Track Your Treats: Halloween Candy GPS Tracker

Created by Tony DiCola


Last updated on 2021-11-15 06:33:19 PM EST
# Table of Contents

## Overview

<table>
<thead>
<tr>
<th>Hardware</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Parts</td>
<td>4</td>
</tr>
<tr>
<td>• Wiring</td>
<td>5</td>
</tr>
<tr>
<td>• Assembly</td>
<td>7</td>
</tr>
</tbody>
</table>

## Software

<table>
<thead>
<tr>
<th>GPS Shield</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Track Your Treats - GPS Shield</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FONA</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Track Your Treats - FONA808 &amp; Adafruit IO</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Google Maps</th>
<th>21</th>
</tr>
</thead>
</table>

| Adafruit IO | 26 |
Overview

So you’ve got an awesome Halloween costume planned, but do you know the best places to go trick or treating this year? Some houses have great candy while others might stick you with a set of wax teeth or a granola bar--yuck! Wouldn’t it be nice to keep a record of the best houses for trick or treating? With this Track Your Treats GPS Candy Tracker project you can build a candy container that logs exactly where you found the best Halloween candy. Using the power of the global positioning system (https://adafru.it/iDU) you can optimize your trick or treat route for maximum candy acquisition!

This project uses an Arduino and either a GPS shield (http://adafru.it/1272) or FONA 808 shield (http://adafru.it/2636) to log your trick or treat route. By pressing a button you can note a location as having good candy and view those good candy locations later on a map--great for planning next year’s trick or treating route. With the GPS shield your path will be logged to a microSD card and viewed on Google Maps, and with the FONA 808 shield your path will be logged to Adafruit IO (https://adafru.it/fH9) through the cellular network where a dashboard will display the trick or treating route live as it happens.

Before you get started it will help to be familiar with the following guides:

- Adafruit Ultimate GPS Shield guide (https://adafru.it/iDV)
- Adafruit FONA 808 Shield guide (https://adafru.it/iDW)
- Adafruit IO Basics series (https://adafru.it/iDX)
If you’re new to Arduino be sure to read about getting started (https://adafruit.it/METROXINTR) and programming a basic blink sketch (https://adafruit.it/iDZ) too.

Also note to use the Adafruit IO version of this project you’ll need to be part of the Adafruit IO beta. If you don’t have access yet check out the weekly show and tell show (https://adafruit.it/iE0) and show off a fun project to show your interest.

Continue on to learn about the hardware used in this project.

Hardware

Parts

You’ll need the following parts to build this project:

- **Adafruit Metro** (https://adafruit.it/dCA) (Arduino Uno) - This project is built to work on an Arduino Uno compatible such as the Adafruit Metro 328P (https://adafruit.it/El6). It might work with other Arduinos that are compatible with the ultimate GPS or FONA 808 shields.

- GPS receiver, pick one of the options below:
  - **Ultimate GPS shield** (http://adafruit.it/1272) - This shield will let you save your trick or treating path to the SD card and view it later with Google Maps. This is a simple option that doesn't require any cell or internet service to use with the hardware. You will need a microSD card (https://adafruit.it/eZR) too.
  - **FONA 808 shield** (http://adafruit.it/2636) - This shield will let you log your trick or treating path to Adafruit IO using a GSM cellular phone network. You'll need to have a 2G GSM SIM card and data service (https://adafruit.it/fbO), access to Adafruit IO, a passive GPS antenna (http://adafruit.it/2461), a cellular network antenna (http://adafruit.it/1991), and a small LiPo battery (http://adafruit.it/258).

- **Shield stacking headers** (http://adafruit.it/85) - Not required but will make it a little easier to connect to digital outputs on the shield. If you don't use stacking headers then just solder wires directly to the shield.

- **Momentary button with integrated LED** (http://adafruit.it/1441) - This will be the main interface to the hardware. The LED will tell you if the GPS receiver has a fix and the button is used to mark a location as having good candy.
• 330 ohm resistor - This limits the current through the LED. In most cases you can use any value from around 200 ohms to 500 ohms (but make sure to use one or else you could burn out the LED).
• Candy container - Use an inexpensive plastic container that you can drill a small hole through to mount the button & LED.
• Power source (http://adafruit.it/67) - You'll need a small battery power source to power the Arduino when you're trick or treating. I recommend a 9V battery supply (http://adafruit.it/67) as it's small and will run the hardware for at least a few hours--long enough for your trick or treating run. If you're building the FONA808 version you can actually power everything from the LiPo battery required by the FONA (just move the switch on the shield from the CHRG to RUN position).
• Wire (http://adafruit.it/1970) and soldering tools (http://adafruit.it/136) - You'll need some wire and soldering tools to connect the button & LED to the Arduino, and assemble the shield. For the button you might consider picking up these arcade button quick connects (https://adafruit.it/dMr).

Wiring

Start by assembling your GPS or FONA808 shield and running through its guide to make sure it works:

• Ultimate GPS Shield Guide (https://adafruit.it/iDV)
• FONA808 Shield Guide (https://adafruit.it/iDW)

Then wire the button & LED to the Arduino as shown below. Note the shield isn't shown in the diagram below, but you'll want to connect the shield to the Arduino and then wire to the shield as shown.

Also note the diagram shows the LED and button as separate components, however if you're using one of the illuminated buttons recommended in this guide it will have the LED integrated inside of it. There are 4 terminals on the back of this button. The terminal marked with a + sign is the anode (positive / longer leg) of the LED and is connected to digital pin #6. The terminal marked with a - sign is the cathode (negative / shorter leg) of the LED and is connected to the 330 ohm resistor down to ground. The other two terminals are the momentary button and one should be connected to digital pin #5 and the other to ground.
Here’s an example of the hardware using the GPS shield:

And here’s an example using the FONA808 shield:
Note that you can tightly zip tie the wires to one of the unused mounting holes on the shield as a form of strain relief. This will help protect the solder joints from taking too much force as the hardware bounces around inside the candy container.

Assembly

Putting everything together is easy once you've wired up the components. Just drill a hole in the candy container so the button can be mounted in a convenient to reach location.

For the button recommended in this guide you'll want to drill a ~5/8" hole in the container (a [stepped 'christmas tree' drill bit](https://adafruit.it/iE1) works great for drilling clean holes in plastic):
Mount the button in the container:

Connect the button as shown in the diagram above. Remember the + lead is the anode of the LED and connects to digital pin #6, and the - lead is the cathode which connects through a resistor to ground:

And place the hardware inside the candy container:
That's it, you're ready to load the software for the project!

Note you might consider protecting the Arduino with a case or small cardboard box.

Software

Once you've assembled the hardware you'll need to load an Arduino sketch for the project. If you're new to Arduino it will help to first familiarize yourself with using it by reading:

- Arduino - Getting Started (https://adafru.it/fCN)
- Arduino Libraries (https://adafru.it/fCN)

Make sure you have the latest version of the Arduino IDE (https://adafru.it/fvm) installed (version 1.6.5 as of the time of this guide writing).

Next you'll need to load a few libraries depending on which version of the project you're building. Remember you can use the library manager (https://adafru.it/fCN) to search for and install these libraries right from the Arduino IDE.

For the GPS shield version load these libraries:

- Adafruit GPS library (https://adafru.it/aMm)
For the FONA808 shield version load these libraries:

- Adafruit FONA library (https://adafruit.it/dDC)
- Adafruit SleepyDog library (https://adafruit.it/fp8)
- Adafruit MQTT library (https://adafruit.it/fp6)

If you already have any of the libraries above installed, make sure to update them to their latest versions. In particular the FONA library needs to be at version 1.2.0 or higher or else it won’t work to grab GPS data.

Now download the code this project from its home on GitHub (https://adafruit.it/EI7).

See the following pages for the GPS shield and the FONA versions of the code.

- Track_Your_Treats - This is the version of the project that uses the GPS shield to log location data to a SD card.
- Track_Your_Treats_FONA808 - This is the version of the project that uses the FONA808 shield to log location data to Adafruit IO.

GPS Shield

Track Your Treats - GPS Shield

For the GPS shield version of the project load the Track_Your_Treats sketch by clicking Download: Project Zip in the code listing below.

```c
#include <SPI.h>
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>
#include <SD.h>

// Configuration (you don't normally need to change these values):
#define LED_PIN 6   // Pin connected to an LED that flashes the status
```
of the project.

#define BUTTON_PIN  5 // Pin connected to the button.
#define LOGGING_PERIOD_SEC 15 // Seconds to wait between logging GPS locations.
#define GPS_RX_PIN  8 // GPS receiver RX (pin 8 on the shield).
#define GPS_TX_PIN  7 // GPS receiver TX (pin 7 on the shield)
#define SD_CS_PIN  10 // Chip select pin for SD card.

// Global state (you don't need to change these values):
SoftwareSerial gpsSerial(GPS_RX_PIN, GPS_TX_PIN); // Software serial connection to GPS receiver.
Adafruit_GPS GPS(&gpsSerial); // GPS class to interact with receiver.
File logfile; // SD card log file.
uint32_t logCounter = 0; // Counter until next location log is recorded.

// Halt function called when an error occurs. Will print an error and stop execution while doing a fast blink of the LED. If the watchdog is enabled it will reset after 8 seconds.
void halt(const __FlashStringHelper *error) {
  Serial.println(error);
  while (1) {
    digitalWrite(LED_PIN, LOW);
    delay(100);
    digitalWrite(LED_PIN, HIGH);
    delay(100);
  }
}

// Timer interrupt called every millisecond to check for new data from the GPS.
SIGNAL(TIMER0_COMPA_vect) {
  // Check for new GPS data.
  GPS.read();
  // Decrease the count since last location log.
  if (logCounter > 0) {
    logCounter--;
  }
}

// Log the current GPS location with the specified note.
void logLocation(const char* note) {
  logfile.print(GPS.latitudeDegrees, 6);
  logfile.print(',', 'l');
  logfile.print(GPS.longitudeDegrees, 6);
  logfile.print(',', 'l');
  logfile.print(note);
  logfile.println();
  logfile.flush();
}

void setup() {
  // Initialize serial port.
  Serial.begin(115200);
  Serial.println(F("Track your Treats - Ultimate GPS Shield"));

  // Initialize LED and button.
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);

  // make sure that the default chip select pin is set to output, even if you don't use it:
pinMode(SD_CS_PIN, OUTPUT);

// Initialize SD card (assumes running on Uno, for Leonardo see below).
if (!SD.begin(SD_CS_PIN)) {
// Initialize SD card for Leonardo or other chips (using software SPI)
//if (!SD.begin(SD_CS_PIN, 11, 12, 13)) {
  halt(F("Card init. failed!"));
} // Create the next log file on the SD card.

char filename[15];
strcpy(filename, "GPSLOG00.CSV");
for (uint8_t i = 0; i < 100; i++) {
  filename[6] = '0' + i / 10;
  filename[7] = '0' + i % 10;
  // Create file if it does not exist.
  if (!SD.exists(filename)) {
    break;
  }
}
Serial.print("Using log file: ");
Serial.println(filename);

// Open the log file.
logfile = SD.open(filename, FILE_WRITE);
if (!logfile) {
  halt(F("Failed to open log file!"));
}

// Set the first line of the log file as the column headers.
logfile.println("latitude,longitude,note");
logfile.flush();

// Connect to the GPS receiver and configure it to receive location updates once
// a second.
GPS.begin(9600);
GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY); // Recommended minimum output only
//all we need for this project).
GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ); // Once a second update rate.
GPS.sendCommand(PGCMD_NOANTENNA); // Turn off antenna status.

// Configure timer0 compare interrupt to run and parse GPS data every millisecond.
// See the SIGNAL function further below for the code that is called during this
//interrupt.
OCR0A = 0xAF;
TIMSK0 |= _BV(OCIE0A);
Serial.println("Ready!");
}

void loop() {
  // Parse GPS messages when they are received.
  if (GPS.newNMEAreceived()) {
    GPS.parse(GPS.lastNMEA());
  }

  // Light the LED solid if there's a GPS fix, otherwise flash it on and off once a
  // second.
  if (GPS.fix) {
    digitalWrite(LED_PIN, HIGH);
  } else {
    // No fix, blink the LED once a second and stop further processing.
    digitalWrite(LED_PIN, (millis() / 1000) % 2);
    return;
  }

  // Check if the button is pressed.
  if (digitalRead(BUTTON_PIN) == LOW) {
// Pause a bit to debounce.
delay(100);
if (digitalRead(BUTTON_PIN) == LOW) {
    // Button pressed! Log the current location as having good candy.
    logLocation("Good Candy");
    // Then flash the light 5 times to signal the location was recorded.
    for (int i=0; i<5; ++i) {
        digitalWrite(LED_PIN, HIGH);
        delay(250);
        digitalWrite(LED_PIN, LOW);
        delay(250);
    }
}

// Periodically log the location.
if (logCounter == 0) {
    logLocation("Location");
    logCounter = LOGGING_PERIOD_SEC*1000;
}

Load the code into the Arduino IDE, you should see something like the following:

You don’t need to change anything and are ready to load it on the hardware. However you can skim the #define values at the top of the sketch to see values that change how the sketch works.
To load the sketch on the hardware first make sure you have a microSD card inserted in the GPS shield's microSD card holder. The card should 'click' into place and be held firmly by the card holder.

Then connect the Arduino to your computer using a USB cable. In the Tools -> Board menu of the Arduino IDE make sure the Arduino Uno board is selected. In the Tools - > Port menu make sure the right serial port is selected for the Arduino (it should say Arduino Uno next to it). Click the Sketch -> Upload command to upload the sketch to the Arduino.

Once the sketch uploads open the Tools -> Serial Monitor to see serial output from the hardware. Make sure the baud is set to 115200 in the lower right corner. You should see serial output similar to the following:

```
Track your Treats - Ultimate GPS Shield
Using log file: GPSLOG24.CSV
Ready!
```

The log file name will automatically increment every time the hardware is powered up or reset. Once the hardware is ready to receive GPS signals it will print Ready! and wait for a GPS location fix to be made.

While the hardware is waiting to get a GPS location fix you should see the button LED blink every second. Once a location fix is made the LED will stay solidly lit. You might need to move the hardware outside where it has a clear view of the sky in order for it to get a GPS fix (it can even take up to 30 minutes in some cases!).

If something goes wrong the button LED will blink quickly (around 5 times a second). Look in the serial monitor to see if an error message is displayed, like for example if a microSD card isn't found and the log file can't be created.

Once the hardware has a GPS fix it will start periodically logging location to the file on the microSD card. Every 15 seconds the location will be logged (but you can change this with the #defines at the top of the sketch).

If you press the button the current location will be logged and noted to have good candy, then the LED will blink 5 times to acknowledge the button press. Note you can't log locations with the button if the GPS doesn't have a fix (i.e. the button is blinking every second). Wait until the GPS has a fix and the button stays lit solidly before pressing the button.
That's all there is to using the track your treats GPS shield sketch! Walk around and press the button a few times to get sample data, then power down the Arduino and jump to the [Google Maps page](https://adafruit.it/iE4) to learn how to view the tracker data.

---

**FONA**

**Track Your Treats - FONA808 & Adafruit IO**

For the FONA808 & Adafruit IO version of the project load the `Track_Your_Treats_FONA808` sketch, click Download: Project Zip in the code window below.

```cpp
#include <SoftwareSerial.h>
#include "Adafruit_SleepyDog.h"
#include "Adafruit_FONA.h"
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_FONA.h"

// Configuration (you need to change at least the APN and AIO username & key values):

#define LED_PIN 6   // Pin connected to an LED that flashes the status of the project.
#define BUTTON_PIN 5 // Pin connected to the button.
#define LOGGING_PERIOD_SEC 15 // Seconds to wait between logging GPS locations.
#define FONA_RX 2   // FONA serial RX pin (pin 2 for shield).
#define FONA_TX 3   // FONA serial TX pin (pin 3 for shield).
#define FONA_RST 4  // FONA reset pin (pin 4 for shield)
#define FONA_APN // APN used by cell data service (leave blank if unused).
#define FONA_USERNAME // Username used by cell data service (leave blank if unused).
#define FONA_PASSWORD // Password used by cell data service (leave blank if unused).
```

©Adafruit Industries
#define AIO_SERVER           "io.adafruit.com"  // Adafruit IO server name.
#define AIO_SERVERPORT       1883  // Adafruit IO port.
#define AIO_USERNAME         ""  // Adafruit IO username (see http://accounts.adafruit.com/).
#define AIO_KEY              ""  // Adafruit IO key (see settings page at: https://io.adafruit.com/settings).
#define PATH_FEED_NAME       "treat-path"  // Name of the AIO feed to log regular location updates.
#define GOOD_CANDY_FEED_NAME "treat-good-candy" // Name of the AIO feed to log good candy locations.
#define MAX_TX_FAILURES      3  // Maximum number of publish failures in a row before resetting the whole sketch.

// Global state (you don't need to change this):
SoftwareSerial fonaSS = SoftwareSerial(FONA_TX, FONA_RX); // FONA software serial connection.
Adafruit_FONA fona = Adafruit_FONA(FONA_RST); // FONA library connection.
const char MQTT_SERVER[] PROGMEM = AIO_SERVER; // MQTT server, client, username, password.
const char MQTT_CLIENTID[] PROGMEM = __TIME__ AIO_USERNAME; // (stored in flash memory)
const char MQTT_USERNAME[] PROGMEM = AIO_USERNAME;
const char MQTT_PASSWORD[] PROGMEM = AIO_KEY;
Adafruit_MQTT_FONA mqtt(&fona, MQTT_SERVER, AIO_SERVERPORT, MQTT_CLIENTID, MQTT_USERNAME, MQTT_PASSWORD);

uint8_t txFailures = 0; // Count of how many publish failures have occurred in a row.
uint32_t logCounter = 0; // Counter until next location log is recorded.

// Publishing feed setup (you don't need to change this):
// Note that the path ends in '/csv', this means a comma separated set of values can be pushed to the feed, including location data like lat, long, altitude.
Adafruit_MQTT_Publish path = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/treat-path/csv");
Adafruit_MQTT_Publish goodCandy = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/treat-good-candy/csv");

// Halt function called when an error occurs. Will print an error and stop execution while doing a fast blink of the LED. If the watchdog is enabled it will reset after 8 seconds.
void halt(const __FlashStringHelper *error) {
    Serial.println(error);
    while (1) {
        digitalWrite(LED_PIN, LOW);
        delay(100);
        digitalWrite(LED_PIN, HIGH);
        delay(100);
    }
}

// Timer interrupt called every millisecond to keep track of when the location should be logged.
SIGNAL(TIMER0_COMPA_vect) {
    // Decrease the count since last location log.
    if (logCounter > 0) {
        logCounter--;
    }
}
void logLocation(float latitude, float longitude, float altitude, Adafruit_MQTT_Publish& publishFeed) {
    // Initialize a string buffer to hold the data that will be published.
    char sendBuffer[128];
    memset(sendBuffer, 0, sizeof(sendBuffer));
    int index = 0;

    // Start with '0,' to set the feed value. The value isn't really used so 0 is used as a placeholder.
    sendBuffer[index++] = '0';
    sendBuffer[index++] = ',';

    // Now set latitude, longitude, altitude separated by commas.
    dtostrf(latitude, 2, 6, &sendBuffer[index]);
    index += strlen(&sendBuffer[index]);
    sendBuffer[index++] = ',';
    dtostrf(longitude, 3, 6, &sendBuffer[index]);
    index += strlen(&sendBuffer[index]);
    sendBuffer[index++] = ',';
    dtostrf(altitude, 2, 6, &sendBuffer[index]);

    // Finally publish the string to the feed.
    Serial.print(F("Publishing: "));
    Serial.println(sendBuffer);
    if (!publishFeed.publish(sendBuffer)) {
        Serial.println(F("Publish failed!"));
        txFailures++;
    } else {
        Serial.println(F("Publish succeeded!"));
        txFailures = 0;
    }
}

void setup() {
    // Initialize serial output.
    Serial.begin(115200);
    Serial.println(F("Track your Treats - FONA808 & Adafruit IO"));

    // Initialize LED and button.
    pinMode(LED_PIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT_PULLUP);
    digitalWrite(LED_PIN, LOW);

    // Initialize the FONA module
    Serial.println(F("Initializing FONA....(may take 10 seconds)"));
    fonaSS.begin(4800);
    if (!fona.begin(fonaSS)) {
        halt(F("Couldn't find FONA"));
    }
    fonaSS.println("AT+CMEE=2");
    Serial.println(F("FONA is OK"));

    // Use the watchdog to simplify retry logic and make things more robust.
    // Enable this after FONA is initialized because FONA init takes about 8-9 seconds.
    Watchdog.enable(8000);
    Watchdog.reset();

    // Wait for FONA to connect to cell network (up to 8 seconds, then watchdog reset).
    Serial.println(F("Checking for network..."));
    while (fona.getNetworkStatus() != 1) {
        delay(500);
    }
}
// Enable GPS.
fonatable.enableGPS(true);

// Start the GPRS data connection.
Watchdog.reset();
fonatable.setGPRSNetworkSettings(F(FONA_APN), F(FONA_USERNAME), F(FONA_PASSWORD));
delay(2000);
Watchdog.reset();
Serial.println(F("Disabling GPRS");
fonatable.enableGPRS(false);
delay(2000);
Watchdog.reset();
Serial.println(F("Enabling GPRS");
if (!fonatable.enableGPRS(true)) {
    halt(F("Failed to turn GPRS on, resetting..."));}
Serial.println(F("Connected to Cellular!");)

// Wait a little bit to stabilize the connection.
Watchdog.reset();
delay(3000);

// Now make the MQTT connection.
int8_t ret = mqtt.connect();
if (ret != 0) {
    Serial.println(mqtt.connectErrorString(ret));
    halt(F("MQTT connection failed, resetting..."));}
Serial.println(F("MQTT Connected!");)

// Configure timer0 compare interrupt to run and decrease the log counter every millisecond.
OCR0A = 0xAF
TIMSK0 |= _BV(OCIE0A);
}

void loop() {
    // Watchdog reset at start of loop--make sure everything below takes less than 8 seconds in normal operation!
    Watchdog.reset();

    // Reset everything if disconnected or too many transmit failures occurred in a row.
    if (!fonatable.TCPConnected() || (txFailures >= MAX_TX_FAILURES)) {
        halt(F("Connection lost, resetting..."));}

    // Grab a GPS reading.
    float latitude, longitude, speed_kph, heading, altitude;
    bool gpsFix = fonatable.getGPS(&latitude, &longitude, &speed_kph, &heading, &altitude);

    // Light the LED solid if there's a GPS fix, otherwise flash it on and off once a second.
    if (gpsFix) {
        digitalWrite(LED_PIN, HIGH);
    } else {
        // No fix, blink the LED once a second and stop further processing.
        digitalWrite(LED_PIN, (millis() / 1000) % 2);
        return;
    }

    // Check if the button is pressed.
    if (digitalRead(BUTTON_PIN) == LOW) {
        // Pause a bit to debounce.
        delay(100);
        if (digitalRead(BUTTON_PIN) == LOW) {
            // Button pressed! Log the current location to the good candy feed.
logLocation(latitude, longitude, altitude, goodCandy);
// Then flash the light 5 times to signal the location was recorded.
for (int i=0; i<5; ++i) {
  digitalWrite(LED_PIN, HIGH);
  delay(250);
  digitalWrite(LED_PIN, LOW);
  delay(250);
}

// Periodically log the location.
if (logCounter == 0) {
  // Log the current location to the path feed, then reset the counter.
  logLocation(latitude, longitude, altitude, path);
  logCounter = LOGGING_PERIOD_SEC*1000;
}

Load the sketch in the Arduino IDE, you should see something like the following:

At the top of the sketch are #define values which configure how the code will behave. There are a few values you will need to change:

- FONA_APN - Set this to the APN that your cellular network provider requires for accessing their data network. With a Ting SIM card in the US the value
"wholesale" is what you want, but for other providers you might need to contact their support to get the value.

- **FONA_USERNAME & FONA_PASSWORD** - If your cellular network provider requires a username and password to access their data network then put those values here. If they aren't required then just leave the default blank values.
- **AIO_USERNAME** - This needs to be set to your Adafruit account username, which you can find at [https://accounts.adafruit.com/] (https://adafruit.it/dyy) under the Username field.
- **AIO_KEY** - This needs to be set to your Adafruit IO key, which you can find from the Adafruit IO settings page for your account at [https://io.adafruit.com/settings] (https://adafruit.it/fVT) by clicking View AIO Keys.

The other configuration values don't necessarily need to be changed, but it's a good idea to skim them to see what they do.

To load the sketch on the hardware first make sure the hardware is ready. You must have a cellular antenna and GPS antenna connected to the FONA808 shield. In addition you need a LiPoly battery connected to the FONA808 shield so it can power the cellular modem.

Then connect the Arduino to your computer using a USB cable. In the Tools -> Board menu of the Arduino IDE make sure the Arduino Uno board is selected. In the Tools -> Port menu make sure the right serial port is selected for the Arduino (it should say Arduino Uno next to it). Click the Sketch -> Upload command to upload the sketch to the Arduino.

Once the sketch uploads open the Tools -> Serial Monitor to see serial output from the hardware. Make sure the baud is set to 115200 in the lower right corner. You should see serial output similar to the following:

```
Track your Treats - FONA808 & Adafruit IO
Initializing FONA....(may take 10 seconds)
FONA is OK
Checking for network...
Disabling GPRS
Enabling GPRS
Connected to Cellular!
MQTT Connected!
```

While the hardware is waiting to get a GPS location fix you should see the button LED blink every second. Once a location fix is made the LED will stay solidly lit. You might need to move the hardware outside where it has a clear view of the sky in order for it to get a GPS fix (it can even take up to 30 minutes in some cases!).
If something goes wrong the button LED will blink quickly (around 5 times a second). Look in the serial monitor to see if an error message is displayed, like for example a cellular data connection can't be made (check your SIM card and cell account has data access), or if the MQTT/Adafruit IO connect fails (check your AIO username and key values).

Once the hardware has a GPS fix it will start periodically logging location to Adafruit IO. Every 15 seconds the location will be logged (but you can change this with the #defines at the top of the sketch).

If you press the button the current location will be logged and noted to have good candy, then the LED will blink 5 times to acknowledge the button press. It takes a few moments for the location to be logged after pressing the button so be sure to hold it down until it starts blinking 5 times. Also note you can’t log locations with button if the GPS doesn’t have a fix (i.e. the button is blinking every second). Wait until the GPS has a fix and the button stays lit solidly before pressing the button.

Once the sketch is running walk around and press the button a few times to get sample data, then jump to the Adafruit IO page (https://adafruit.io) to learn how to view the tracker data on Adafruit IO.

---

**Google Maps**

To view data logged by the GPS shield you can use the Google My Maps (https://adafruit.io) service to load and display the CSV data saved to the SD card.

Start by copying the CSV file off of the GPS shield’s SD card and moving it to your computer. Then in a web browser access Google My Maps at:


Note that if you receive an error that you can’t access Google My Maps then you probably have a Google apps account and need to ask the administrator of it to enable Google My Maps (https://adafruit.io).

Once Google My Maps loads you’ll want to click Create a new map:
Now click the blue Import link under the new Untitled layer in the upper left:

In the dialog that loads click the Select a file from your computer button and navigate to the CSV file you copied from the SD card to select it:
Once Google My Maps loads the file it will ask you to choose the columns that contain location information. Make sure only the latitude and longitude columns are checked, then click Continue:

In the next dialog select the note column as the title for the markers, then click Finish:
You should see all of the locations loaded as points on the map, awesome!

Now to change how each type of location is displayed click the Uniform style link with the paint roller icon on the layer in the upper left:

Then click the Uniform style combo box and change it to the note column under Style by data column:
Click the X in the upper right of the Group places by dialog to clear it. You should see inside the layer there are two groups, one called Location with a marker for each periodic location log and another called Good Candy with a marker for each location the button was pressed.

To better view the good candy locations you can change the styling of the markers. Highlight the Good Candy group and click the paint can icon on the right side of it, this will bring up a menu to change the marker style for the good candy locations:
Pick a different color and icon for the good candy locations. Then do the same for the other group, Location, to change how the normal location markers are displayed. You can adjust the marker style to anything you desire!

Change the map title by clicking the Untitled map title in the upper left. Give the map a good name and description.

For example here's a map with red stars for the good candy locations and green markers for the other locations:

![Map with red stars for good candy locations](image)

That's all there is to viewing the track your treats GPS data on Google My Maps!

The map should automatically be saved to Google Drive so you can access it again in the future. Use the data you learned to help improve and optimize future trick or treat runs for maximum candy acquisition!

## Adafruit IO

To view data from the FONA808 shield on Adafruit IO, first make sure you have access to the [Adafruit IO](https://adafruit.io/fH9) beta (if you don’t have access yet, a great way to get an invite is to show off a fun internet of things-related project on the [weekly show & tell show](https://adafruit.it/iE0)).
Next you’ll want to familiarize yourself with Adafruit IO by checking out the guides in this Adafruit IO series (https://adafruit.it/iDX), in particular:

- Adafruit IO Basics: Feeds (https://adafruit.it/ioA)
- Adafruit IO Basics: Dashboards (https://adafruit.it/f5m)

Then make sure you've assembled and run the FONA808 version of the sketch from the previous page. Once the hardware has run for a bit there should be data populated in Adafruit IO and you can build a dashboard to view it.

To build the dashboard, start by navigating to Adafruit IO at:


You should see a welcome dashboard like the following:

Click My Dashboards in the upper left to go to the main Dashboard page:

Now create a new dashboard by clicking Create Dashboard in the upper right:
Give the dashboard a good name like 'Track Your Treats', then click Create Dashboard:

The new dashboard will be empty, however you can click New Block in the upper right to add a map block:

Scroll down to the map block and click Create:
Once the feed list loads you want to find one of the feeds created by the sketch. By default the project uses two feeds:

- **treat-path** - This feed stores the regular location updates (every 15 seconds) so you can see the full path of trick or treating that you followed.
- **treat-good-candy** - This feed only stores locations where the button was pressed, i.e. houses that had good candy.

Let's start by creating a map block for the treat path. You can scroll down to find the treat-path feed, or type in treat-path to the search bar to filter to just that feed to find it:

Click the Choose action to select the treat-path feed for this map block, then click Next Step to configure the block:
Give the block a good title like 'Candy Path', and then put in a large value for the hours of history, like 9999. Right now the map block is made to display recent location data so putting in a large value like 9999 will display all the map data regardless of how old it is. In the future there could be new options to simplify showing all map data, but for now use a large value if you want to have the data visible on the map for a long time.

You can also change the map style from high contrast to satellite or other map views. High contrast looks great on the dashboard so we'll stick with it for now.

Click Create Block to add the new map block to the dashboard:

You can resize and reconfigure the map by going into edit mode. Click the Edit button in the top left and notice the block changes slightly to show more options:
You can click the trash can in the upper right corner of the block to delete it. Or click the gear icon next to the trash can to go back to the block configuration and change its title, hours of history, etc.

When you mouse over the block you will see a resize icon in the lower right corner of it. Grab the icon with the mouse and drag it out to resize the block:

Once you've finished resizing and editing the block click Done Editing in the upper left to save the changes.

Now let's add the good candy location as another map block on the dashboard. Just like before click New Block to create a new block. This time select the treat-good-candy feed (make sure you've run the hardware and pressed the button a few times to have this feed created and populated with data):
Configure this good candy map with a title like 'Good Candy', and again put a large value like 9999 for hours of history. Then click Create Block to add the block to the dashboard:

Resize the good candy map block just like you resized the path block. You can fit both maps on the dashboard to have a nice view of your full trick or treat path, and the houses that had good candy:

An awesome feature of Adafruit IO is that dashboards will update in real-time as data is added to the map. This means you and your friends & family can watch the treat
path update as you walk around collecting candy. The periodic location updates and the good candy location button presses will pop onto the dashboard as they occur.

If you’d like to make the dashboard public so other people can view it, click the lock icon in the upper right to unlock the dashboard and make it public. Then you can copy the URL from the browser and send it to someone else to view.

Congratulations on building the track your treats candy tracker with the FONA808 shield & Adafruit IO! Use the data you learned to help improve and optimize future trick or treat runs for maximum candy acquisition!