adafruit learning system

Flora GPS Jacket

Created by Becky Stern

Last updated on 2014-12-23 04:45:19 PM EST

Guide Contents

Overview

Make your coat react to your location with color-changing LEDs! The Flora GPS Jacket tracks your coordinates and then pulses the lights around the collar when you reach your destination. Change the waypoints and range in the provided project code to make your garment light up near your favorite coffee shops or the perfect picnic spot.

Before you begin this project, please freshen up on the following guides containing skills you'll need:

- Getting started with FLORA (http://adafru.it/aSZ)
- Flora RGB Smart Neo Pixels (http://adafru.it/aRT)
- Flora Wearable GPS (http://adafru.it/aRP)
- Conductive Thread (http://adafru.it/aVx)
- Flora with Snaps (http://adafru.it/aUQ)

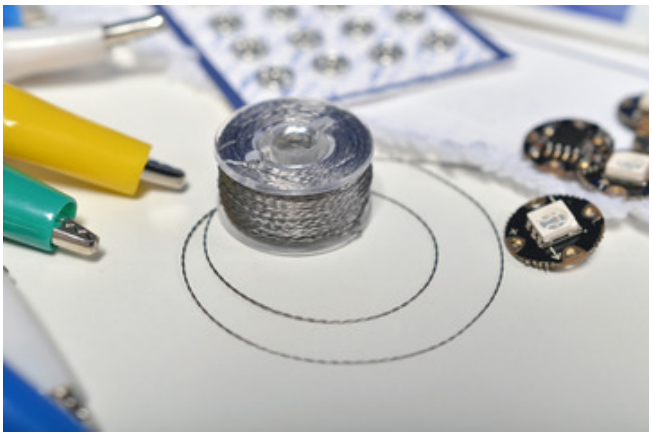
Tools & supplies

The Flora GPS Starter Pack (http://adafru.it/1090) is perfect for this project. It contains all of the supplies you need, just grab a few tools and get to work!

If you don't have the starter pack, you'll need a Flora main board (http://adafru.it/659), Flora GPS (http://adafru.it/1059) and eight Flora pixels (http://adafru.it/1060).

Other parts:

- Battery holder (3xAAA w/JST (http://adafru.it/260) recommended)
- USB cable (A to miniB) (http://adafru.it/260)
- Sewable battery holder (http://adafru.it/653) and coincell battery (http://adafru.it/654) (optional for faster GPS fix)



Conductive thread

3 ply conductive thread (http://adafru.it/641) is best for thick fabrics like that of our jacket. Don't forget sewing needles (http://adafru.it/615) and scissors!

Check out our guide to working with conductive thread! (http://adafru.it/aVx)

Multimeter

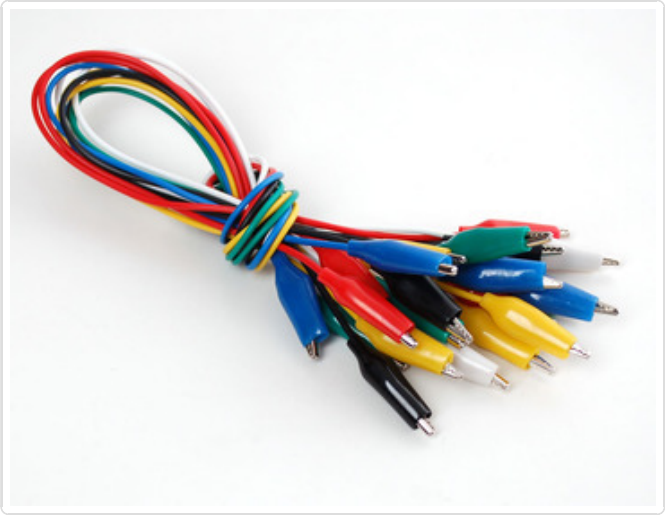
You will need a good quality basic multimeter that can measure voltage and continuity.

Click here to buy a basic multimeter. (http://adafru.it/71)

Click here to buy a top of the line multimeter. (http://adafru.it/308)

Click here to buy a pocket multimeter. (http://adafru.it/850)

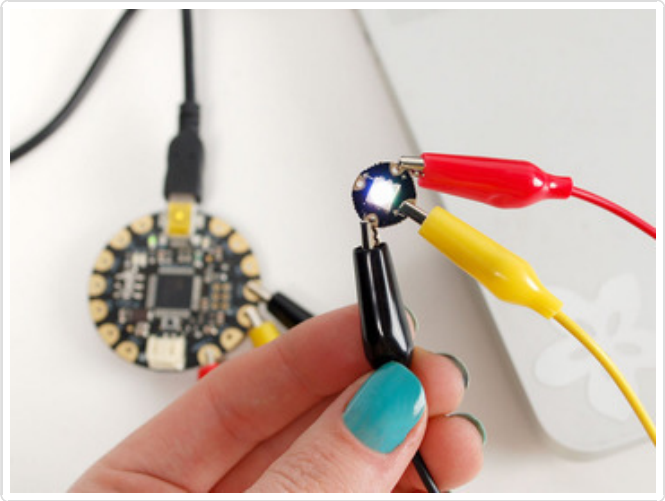
Don't forget to learn how to use your multimeter too! (http://adafru.it/aOy)



Alligator Clips

Great for testing out your circuit or mocking up designs, alligator clips can connect different components or clip to conductive threads and your multimeter for measuring continuity and resistance.

Click here to buy a set of small alligator clip test leads. (http://adafru.it/1008)



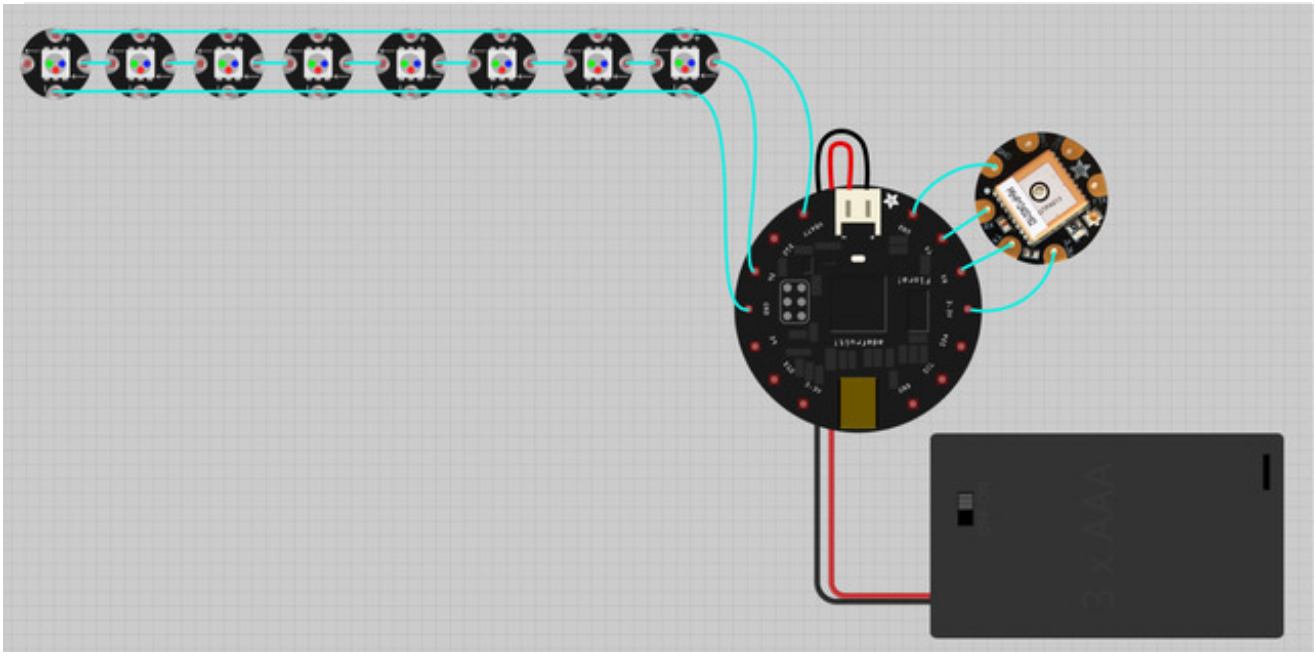
Snaps (optional)

Snaps can be used for quick-connecting circuit boards. You can sew them with conductive thread and/or solder them to circuit boards.

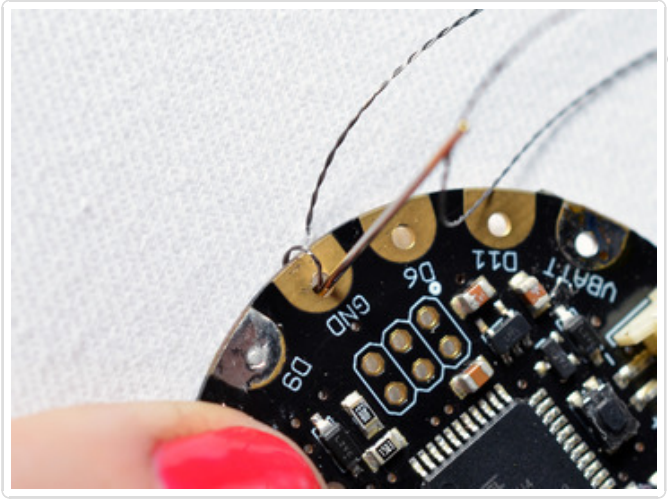
Click here to buy 5mm tin-plated brass snaps. (http://adafru.it/1126)

Check out our guide to using snaps with Flora! (http://adafru.it/aUQ)

Build it

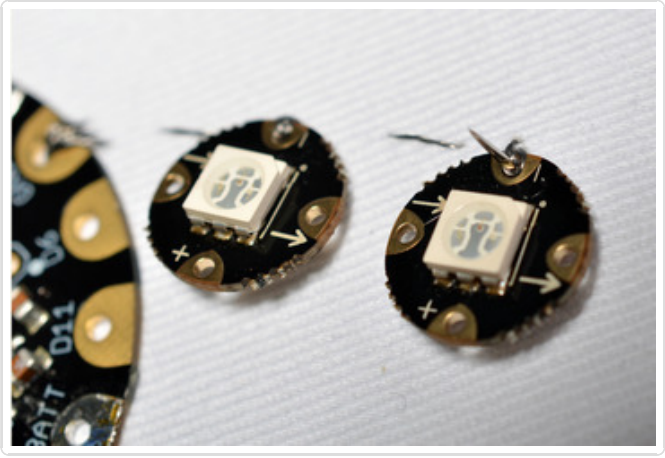


Here's a diagram of the jacket's circuit. You'll chain eight pixels together around the collar and attach the GPS to 3.3v, TX, RX, and ground. A 3xAAA battery holder hides in a pocket and extends through a seam to plug into the JST port on the Flora.



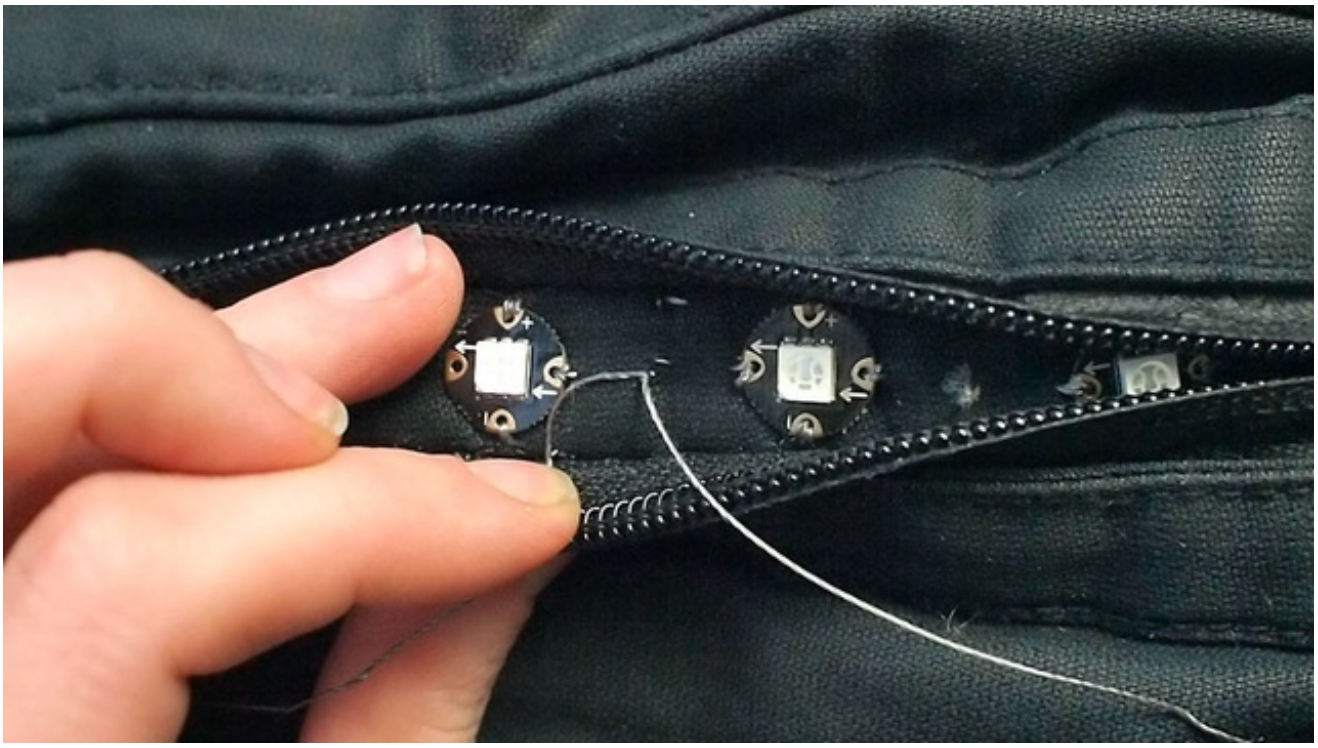
Make the connections in your circuit with conductive thread.

Follow our guide on conductive thread (http://adafru.it/aVx) so you can stitch up your circuit like a pro!

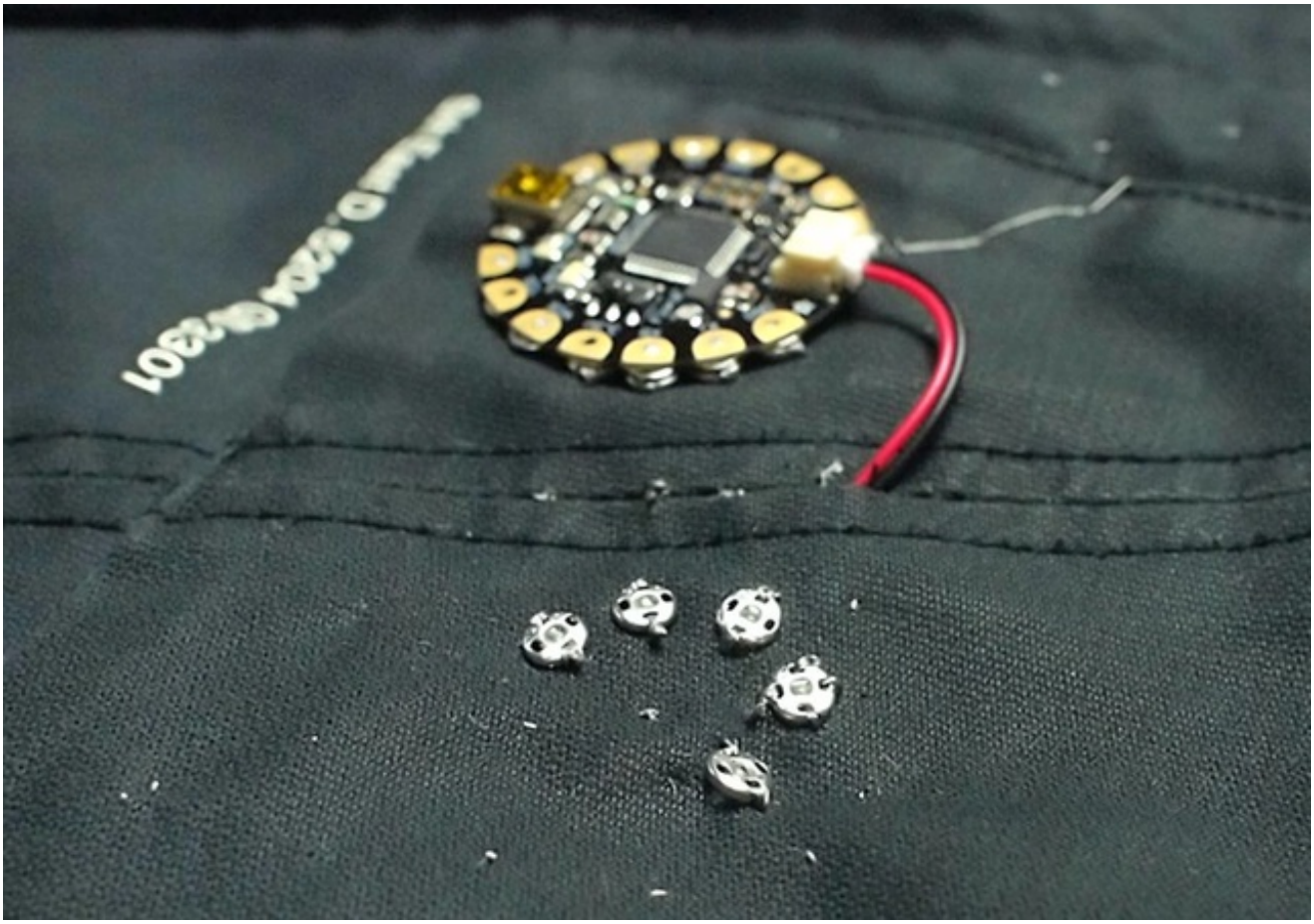


Start by stitching the ground bus from GND on Flora to the (-) pads on each pixel.

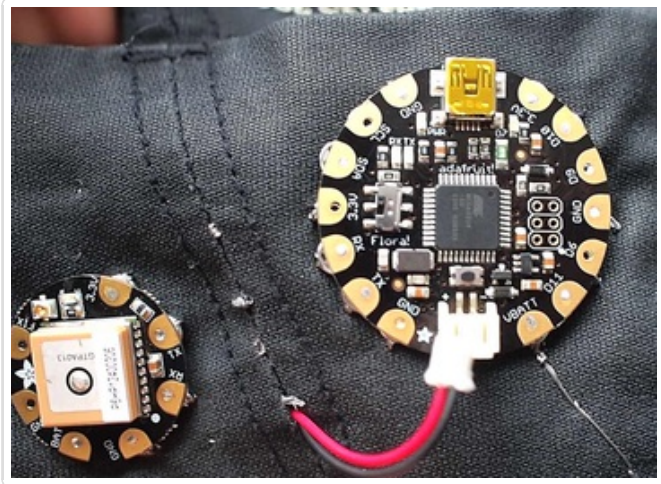
Next stitch the power bus from VBATT to all of the (+) pads on your pixels.



Then connect up the short data connections between each pixel. Be sure to seal the knots with clear nail polish or Fray Check.



Attach the GPS next! You can sew it directly to your garment or make it snappy with our Flora with Snaps tutorial. Connect 3.3v on the Flora to 3.3v, GND to GND, and RX->TX and TX->RX.



You can hide the conductive thread traces just under the top layer of fabric, just make sure the stitches aren't so long that they can move and come into contact with each other. You can see that the thread periodically appears on the outside of the garment to anchor the stitched lines.

You can optionally sew a coincell battery holder on the inside of the garment, connecting + to BAT on the GPS and - to GND. This will help the GPS get a fix on your position faster, since it can keep track of its last known position even when the device is off.

The battery holder is hidden in a pocket, with the JST connector poking out through a seam to connect to the Flora. If your pocket is far away from your circuit, Use one of our handy JST

battery extension cables (http://adafru.it/1131).

The Code

The code for the Flora GPS Jacket is pretty simple and straight forward. We are using the standard Adafruit GPS Library, and the NeoPixel Library. Click the buttons below to download the required libraries.

Download the Adafruit GPS Library

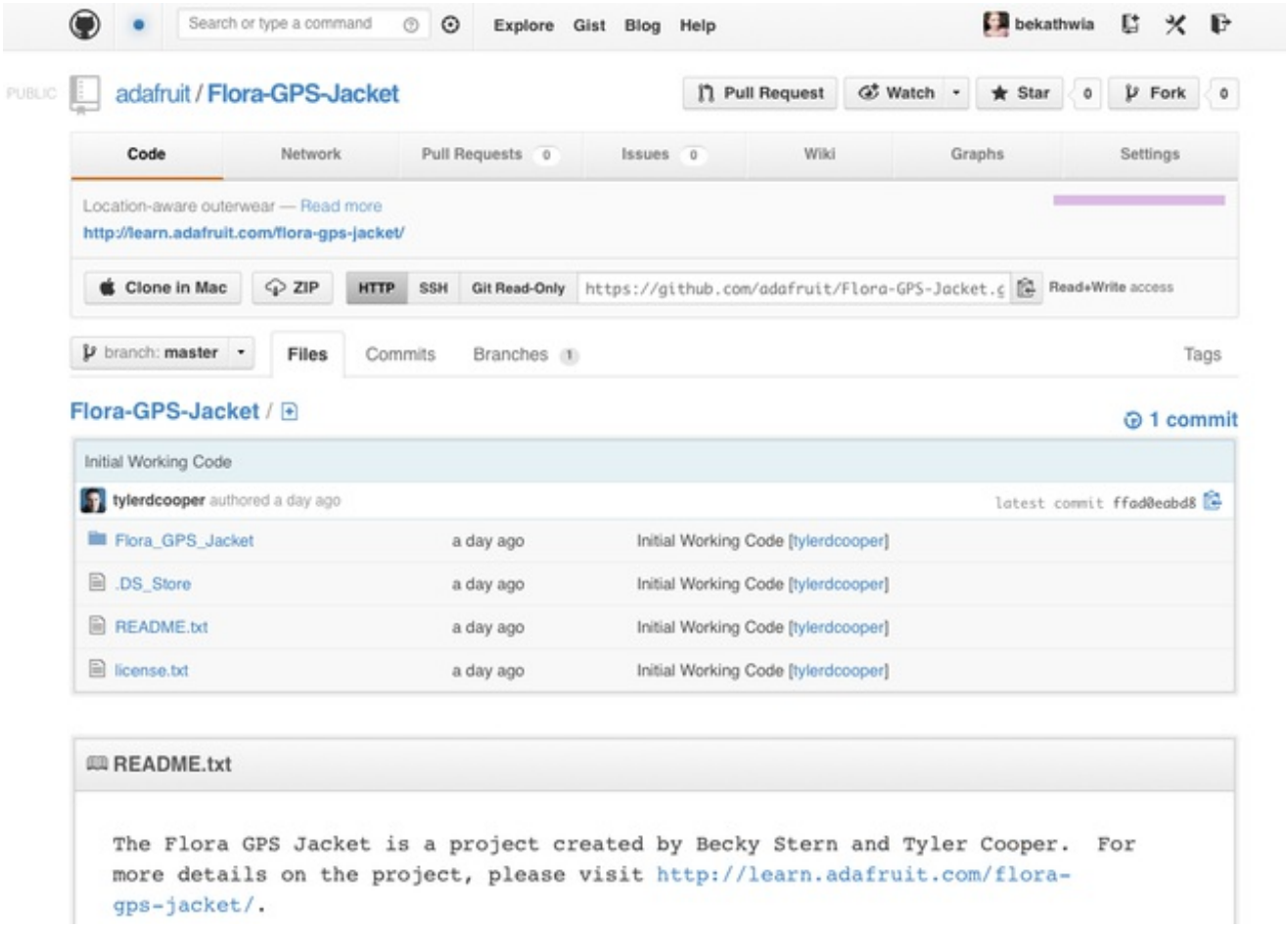
http://adafru.it/aMm

Download the NeoPixel Library

http://adafru.it/aZU

Follow the Flora Sewable NeoPixel tutorial (http://adafru.it/aRT) for more information about installing the library. Test out the pixel sample code to be sure your pixels are functioning properly.

Next follow the Flora GPS tutorial (http://adafru.it/aVC) to test out your GPS module. To make sure your GPS has a direct view of the sky, dip your jacket out the window (but don't let it fall out!). I used a USB extension cable so I could have the jacket out the window but still plugged in to the computer!



If both your pixels and your GPS module are working perfectly in your jacket circuit, head to GitHub to download the Flora GPS Jacket code (http://adafru.it/aVD) (click the "zip" button) or grab the code directly from this page:

```
// Flora GPS + LED Pixel Code
//
// This code shows how to listen to the GPS module in an interrupt
// which allows the program to have more 'freedom' - just parse
// when a new NMEA sentence is available! Then access data when
// desired.
//
// Tested and works great with the Adafruit Flora GPS module
//    ------> http://adafruit.com/products/1059
// Pick one up today at the Adafruit electronics shop
// and help support open source hardware & software! -ada

#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>
#include "Adafruit_NeoPixel.h"
Adafruit_GPS GPS(&Serial1);
```

```
// Set GPSECHO to 'false' to turn off echoing the GPS data to the Serial console
// Set to 'true' if you want to debug and listen to the raw GPS sentences
#define GPSECHO false

#define PIN 6

// this keeps track of whether we're using the interrupt
// off by default!
boolean usingInterrupt = false;

//-------------------------------------------------|
//                  WAYPOINTS                      |
//-------------------------------------------------|
//Please enter the latitude and longitude of your   |
//desired destination:                              |
  #define GEO_LAT               44.995012
  #define GEO_LON              -93.228967
//-------------------------------------------------|

//-------------------------------------------------|
//                  DISTANCE                       |
//-------------------------------------------------|
//Please enter the distance (in meters) from your   |
//destination that you want your LEDs to light up:  |
  #define DESTINATION_DISTANCE   20
//-------------------------------------------------|


// Navigation location
float targetLat = GEO_LAT;
float targetLon = GEO_LON;

// Trip distance
float tripDistance;

boolean isStarted = false;

// Parameter 1 = number of pixels in strip
// Parameter 2 = Arduino pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
Adafruit_NeoPixel strip = Adafruit_NeoPixel(8, PIN, NEO_GRB + NEO_KHZ800);
```

```
uint8_t LED_Breathe_Table[]  = {    80,  87,  95, 103, 112, 121, 131, 141, 151, 161, 172, 182, 192, 202,
            228, 236, 242, 247, 251, 254, 255, 255, 254, 251, 247, 242, 236, 228, 220, 211,
            202, 192, 182, 172, 161, 151, 141, 131, 121, 112, 103,  95,  87,  80,  73,  66,
             60,  55,  50,  45,  41,  38,  34,  31,  28,  26,  24,  22,  20,  20,  20,  20,
             20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,
             20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  22,  24,  26,  28,
             31,  34,  38,  41,  45,  50,  55,  60,  66,  73 };


#define BREATHE_TABLE_SIZE (sizeof(LED_Breathe_Table))
#define BREATHE_CYCLE    5000      /*breathe cycle in milliseconds*/
#define BREATHE_UPDATE    (BREATHE_CYCLE / BREATHE_TABLE_SIZE)
uint32_t lastBreatheUpdate = 0;
uint8_t breatheIndex = 0;

void setup()
{
  // connect at 115200 so we can read the GPS fast enough and echo without dropping chars
  // also spit it out
  Serial.begin(115200);

  // 9600 NMEA is the default baud rate for Adafruit MTK GPS's- some use 4800
  GPS.begin(9600);

  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix data) including altitu
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  // uncomment this line to turn on only the "minimum recommended" data
  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
  // For parsing data, we don't suggest using anything but either RMC only or RMC+GGA since
  // the parser doesn't care about other sentences at this time

  // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);   // 1 Hz update rate
  // For the parsing code to work nicely and have time to sort thru the data, and
  // print it out we don't suggest using anything higher than 1 Hz

  delay(1000);
  // Ask for firmware version
  Serial1.println(PMTK_Q_RELEASE);

    // Start up the LED strip
  strip.begin();

  // Update the strip, to start they are all 'off'
```

```
  strip.show();
}

uint32_t timer = millis();

void loop()                     // run over and over again
{
  // read data from the GPS in the 'main loop'
  char c = GPS.read();
  // if you want to debug, this is a good time to do it!
  if (GPSECHO)
      if (c) Serial.print(c);

  // if a sentence is received, we can check the checksum, parse it...
  if (GPS.newNMEAreceived()) {
    // a tricky thing here is if we print the NMEA sentence, or data
    // we end up not listening and catching other sentences!
    // so be very wary if using OUTPUT_ALLDATA and trytng to print out data
    //Serial.println(GPS.lastNMEA());   // this also sets the newNMEAreceived() flag to false

    if (!GPS.parse(GPS.lastNMEA()))   // this also sets the newNMEAreceived() flag to false
      return;  // we can fail to parse a sentence in which case we should just wait for another
  }


    if (GPS.fix) {
      //Serial.print("Location: ");
      //Serial.print(GPS.latitude, 2); Serial.print(GPS.lat);
      //Serial.print(", ");
      //Serial.print(GPS.longitude, 2); Serial.println(GPS.lon);

      float fLat = decimalDegrees(GPS.latitude, GPS.lat);
      float fLon = decimalDegrees(GPS.longitude, GPS.lon);

      if (!isStarted) {
        isStarted = true;
        tripDistance = (double)calc_dist(fLat, fLon, targetLat, targetLon);
      }

      //Uncomment below if you want your Flora to navigate to a certain destination.  Then modify the h
      /*if ((calc_bearing(fLat, fLon, targetLat, targetLon) - GPS.angle) > 0) {
        headingDirection(calc_bearing(fLat, fLon, targetLat, targetLon)-GPS.angle);
       }
      else {
        headingDirection(calc_bearing(fLat, fLon, targetLat, targetLon)-GPS.angle+360);
      }*/
```

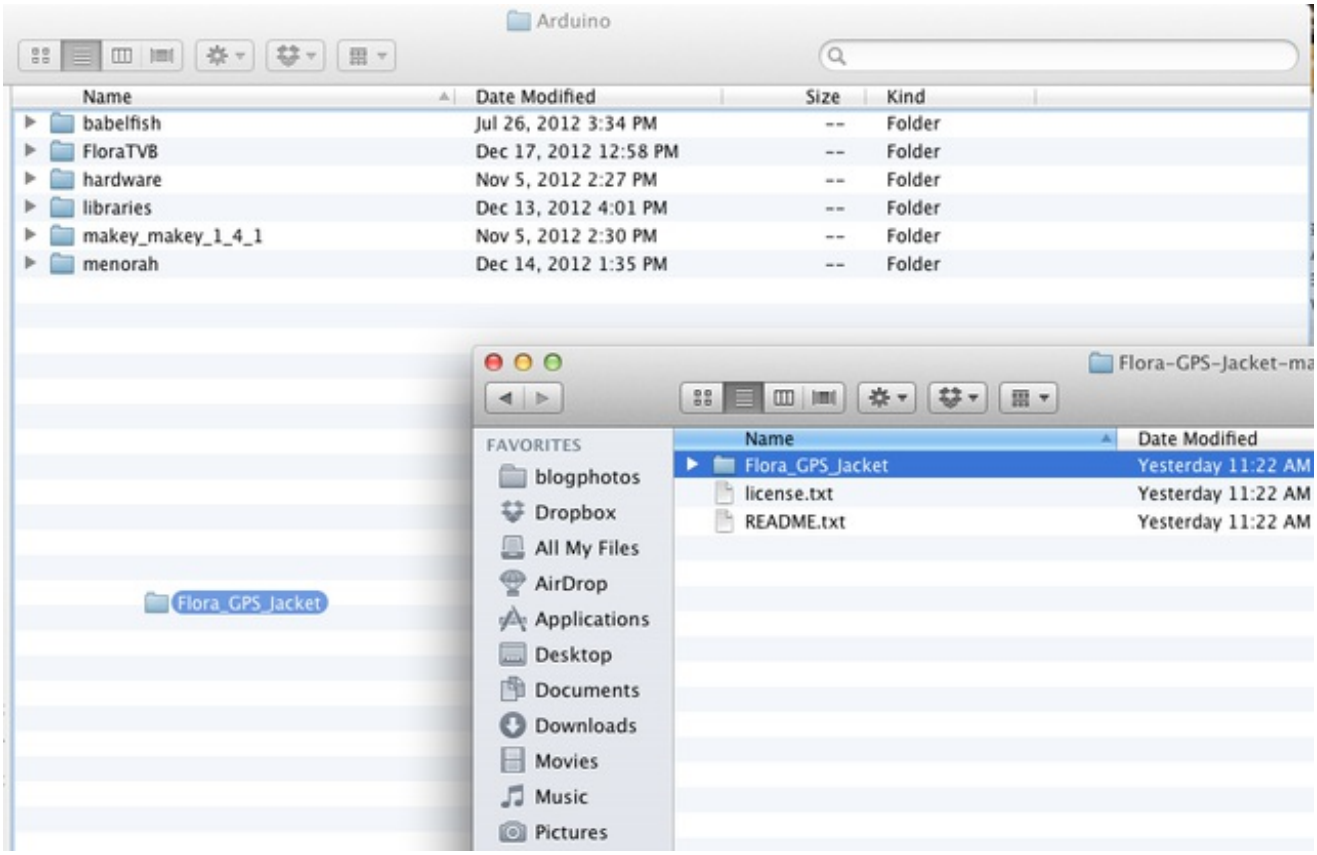
```
      headingDistance((double)calc_dist(fLat, fLon, targetLat, targetLon));
      //Serial.print("Distance Remaining:"); Serial.println((double)calc_dist(fLat, fLon, targetLat, targetLon

    }
  //}

}

int calc_bearing(float flat1, float flon1, float flat2, float flon2)
{
  float calc;
  float bear_calc;

  float x = 69.1 * (flat2 - flat1);
  float y = 69.1 * (flon2 - flon1) * cos(flat1/57.3);

  calc=atan2(y,x);

  bear_calc= degrees(calc);

  if(bear_calc<=1){
    bear_calc=360+bear_calc;
  }
  return bear_calc;
}

void headingDirection(float heading)
{
  //Use this part of the code to determine which way you need to go.
  //Remember: this is not the direction you are heading, it is the direction to the destination (north = fo
  if ((heading > 348.75)||(heading < 11.25)) {
    Serial.println("  N");
    //Serial.println("Forward");
  }

  if ((heading >= 11.25)&&(heading < 33.75)) {
    Serial.println("NNE");
    //Serial.println("Go Right");
  }

  if ((heading >= 33.75)&&(heading < 56.25)) {
    Serial.println(" NE");
    //Serial.println("Go Right");
  }
```

```
if ((heading >= 56.25)&&(heading < 78.75)) {
  Serial.println("ENE");
  //Serial.println("Go Right");
}

if ((heading >= 78.75)&&(heading < 101.25)) {
  Serial.println("  E");
  //Serial.println("Go Right");
}

if ((heading >= 101.25)&&(heading < 123.75)) {
  Serial.println("ESE");
  //Serial.println("Go Right");
}

if ((heading >= 123.75)&&(heading < 146.25)) {
  Serial.println(" SE");
  //Serial.println("Go Right");
}

if ((heading >= 146.25)&&(heading < 168.75)) {
  Serial.println("SSE");
  //Serial.println("Go Right");
}

if ((heading >= 168.75)&&(heading < 191.25)) {
  Serial.println("  S");
  //Serial.println("Turn Around");
}

if ((heading >= 191.25)&&(heading < 213.75)) {
  Serial.println("SSW");
  //Serial.println("Go Left");
}

if ((heading >= 213.75)&&(heading < 236.25)) {
  Serial.println(" SW");
  //Serial.println("Go Left");
}

if ((heading >= 236.25)&&(heading < 258.75)) {
  Serial.println("WSW");
  //Serial.println("Go Left");
}

if ((heading >= 258.75)&&(heading < 281.25)) {
```

```
    Serial.println("  W");
    //Serial.println("Go Left");
   }

   if ((heading >= 281.25)&&(heading < 303.75)) {
    Serial.println("WNW");
    //Serial.println("Go Left");
   }

   if ((heading >= 303.75)&&(heading < 326.25)) {
    Serial.println(" NW");
    //Serial.println("Go Left");
   }

   if ((heading >= 326.25)&&(heading < 348.75)) {
    Serial.println("NWN");
    //Serial.println("Go Left");
   }
}

void headingDistance(float fDist)
{
  //Use this part of the code to determine how far you are away from the destination.
  //The total trip distance (from where you started) is divided into five trip segments.
 Serial.println(fDist);
 if ((fDist >= DESTINATION_DISTANCE)) { // You are now within 5 meters of your destination.
    //Serial.println("Trip Distance: 1");
    //Serial.println("Arrived at destination!");
    int i;
    for (i=0; i < strip.numPixels(); i++) {
      strip.setPixelColor(i, 0, 0, 0);
    }
    strip.show();   // write all the pixels out
   }


   if ((fDist < DESTINATION_DISTANCE)) { // You are now within 5 meters of your destination.
    //Serial.println("Trip Distance: 0");
    //Serial.println("Arrived at destination!");
    breath();
   }

}

unsigned long calc_dist(float flat1, float flon1, float flat2, float flon2)
{
```

```
  float dist_calc=0;
  float dist_calc2=0;
  float diflat=0;
  float diflon=0;

  diflat=radians(flat2-flat1);
  flat1=radians(flat1);
  flat2=radians(flat2);
  diflon=radians((flon2)-(flon1));

  dist_calc = (sin(diflat/2.0)*sin(diflat/2.0));
  dist_calc2= cos(flat1);
  dist_calc2*=cos(flat2);
  dist_calc2*=sin(diflon/2.0);
  dist_calc2*=sin(diflon/2.0);
  dist_calc +=dist_calc2;

  dist_calc=(2*atan2(sqrt(dist_calc),sqrt(1.0-dist_calc)));

  dist_calc*=6371000.0; //Converting to meters
  return dist_calc;
}

// Convert NMEA coordinate to decimal degrees
float decimalDegrees(float nmeaCoord, char dir) {
  uint16_t wholeDegrees = 0.01*nmeaCoord;
  int modifier = 1;

  if (dir == 'W' || dir == 'S') {
    modifier = -1;
  }

  return (wholeDegrees + (nmeaCoord - 100.0*wholeDegrees)/60.0) * modifier;
}

void breath()
{
  uniformBreathe(LED_Breathe_Table, BREATHE_TABLE_SIZE, BREATHE_UPDATE, 127, 127, 127);
}

void uniformBreathe(uint8_t* breatheTable, uint8_t breatheTableSize, uint16_t updatePeriod, uint16_t
{
  int i;

  uint8_t breatheBlu;
```

```
  if ((millis() - lastBreatheUpdate) > updatePeriod) {
    lastBreatheUpdate = millis();


    for (i=0; i < strip.numPixels(); i++) {
      breatheBlu = (b * breatheTable[breatheIndex]) / 256;
      strip.setPixelColor(i, 0, 0, breatheBlu);
    }
    strip.show();

    breatheIndex++;
    if (breatheIndex > breatheTableSize) {
      breatheIndex = 0;
    }
  }
}
```

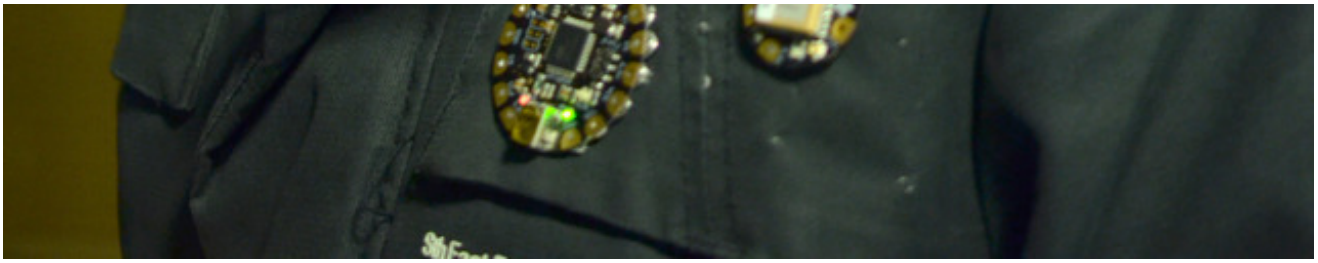


Unzip the package and drag the Flora_GPS_Jacket folder into your Arduino folder (located by default in Documents on OS X). Open the enclosed Flora_GPS_Jacket.ino file in the Adafruit Arduino IDE. (http://adafru.it/aVE)

The code is easy to modify for your own waypoint and range of sensitivity. Adjust these

variables to your own preference. We like iTouchMap (http://adafru.it/aVF) for finding latitudes and longitudes online. Upload your customized code to your jacket.

Wear it!



Wear your GPS jacket with pride! Try changing the colors for different types of notifications, and make this project your own. We like to wear the Flora board on the outside of our jackets so we can show it to people and talk to them about it, but you can just as easily hide the main board inside the lining. The GPS, however, must be facing outward to have direct line of sight to the sky.

To wash the jacket, simply remove the batteries. All other components can get wet while off, but should be thoroughly dry before powering back up.