW[channel];
230             }
231         }
232     } else {
233         for (local pixel = 0; pixel < pixelvalues.len(); pixel++) {
234             // If there's any color data in this pixel, fade it down
235             next = false;
236             if (pixelvalues[pixel][0]) { pixelvalues[pixel][0] = math.floor(pixelvalues[pixel][0] * DIMPIXELPERCENT); }
237             if (pixelvalues[pixel][1]) { pixelvalues[pixel][1] = math.floor(pixelvalues[pixel][1] * DIMPIXELPERCENT); }
238             if (pixelvalues[pixel][2]) { pixelvalues[pixel][2] = math.floor(pixelvalues[pixel][2] * DIMPIXELPERCENT); }
239
240             // Skip random number generation if we just dimmed
241             if (!next) {
242                 newdrop = math.rand() % NEWPIXELFACTOR;
243                 if (newdrop <= threshold) {
244                     switch (newdrop % NUMCOLORS) {
245                         case 0:
246                             for (local channel = 0 ; channel < 3 ; channel++) {
247                                 pixelvalues[pixel][channel] = BLUE[channel];
248                             }
249                             break;
250
251                         default:
252                             for (local channel = 0 ; channel < 3 ; channel++) {
253                                 pixelvalues[pixel][channel] = MAGENTA[channel];
254                             }
255                         }
256                     }
257                 }
258
259                 writePixel(pixel, pixelvalues[pixel]);
260             }
261         }
262
263         writeFrame();
264     }
265
266     // Rotate pixelvalues array
267     // this is used to animate the ice, mist, and fog effects

```

```

267 // This is used to animate the ice, mist, and fog effects.
268
269 function rotate_gradient() {
270     // Schedule refresh
271     wakehandle = imp.wakeup((REFRESHPERIOD), function() {rotate_gradient()}.bindenv(this));
272
273     // Wrap around the end of the array
274     for (local ch = 0 ; ch < 3 ; ch++) {
275         pixelvalues[frameSize - 1][ch] = pixelvalues[0][ch];
276     }
277
278     writePixel(frameSize - 1, pixelvalues[frameSize - 1]);
279
280     //Shift each pixel over by one
281     for (local pixel = 0 ; pixel < (frameSize - 1) ; pixel++) {
282         for (local ch = 0 ; ch < 3 ; ch++) {
283             pixelvalues[pixel][ch] = pixelvalues[pixel + 1][ch];
284         }
285
286         writePixel(pixel, pixelvalues[pixel]);
287     }
288
289     writeFrame();
290 }
291
292 // Blue / white gradient circles the display
293 // No input parameters
294
295 function ice() {
296     // Cancel any previous effect currently running
297     if (wakehandle) { imp.cancelwakeup(wakehandle); }
298
299     // Schedule refresh
300     wakehandle = imp.wakeup((REFRESHPERIOD), function() {rotate_gradient()}.bindenv(this));
301     local opposite = frameSize / 2;
302     local step = (1.0 * WHITE[0]) / ((1.0 * opposite));
303     for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
304         if (pixel < opposite) {
305             pixelvalues[pixel][0] = math.floor((opposite - pixel) * step);
306             pixelvalues[pixel][1] = math.floor((opposite - pixel) * step);
307             pixelvalues[pixel][2] = math.floor(((opposite - pixel) * step) + (pixel * step));
308         } else {
309             pixelvalues[pixel][0] = math.floor((pixel - opposite) * step);
310             pixelvalues[pixel][1] = math.floor((pixel - opposite) * step);
311             pixelvalues[pixel][2] = math.floor(((pixel - opposite) * step) + (frameSize - pixel) * step);
312         }
313
314         writePixel(pixel, pixelvalues[pixel]);
315     }
316
317     writeFrame();
318 }
319
320 // Cyan / white gradient circles the display
321 // No input parameters
322
323 function mist() {
324     // Cancel any previous effect currently running
325     if (wakehandle) { imp.cancelwakeup(wakehandle); }
326
327     // Schedule refresh
328     wakehandle = imp.wakeup((REFRESHPERIOD), function() {rotate_gradient()}.bindenv(this));
329     local opposite = frameSize / 2;
330     local step = (1.0 * WHITE[0]) / ((1.0 * opposite));
331     for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
332         if (pixel < opposite) {
333             pixelvalues[pixel][0] = math.floor((opposite - pixel) * step);
334             pixelvalues[pixel][1] = math.floor(((opposite - pixel) * step) + (pixel * step));
335             pixelvalues[pixel][2] = math.floor(((opposite - pixel) * step) + (pixel * step));

```

```

335         pixelvalues[pixel][4] = math.floor(((opposite - pixel) * step) + (pixel * step));
336     } else {
337         pixelvalues[pixel][0] = math.floor((pixel - opposite) * step);
338         pixelvalues[pixel][1] = math.floor(((pixel - opposite) * step) + (frameSize - pixel) * step);
339         pixelvalues[pixel][2] = math.floor(((pixel - opposite) * step) + (frameSize - pixel) * step);
340     }
341
342     writePixel(pixel, pixelvalues[pixel]);
343 }
344
345 writeFrame();
346 }
347
348 // White gradient circles the display
349 // No input parameters
350
351 function fog() {
352     local baseval = 4;
353
354     // Cancel any previous effect currently running
355     if (wakehandle) { imp.cancelwakeup(wakehandle); }
356
357     // Schedule refresh
358     wakehandle = imp.wakeup((REFRESHPERIOD), function() {rotate_gradient()}.bindenv(this));
359     local opposite = frameSize / 2;
360     local step = (2.0 * WHITE[0]) / ((1.0 * opposite));
361     for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
362         if (pixel < opposite) {
363             pixelvalues[pixel][0] = math.floor((opposite - pixel) * step) + baseval;
364             pixelvalues[pixel][1] = math.floor((opposite - pixel) * step) + baseval;
365             pixelvalues[pixel][2] = math.floor((opposite - pixel) * step) + baseval;
366         } else {
367             pixelvalues[pixel][0] = math.floor((pixel - opposite) * step) + baseval;
368             pixelvalues[pixel][1] = math.floor((pixel - opposite) * step) + baseval;
369             pixelvalues[pixel][2] = math.floor((pixel - opposite) * step) + baseval;
370         }
371
372         writePixel(pixel, pixelvalues[pixel]);
373     }
374
375     writeFrame();
376 }
377
378 // Set the color based on the temperature, and the brightness based on the
379 // factor. Temperature range is -10 to 30 C, where -10 is all-blue, and
380 // 30 is all-red.
381
382 function temp(val, factor) {
383     // Cancel any previous effect currently running
384     if (wakehandle) { imp.cancelwakeup(wakehandle); }
385     factor = factor / TEMPFACORDIV;
386
387     // min temp is -10 C (full-on blue)
388     // max temp is 30 C (full-on red)
389     // scale temp up by 10 so we're dealing only with positive numbers
390     if (TEMPMIN < 0) val += (-1 * TEMPMIN) ;
391     if (val < 0) val = 0;
392     if (val > TEMPRANGE) val = TEMPRANGE;
393
394     // Scale red proportionally to temp, from -10 to 20 C
395     local r_scale = (RED[0] * 1.0) / (TEMPRANGE - TEMPRBOFFSET);
396     local r = (factor * r_scale * (val - TEMPRBOFFSET));
397     if (val < TEMPRBOFFSET) r = 0;
398
399     // Scale green proportionally to temp from 0 to 10, inversely to temp from 10 to 20
400     local g_range = (TEMPRANGE - (2 * TEMPRBOFFSET)) / 2; // green shifts over a 10-degree range
401     local g_max = (2 * TEMPRBOFFSET); // green max occurs at val = 20 (10 degrees)
402     local g_scale = 2 * ((GREEN[1] * 1.0) / g_range);
403     local g = 0;

```

```

404     if ((val < TEMPRBOFFSET) || (val > (TEMPRANGE - TEMPRBOFFSET))) {
405         g = 0;
406     } else {
407         g = factor * g_scale * (g_range - math.abs(g_max - val));
408     }
409
410     // Scale blue inverse to temp, from -10 to 20 C
411     local b_scale = (BLUE[2] * 1.0) / (TEMPRANGE - TEMPRBOFFSET);
412     local b = (factor * b_scale * ((TEMPRANGE - TEMPRBOFFSET) - val));
413     if (val > (TEMPRANGE - TEMPRBOFFSET)) b = 0;
414     for (local pixel = 0 ; pixel < pixelvalues.len() ; pixel++) {
415         pixelvalues[pixel][0] = r;
416         pixelvalues[pixel][1] = g;
417         pixelvalues[pixel][2] = b;
418         writePixel(pixel, pixelvalues[pixel]);
419     }
420
421     writeFrame();
422 }
423 }
424
425 // AGENT CALLBACKS
426
427 agent.on("seteffect", function(val) {
428     local cond = null;
429     local temp = null;
430
431     try {
432         cond = val.conditions;
433         temp = val.temperature;
434     } catch (err) {
435         server.log("Invalid Request from Agent: " + err);
436         return;
437     }
438
439     if (cond.find("Thunderstorm") != null) {
440         if (cond.find("Light") != null) {
441             display.thunder(0);
442         } else if (cond.find("Heavy") != null) {
443             display.thunder(4);
444         } else {
445             display.thunder(2);
446         }
447     } else if (cond.find("Hail") != null) {
448         if (cond.find("Light") != null) {
449             display.hail(0);
450         } else if (cond.find("Heavy") != null) {
451             display.hail(4);
452         } else {
453             display.hail(2);
454         }
455     } else if (cond.find("Drizzle") != null) {
456         if (cond.find("Light") != null) {
457             display.rain(0);
458         } else if (cond.find("Heavy") != null) {
459             display.rain(2);
460         } else {
461             display.rain(1);
462         }
463     } else if (cond.find("Rain") != null) {
464         if (cond.find("Light") != null) {
465             display.rain(3);
466         } else if (cond.find("Heavy") != null) {
467             display.rain(5);
468         } else {
469             display.rain(4);
470         }
471     } else if (cond.find("Snow") != null) {

```

```

472     if (cond.find("Light") != null) {
473         display.snow(2);
474     } else if (cond.find("Heavy") != null) {
475         display.snow(8);
476     } else {
477         display.snow(4);
478     }
479 } else if (cond.find("Ice") != null) {
480     display.ice();
481 } else if ((cond.find("Fog") != null) || (cond.find("Haze") != null) || (cond.find("Dust") != null) || (cond.find("
482     display.fog();
483 } else if ((cond.find("Mist") != null) || (cond.find("Spray") != null)) {
484     display.mist();
485 } else if (cond.find("Clear") != null) {
486     display.temp(temp, 3);
487 } else if (cond.find("Cloud") != null) {
488     display.temp(temp, 2);
489 } else if (cond.find("Overcast") != null) {
490     display.temp(temp, 1);
491 } else {
492     display.temp(temp, 1);
493 }
494 });
495
496 // RUNTIME BEGINS HERE
497
498 // The number of pixels in your chain
499 const NUMPIXELS = 5;
500
501 spi <- hardware.spi257;
502 spi.configure(MSB_FIRST, 7500);
503 display <- RGBWeather(spi, NUMPIXELS);
504
505 // Let the agent know we've just booted, which will trigger a weather update.
506 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.

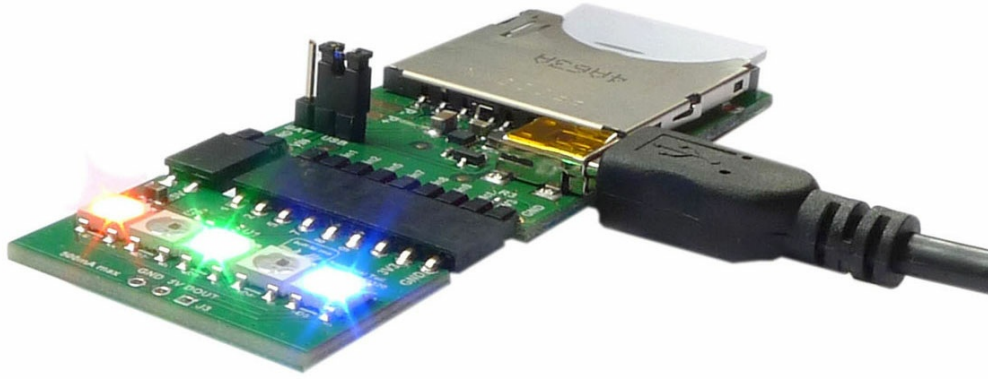
When you have your API key, look for this line in your agent code, up towards the top:

```
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](#)