adafruit learning system

Fair Weather Friend: Internet-Connected Migraine or Allergies Detector

Created by Phillip Burgess

Last updated on 2018-08-22 03:44:05 PM UTC

Guide Contents

Overview

An ***ambient* information display** is an indicator which conveys meaningful data non-verbally. Unlike a computer screen which must be actively read, an ambient display needs no mental "mode shift" to interpret, and often just sits in one's peripheral vision. The *low fuel* light on a car's dashboard is an example of an ambient display.

This project uses our **Feather HUZZAH ESP8266** wireless microcontroller board to pull a forecast from the accuweather.com web site, then distills this to its barest essence: **good news or bad news? Yes or no?**

AccuWeather was chosen because it offers some interesting forecasts beyond the usual temperature or precipitation: there's predictions for health concerns like allergies or arthritis, or for leisure activities like golf or sailing. Personally I'd found their *migraine* forecast to be helpful at times...problem is, I'd forget to check it, and when symptoms have already hit there's usually no recourse but to tough it out. So my idea was to build an early warning system...it periodically checks the forecast and provides a simple reminder. Flashing light = bad news, have the Excedrin ready! *That's all.* No numbers to interpret, no buttons to click on, just an immediate and intuitive course of action.

This can be adapted to *all kinds* of things...it doesn't have to be so grim...consider the site's "astronomy" or "outdoor DIY project" weather forecasts as a more upbeat thing to track. I plan to build a second one as a "good biking weather" indicator!

DISCLAIMER: This is not a medical diagnostic or treatment tool. The example mentioned here — migraine headaches — is serious enough that one should discuss symptoms with their doctor first. I did...as it turns out, my headaches are neither *chronic* nor technically always *migraines*...so finding personal solutions wasn't unreasonable. I'd just happened to notice a frequent correlation with this site's forecast (perhaps due to barometric changes...many folks

experience the same for arthritis symptoms).

Connections

Read through this guide before commiting to any hardware. This project offers a *lot* of potential for customization, and you might want to build something that works a little differently.

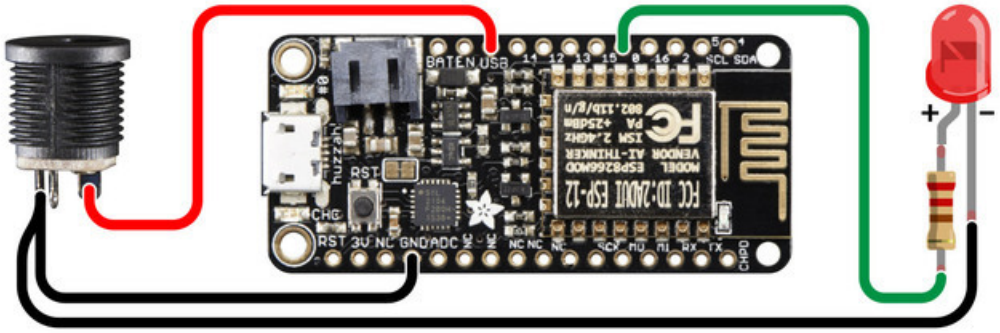
My setup happened to include:

- **Adafruit Feather HUZZAH ESP8266** WiFi microcontroller (http://adafru.it/2821)
- **10mm red LED (https://adafru.it/eby)**
- **220 Ohm resistor (http://adafru.it/2780)**
- **5V DC power supply (http://adafru.it/276)**
- **Panel-mount DC barrel jack (http://adafru.it/610)**
- **2-pin JST Plug & Receptacle Cable Set (http://adafru.it/2880)**

You will also need the usual electronics project bits like **wire** and **soldering paraphernlia**.

There are some **other ESP8266 board variants** (like our HUZZAH ESP8266 breakout (http://adafru.it/2471)) and the code we provide should work on them with **little or no modification**. The **Feather HUZZAH** was chosen because it offers a lot of flexibility for just a few extra bucks...built-in USB programming (No FTDI cable needed), the option to power the project using a USB phone charger you might already have (instead of the DC jack & power supply), and built-in LiPoly battery charging if you wanted to make something that's self-contained. **It all depends on how you want to build and enclose it.**

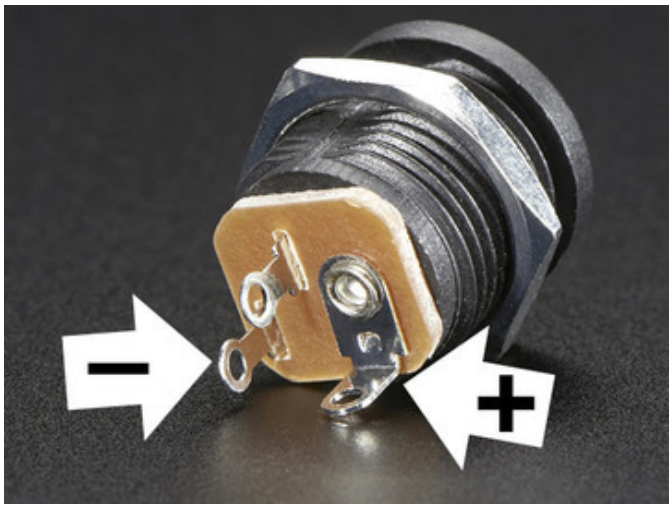
The circuit layout for the one I built looks like this:



The + leg of the **LED** is connected through a **220 Ohm** resistor to **pin 15** on the HUZZAH board (use the labels on the back side of the board, it's easier to tell which pin is which). The **–** leg of the **LED** *as well as* **GND** on the HUZZAH board are *both* connected to the **–** lug of the **DC jack**. The HUZZAH **USB** pin is connected to the + lug.

Not shown here, my build added a **JST plug & receptacle** on the DC power input so I can separate the electronics & enclosure...**yours might not need that**, I'll explain the "why" below.

If you're powering it through the **USB port**, the circuit's even simpler...you can leave off the DC jack and just wire up the LED and resistor (the – leg goes to GND). Or simpler still, leave that out and use the onboard LED on pin #0...no soldering at all. Or add some DotStars or one of our tower lights (http://adafru.it/2994), whatever achieves the look you want and you're willing to add the necessary code for. The single LED was an experiment in minimalism...headaches are horrible and I wanted this a *no-brainer.* (rimshot!)



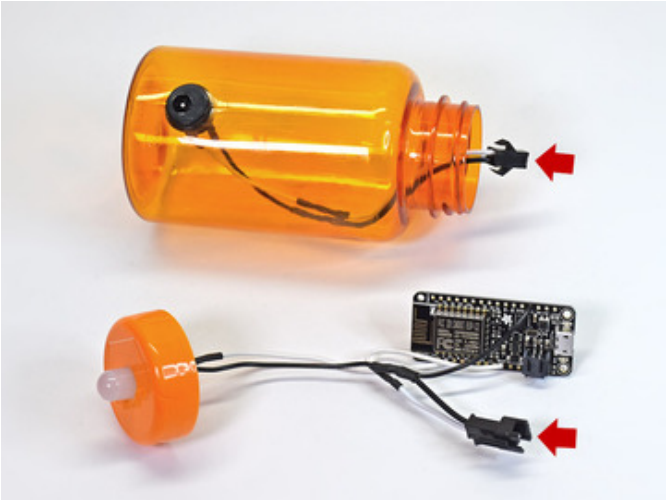
If you use the panel-mount DC jack with a “tip positive” power supply (like the ones we sell), the large lug is the positive (+) connection, and the **outer** of the two small lugs is the ground or negative (–) connection. The **middle lug is not used** here.

If using a DC jack as shown here, do not plug in USB and the DC power supply at the same time!Unplug the DC power source before uploading code to the board.

A Suitable Vessel...

“Dressing up” the project is totally not required, but can be a lot of fun! It adds a bit of arts & crafts rather than being all about electronics or code.

Some of the most clever and “high concept” ambient displays are themed or camoflaged to fit in their environment. How about an umbrella stand that lights for rainy weather, or a whole ceiling constellation that twinkles in anticipation of good stargazing? Mine isn’t nearly that cool...but as it was somewhat health-related, the plastic pill bottle seemed reasonably *thematic.*



My choice of enclosure created a “ship in a bottle” situation. Adding a JST plug & socket (not shown in the wiring diagram) between the DC jack and electronics made it easier to work on the soldering. Most normal enclosures probably won’t need this.

Adding some glass marbles helped camouflage the electronics inside and adds heft so the bottle stays put.

Clearly, this design is **not fit for a home with small children**. Keep *context* in mind when designing, that it’s *safe* and conveys an *appropriate message.*

Code

If this is your first time using the Feather HUZZAH ESP8266, you'll want to begin with our guide for setting that up. There's some software to install and a few persnickety items to be selected *just right* in the Arduino IDE:

Using the Feather HUZZAH ESP8266 with the Arduino IDE(https://adafru.it/IRC)

To confirm that you have the driver installed and IDE properly configured, load the basic "blink" example sketch, edit the LED pin number to **pin 0**, then try uploading to the board. If it won't cooperate, work carefully through each of the steps in the guide linked above.

> Do not continue until you have the "blink" sketch successfully working on the Feather HUZZAH ESP8266 board.

Once you're ready to proceed, copy and paste the following code into a new sketch, and **edit the wireless network name and password** (around line 34) to match your setup.

Further down, just below the code, we'll explain how to set this up for your location...

```
/* ------------------------------------------------------------------------
FAIR WEATHER FRIEND -- a migraine headache forecaster using the Adafruit
HUZZAH ESP8266 WiFi microcontroller (can also work on other ESP8266 boards)
with forecasts provided by AccuWeather.com.  LED on pin 15 shows status:

  Steady on      = One-time initialization
  Fast flicker   = Connecting to network, polling data
  Slow blink     = Symptoms predicted in forecast (24-48 hrs)
  Blip ea. 4 sec = Symptoms not in forecast (sleeping)

Adafruit invests time and resources providing this open source code,
please support open-source hardware by purchasing products from Adafruit!

------------------------------------------------------------------------
Configure the code below with your WiFi credentials.  Then visit
www.accuweather.com and look up the migraine (or other!) forecast for your
location.  Copy the URL into the appropriate spot in the code below.
They have other forecasts that are less grim -- hiking, golf weather, etc.
When changing the forecast type, you'll need to dig through the page's HTML
source to find a string that uniquely identifies the condition sought, while
avoiding false positives.  This code just uses string matches and is not
sophisticated in that regard.  See additional notes later in the code.

DISCLAIMER: THIS IS NOT A MEDICAL DIAGNOSTIC OR TREATMENT TOOL.
------------------------------------------------------------------------*/

#include <ESP8266WiFi.h>

#define LED_PIN       15        // LED+ is connected here
#define POLL_INTERVAL (15 * 60) // Time between server queries (seconds)
#define FAIL_INTERVAL 30        // If error, time before reconnect (seconds)
#define READ_TIMEOUT  10000L    // Client read timeout, milliseconds

char ssid[] = "NETWORK_NAME",
     pass[] = "NETWORK_PASSWORD",
     host[] = "
```

```
     host[] = "www.accuweather.com",
     page[] = "/en/us/new-york-ny/10013/migraine-weather/3709_pc";

// This structure is used during string-matching operations.  Only the
// 'string' element is initialized here; other elements are initialized
// or modified as needed in multiFind().  This code is NOT AVR-friendly;
// PROGMEM strings are not used, it's assumed this will be running on
// an ESP8266 (or ported to other non-AVR board that just normally puts
// const strings in program memory instead of RAM).
struct stringMatch {
  const char * const string;
  uint8_t            stringLength;
  uint8_t            matchedLength;
} matchList0[] = {
  { "<h3>Today</h3>"    },
  { "<h3>Tomorrow</h3>" },
  { NULL                }  // END OF LIST, don't remove this
}; // Can create add'l string match lists here if needed

WiFiClient client;

void setup(void) {
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, HIGH); // Steady on = startup
  Serial.begin(57600);
  Serial.println("Hello!");
}

// STRING-MATCH FUNCTION ------------------------------------------------

// multiFind() scans a connected Client object for one or more strings
// (NULL-terminated stringMatch array 'list'), returns index of the first
// string matched (0 to n-1) or -1 if timeout or no match.
static int8_t multiFind(struct stringMatch *list) {
  uint32_t t;
  char     c;
  uint8_t  i;

  // Reset all stringMatch items prior to search...
  for(i=0; list[i].string; i++) {
    list[i].stringLength  = strlen(list[i].string);
    list[i].matchedLength = 0;
  }

  for(t=millis();;) {
    if(client.available()) {          // Data pending from Client?
      c = client.read();              // Read it
      if(c == 0) break;               // End of data reached, no match
      for(i=0; list[i].string; i++) { // Compare against each stringMatch item...
        if(c == list[i].string[list[i].matchedLength]) { // Matched another byte?
          if(++list[i].matchedLength ==                  // Matched whole string?
               list[i].stringLength) return i;           // WINNER, return index
        } else { // Character mismatch, reset counter to start
          list[i].matchedLength = 0;
        }
      }
      t = millis(); // Reset timeout
    } else if((millis() - t) > READ_TIMEOUT) {
      Serial.println("Timeout");
      break;
```

```
    }
  }
  return -1; // No string match, or timeout
}

void loop() {

  uint32_t t, hi, lo, pauseTime = FAIL_INTERVAL;

  // Fast blink during WiFi connection...
  analogWriteFreq(4);        // 4 Hz
  analogWrite(LED_PIN, 100); // ~10% duty cycle

  Serial.print("WiFi connecting..");
  WiFi.begin(ssid, pass);
  while(WiFi.status() != WL_CONNECTED) {
    Serial.write('.');
    delay(500);
  }
  Serial.println("OK!");

  // Slightly slower (but still quick) blink while searching
  analogWriteFreq(1);        // 1 Hz
  analogWrite(LED_PIN, 100); // ~10% dury cycle

  Serial.print("Contacting server...");
  if(client.connect(host, 80)) {
    Serial.print(F("OK\r\nRequesting data..."));
    client.print("GET ");
    client.print(page);
    client.print(" HTTP/1.1\r\nHost: ");
    client.print(host);
    client.print("\r\nConnection: Close\r\n\r\n");

    // multiFind() searches the incoming stream for a list of possible
    // string matches, returning the index of the found item (or -1 if
    // no match).  Stream position will be immediately after the found
    // item (allowing further searches to be performed from that point
    // forward), or end of stream in -1 case.
    // client.find() is the normal Arduino Stream search function, which
    // looks for a single item.  In this code, we're using multiFind()
    // to skip past some of AccuWeather's false positives, to pick a
    // starting point for a simple string search that more reliably
    // indicates migraine weather in the forecast...
    if((multiFind(matchList0) >= 0) &&
       client.find("Migraine Headache <span>Weather")) {
      // Found it -- migraine weather in next 24-28 hrs.
      Serial.println(F("FOUND"));
      hi = lo = 500; // 1 Hz, 50% duty cycle
    } else { // No match
      Serial.println(F("not found"));
      hi =   10; // Tiny blip
      lo = 3990; // at about 1/4 Hz
    }
    // This is just one example...more complex code might need multiple
    // find() and/or multiSearch() calls with different lists as a sort
    // of decision tree.

    Serial.println("Closing server connection.");
    client.stop();
```

```
    client.stop();
    pauseTime = POLL_INTERVAL;
  } else {
    Serial.println("failed.");
  }
  // WiFi is turned off between server queries, to save a little power if
  // you decide to make this battery-operated.
  Serial.println("Stopping WiFi.");
  WiFi.disconnect();
  analogWrite(LED_PIN, 0);

  // Delay until next server query time. The values of 'hi' and 'lo'
  // determine the LED blink speed. This code doesn't use any low-power
  // sleep techniques, as the ESP8266 doesn't appear to support PWM while
  // sleeping...it has to be blinked in software.
  Serial.print("Pausing for ");
  Serial.print(pauseTime);
  Serial.println(" seconds.");
  t = millis();
  while((millis() - t) < (pauseTime * 1000)) {
    digitalWrite(LED_PIN, HIGH);
    delay(hi);
    digitalWrite(LED_PIN, LOW);
    delay(lo);
  }
}
```

Setting Up for Your Location

Just below the WiFi network name and password are these two lines:

```
host[]  = "www.accuweather.com",
page[]  = "/en/us/new-york-ny/10013/migraine-weather/3709_pc";
```

The first line stays as-is. The second, we need to visit the AccuWeather web site and type our location into the search field...

After bringing up the local forecast, click the "+" button in the forecast dock to show different available forecasts, then click "MIGRAINE" to add this to the dock options. Then click "MIGRAINE" in the dock to display the current forecast.

On the next page we'll show how to set up the software to scan for different things, but for now let's look up migraines.

With the migraine page now loaded, copy the page URL (not the entire site URL, just the page, from the first slash (/) forward), like this:

Paste this over the page[] string in the code...keep the quotes and semicolon...

```
page[]  = "/en/us/elephant-butte-nm/87935/migraine-weather/344484";
```

Make sure all your **Tools→Board** menu options are set up for the Feather HUZZAH ESP8266, plug in USB and click the "upload" button.

> If using a DC power supply, do not have this and USB connected at the same time. Unplug one before connecting the other.

"Reading" the Indicator LED

When first powered up, a steady LED indicates the system is initializing. This should only take a brief moment. If not...

- If the LED doesn't turn on at all: possibly an electrical short, or the LED is on the wrong pin.

Check your wiring if there's a problem!

Steady Blink

A **slow, steady blink** (1/2 second on, 1/2 second off) indicates that a match was found...possible headache- or migraine-aggravating conditions in the next **24 to 48 hours**. In my case, this means being mindful of symptoms and taking medication at the earliest sign of trouble and/or before going to bed. Or go and check the AccuWeather web site to see the specifics.

Off with Periodic "Blip"

If there's **no steady blink**, this means that migraine conditions are **not currently forecast** in the next 48 hours. A tiny "blip" of the LED (every four seconds) lets you know that the program is still alive and is just resting until the next search.

Fast Blink

A **very fast blink** or flutter of the LED lets you know it's connecting to the WiFi network and searching the AccuWeather forecast. It will be extra fast during the WiFi connect phase, then a little slower while processing data from the server...it's a complex web page and may take **several seconds** to complete.

If this mode exits quickly (within a couple seconds), it may be having trouble accessing the wireless network. Check the network name and password in the code. Or it might be a malformed URL. Did you copy the page name correctly? WITH the leading slash, but NO http or domain name included. Connect a USB cable and **check the Serial Monitor** for status messages.

The connection and scan occur every **15 minutes**; it really doesn't need to be very frequent at all. This is set by the 'POLL_INTERVAL' variable in the code...it's the number of seconds between scans. (15 * 60) gives us 15 minutes.

If the wireless network is unreachable, the software will try connecting again in 30 seconds.

Okay! Now, what if we want a *fun* forecast? This gets more involved...

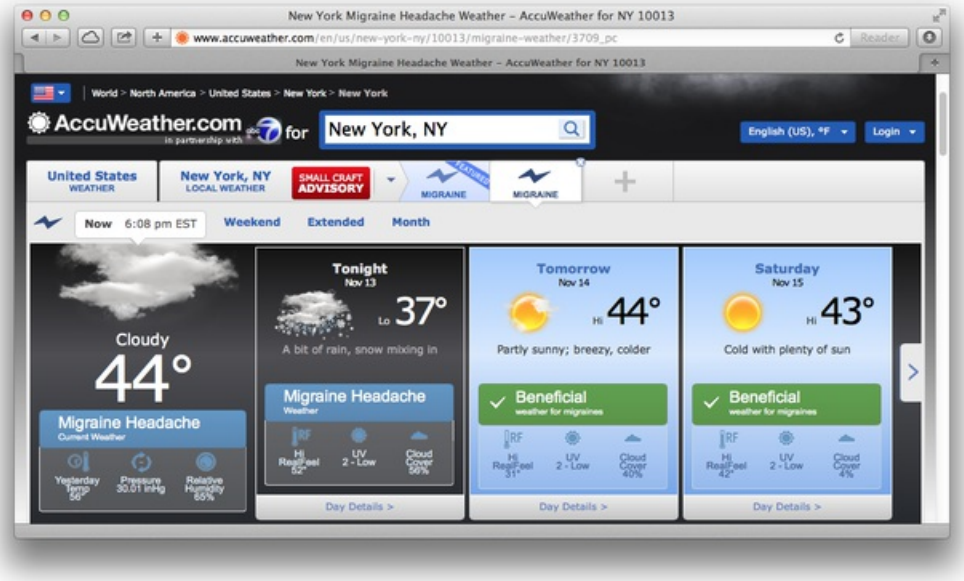
Customize

AccuWeather has an API for programmers, but it's a complex affair requiring a signed contract and is not used here. Instead, our code downloads a forecast web page the same way your browser does...but instead of displaying it, we just pick through the raw HTML source for any kernels of interest. ***"Web scraping,"*** it's called.

The **good news** is that even a small Arduino-compatible board like the Feather HUZZAH ESP8266 can handle this task.

The **bad news** is that it requires some familiarity with **HTML** — the "source code" of web pages — and that the AccuWeather site is particularly complex (a typical forecast page is over 120 kilobytes).

Start by loading up a page of interest in your web browser...like let's try the same migraine forecast...



Right off the bat, I identified a problem...**a bug in AccuWeather's migraine forecast**...the current conditions *always* indicate migraine weather! Any time, any location. I've reported the bug, but it has yet to be fixed. This probably doesn't apply to other predictions, but it's something to check for.

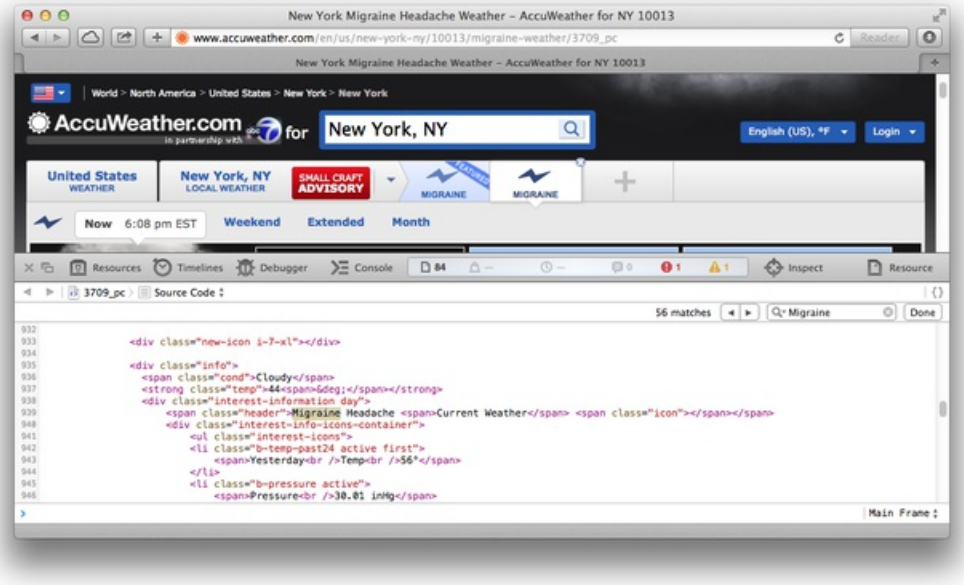
Likewise, nighttime and "early AM" predictions always show migraine weather. Daytime predictions are fine, but **night presents a false positive.**

Fortunately, there's still enough usable information on the page that we can work around this! Software to the rescue...

We know that the first "Migraine Headache" string is right out, and the second is out *if* it follows the "Tonight" forecast. So the trick here is to look specifically for the word "Today" (in which case the forecast immediately following is valid), or "Tomorrow" (also valid). Anything else, like "Tonight" or "Early AM" presented problems and should be ignored.

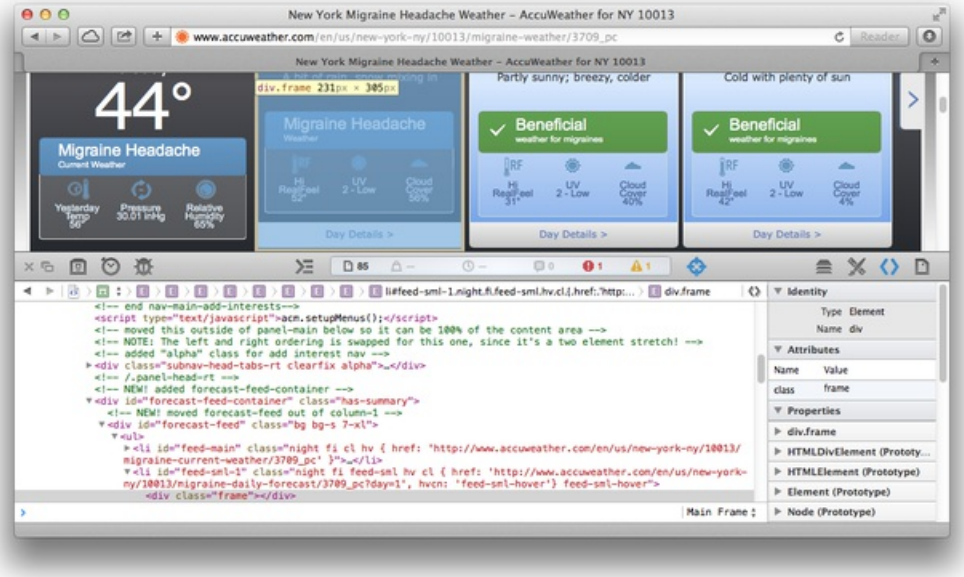
Thing is, words like "Migraine" and "Today" appear *all over the page.* We need to narrow in on a *specific instance* by looking at the HTML source and snagging some of the surrounding formatting tags (normally invisible on the page) along with the text. Each section of the page usually has some unique formatting going on.

Use your browser's "view source" option. If it doesn't offer one or you can't find it, you can also save the page source to a file and examine it in a text editor.



There's 56 occurrences of the word "migraine" on this page, so one really has to dig through the page to find one that correctly indentifies something in the forecast, and not just a random label or banner ad.

Some browsers have a more advanced "inspector" mode that'll take you right to the part of the source corresponding to an element on the page.



Once some reliable "beacon" text is identified, it's copied and pasted verbatim into the Arduino sketch. This must include exact spaces, HTML tags, everything.

In our case, to get past these false positives, we want to look for two different strings...if we find either of those, they then serve as a *starting point* for locating migraine symptoms on the page. As explained above, "Today" provides a viable forecast, as does "Tomorrow". Near the top of the code, there's this list of search strings:

```
} matchList0[] = {
  { "<h3>Today</h3>"    },
  { "<h3>Tomorrow</h3>" },
  { NULL                }  // END OF LIST, don't remove this
}; // Can create add'l string match lists here if needed
```

A couple of things to notice here:

- There's some HTML in each of the strings, to uniquely tell them apart from any "Today" that's on the page. It's exactly as it appears in the HTML source.
- Each string is inside a set of { brackets }, because these are actually C struct elements. Formatting is *super* persnickety here.
- The list of strings is assigned a unique name: matchList0[]. More complex programs might have multiple lists (e.g. matchList1[], etc.), but this example just needs one.

Later in the code...in the loop() function...you'll see this line:

```
    if((multiFind(matchList0) >= 0) &&
```

This searches the page for any of the items in matchList0, and returns the **index of the first string** found (e.g. if "<h3>Today</h3>" is found, a value of 0 is returned), or **-1 if no match** was found.

Usually, when reading the migraine forecast, you *will* find either a Today string (during the day) or a Tomorrow string (any time). Those get us past the false positives, and we can search from that point forward for a valid migraine "beacon" in the HTML source, which is what the next line in the code does...

```
      client.find("Migraine Headache <span>Weather")) {
```

If both conditions are true, then migraine conditions were detected in the forecast. The variables “hi” and “lo” are set to the LED blink on and off times, in milliseconds…slow and steady if a match was found, or a tiny periodic “blip” to indicate there’s nothing in the forecast, but the microcontroller is still active and hasn’t locked up.

The real “trick” to web scraping like this is that you only get ONE PASS forward through the data. You CAN NOT REWIND and try a new search again…one must plan a combination of find() and/or multiFind() calls and if/then/else conditions that will produce an acceptable result.