FITBIT STEP COUNTER

Bolded words will have definitions at the bottom of each section

STEP 1: Fitbit Authorization

Fitbit uses OAuth 2. There is a lot of documentation at https://dev.fitbit.com/docs/. You will need to make a fitbit dev account. Once you make the account, you will need to make an application. An application is basically a project that you are working on, and it asks you questions about what you are doing with the project (is it personal, or for a business). When making your application, it will ask you for the **redirect URI** (also labeled Callback URI). This and the name of your application are the two main parts. The other sections, you can more or less make up what you put in. It asks you for your organization and website just to be sure that it is real, so type in real websites for those sections, but they won't be used. Click save, and your application will be made. You will also be given a client ID and client secret. These allow you to get an **access code** which you must trade in for an **access token** and **refresh token**.

Type this website into your browser (make sure to put in YOUR client id and redirect URI in place of {client ID} and {redirect URI} respectively):
https://www.fitbit.com/oauth2/authorize?response_type=code&client_id={client ID}&redirect_uri={redirect URI}&scope=activity
!!NOTE!! make sure that the redirect URI is URI encoded. Here is a website to encode your URI: http://www.url-encode-decode.com/
This will then bring you to a page that asks you to sign into fitbit and allow the application you made access to the person's fitbit. Once you click allow, it will bring you to your redirect URI. The URL in your browser should be{redirect URI}/?code={string of numbers and letters here}#_=_ The string of numbers and letters is your access code. You will use this in the next step to get your access token and refresh token.

After granting access to the application and getting an access code, you will make a post command to get an access token which will be used to actually talk to the fitbit api and get information from the user's activity data. Using HTTP requests, make a POST to https://api.fitbit.com/oauth2/token with headers

Authorization: Basic {Base64 encoded string "client ID:client secret"}
Content-Type: application/x-www-form-urlencoded

The header "Content-Type" tells you the type of body that should be in the POST command. This is what the body should be

client_id={client ID}&grant_type=authorization_code&redirect_uri={uri encoded redirect uri}&code={authorization code that you got in the last step}

Just a note: the authorization code only lasts 10 min, so you must do this within 10 min of getting the authorization code. Once you make the post command, you will receive back a json string with an access token and a refresh token and user id (to be used later). The access token

is used to make GET and POST commands to the fitbit website. It can only be used for 8 hours. Then you must use the refresh token to get a new access token (similar to how you used the access code to get the original access token and refresh token). The POST command to use the refresh token to get a new access token again to https://api.fitbit.com/oauth2/token with headers

Authorization: Basic {Base64 encoded string “client ID:client secret”}
Content-Type: application/x-www-form-urlencoded

And only now the body is different, and it is

grant_type=refresh_token&refresh_token={refresh token}

This will send you a new access token which you can use for the next 8 hours and a new refresh token which you will have to use when the access token runs out again.

Definitions
redirect URI: A url that the person is redirected to once they have allowed the application access to their account. You can create your own redirect uri as a google site or a wordpress site.
access code: a string of letters and numbers for when a person has allowed your application to access their account. It is traded in for an access token. You should only have to get an access code once (unless you want to get information from a different user). From now on you should be able to use the refresh token to get a new access token.
access token: a string of numbers and letters that is used when making a GET or POST command to get information from the user’s activity data. It lasts 8 hours and then it is no longer valid. **refresh token**: a string of numbers and letters that is used to get a new access token and refresh token. That way you do not need to get a new access code every 8 hours.

* Required

Application Name *

CEEO Step Counter

Description *

Tracks the steps of a group using LabVIEW code.

Application Website *

https://www.tufts.edu/

Organization *

Tufts CEEO

Organization Website *

http://ceeo.tufts.edu/

OAuth 2.0 Application Type *

○ Server ◉ Client ○ Personal

Callback URL *

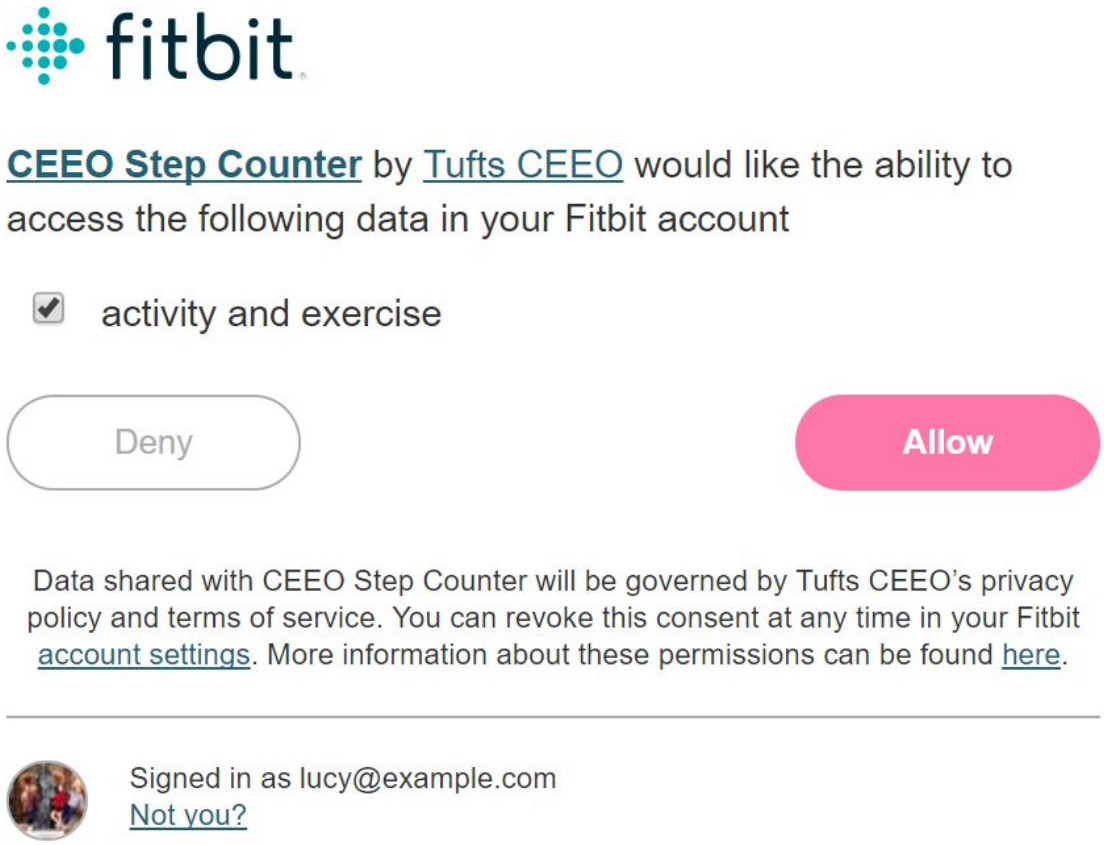
https://ceeostepcounter.wordpress.com

Default Access Type *

○ Read & Write ◉ Read-Only

+ Add a subscriber

Save Cancel

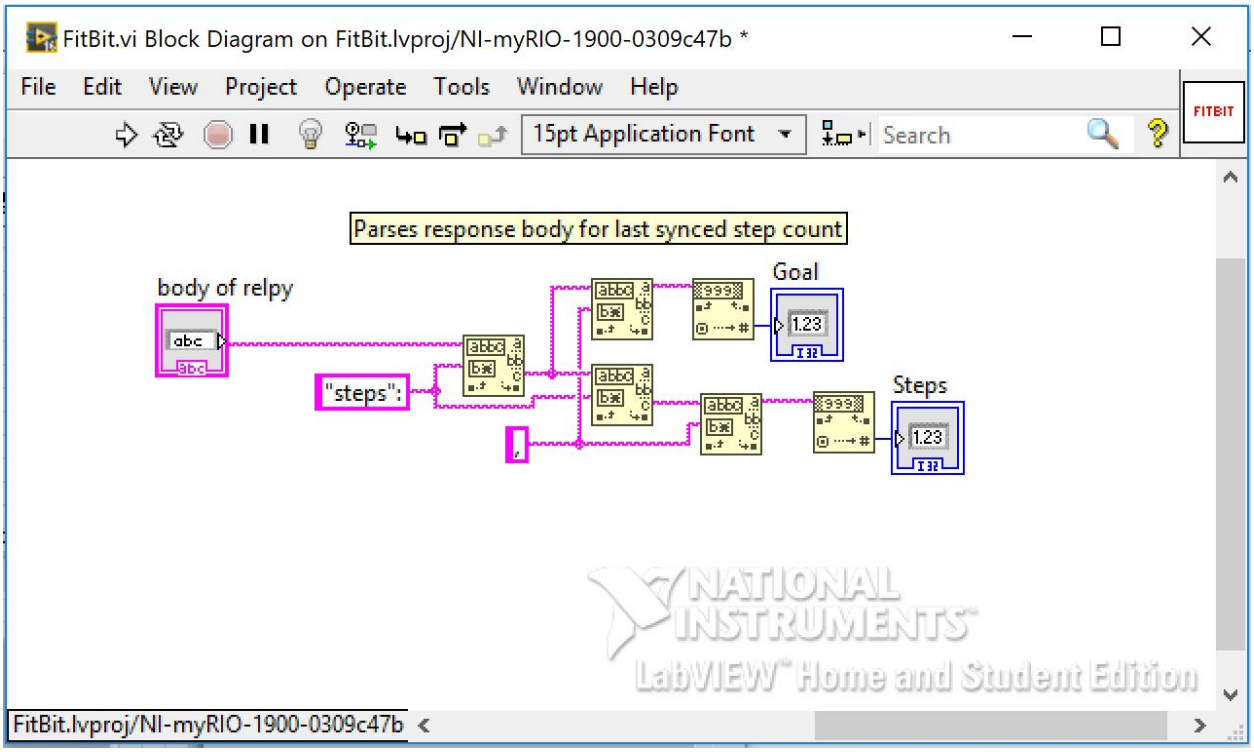


STEP 2: REST api calls to fitbit

Now that you have access to the fitbit api, you can start making GET requests in order to get the activity data. For this project all you need is the step count of each person. To get this it is just a GET command to https://api.fitbit.com/1/user/{**user ID**}/activities/date/today.json This will get you today's current step count for that person. You need a header with the access token you got that matches the person whose user ID is in the GET command

Authorization: Bearer {access token}

This returns a long json string that you must parse in order to find the current steps. You can do this by searching for "steps", taking the string after that, looking for "steps" again and take what's after that, and search for ",", and takes the number before that and after "steps".



Definitions

user ID: a 6 letter/number code that identifies the user. You get this code in the response from the access code post and refresh token post (this user ID does not change).

STEP 3: Connecting to **Thingworx**

A beta version of a Thingworx library for LabVIEW just came out recently, and you should use that to make your device IoT enabled. Thingworx is a good place to store information that needs to be transferred between internet capable devices (such as a myRIO and computer in this project). For this project I used Thingworx to store the access codes and refresh tokens, because they changed so frequently, and the myRIO would lose them whenever you restarted the program.

Thingworx allows you to create “Things” that have “properties”. You can read/edit the value of properties using GET/POST commands (made easy by the Thingworx library). It would be good to try to become acquainted with Thingworx before going too deep into this project.

You will want to start a thing and make properties on it called “Current_Users” (list of the current users), “goal” (the office step count goal), “steps” (total number of steps). These are the basic properties that you want to have (more will be added later). You will also eventually want properties for each person’s individual access token and refresh token.

Definition

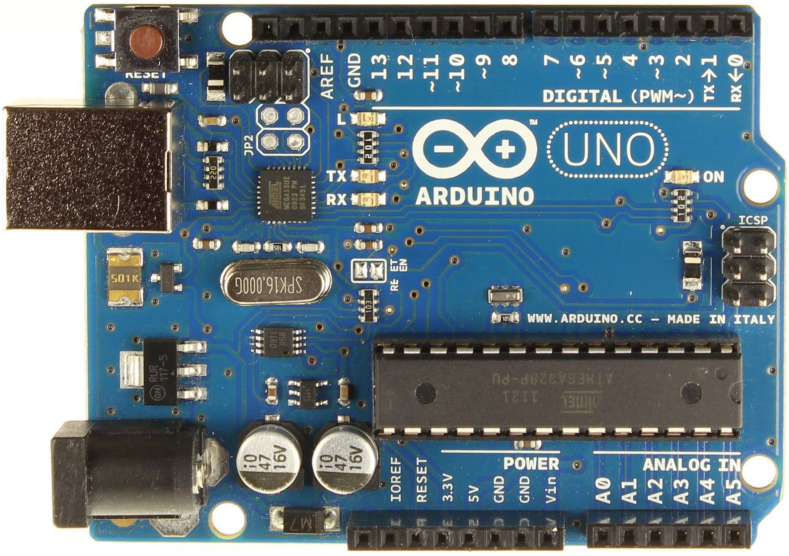
Thingworx: a cloud platform that stores information and can be accessed by internet capable devices.

STEP 4: Connecting an arduino to lights

For the cool light display, you will need to be able to control the lights and make them different colors. You can control the lights using an arduino and an external power source (7-9 volts).

First, you will need to download the Adafruit_Neopixel library. Open the strip test example.

Second you will need to hook up the LED strip to the arduino. You can do this using a breadboard. Connect the positive end of the external power source to one row on the breadboard and the negative end to a different row. You will need to connect the ground of the LED strip to the same row as the negative end of the external power source, and the positive wire of the LED strip to the row that has the positive end of the external power source. You will also need to ground the arduino with the same ground, so you will need a jumper that goes from a ground pin on the arduino to the row that has the negative of the external power source and the ground of the LED strip. Lastly connect the data wire of the LED strip to a PWM pin on the arduino (a pin that has a ~ before the number).



Make sure that the pin number variable in the code is the same as the pin number you plugged the LED strip data wire into. Then try running the code. Make sure you get this working and that the circuit setup is correct before moving on.

Once you have gotten that working, you can start writing your own code. Use the same structure as the example code. Since the LED strip that I used was 144 lights, I split the strip in 3rds, meaning that if the total step count was below a third of the whole goal, the lights would be red, if the total was between one to two thirds of the goal, the lights would be orange, and if the total was above two thirds of the goal, the lights would be green. As an indication that the total steps for the day was above the goal, all the lights would be purple.
Here is the code:

```
#include <Adafruit_NeoPixel.h>      //makes sure to include the neopixel library
#ifdef __AVR__
#include <avr/power.h>
```

```
#endif
#define PIN           3              //the pin number of the data wire
#define NUMPIXELS     144    //the number of neopixels on the strip

// defines what pixels and strip mean
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);

float goal = (type goal here);                    //goal
float steps = (type current steps here);          //current steps

void setup() {
  pixels.setBrightness(30);       //turns down brightness (the lights are very bright)
  strip.begin();                  // This initializes the NeoPixel lights (makes sure they are all off).
  strip.show();
}

void loop() {
    if(steps == 0){
      for(int p = 0; p <= 144; p++){
      pixels.setPixelColor(p, 0,0,0);   //resets the color of all the lights if the current total is zero
      pixels.show();
      }
    }

  if(goal > steps){
    for(int i = 1; i <= 144; i++){    // checks what "percentage" you are at
      if((float)i*(goal/144.0)> steps){
        k = i -1;                     //tells you how many light should light up
        break;
      }
    }

    if(k<48){
      for(int p = 0; p <= k; p++){
        pixels.setPixelColor(p, 255,0,0);   //set the number of lights, and if < 48 lights => red
        pixels.show();                      // displays lights
      }
    }

    if(k>48 && k<96){
      for(int p = 0; p <= k; p++){
        pixels.setPixelColor(p, 255,50,0);  //set the number of lights, and if < 96 lights => orange
```

```
      pixels.show();
    }
   }

   if(k>96){
    for(int p = 0; p <= k; p++){
      pixels.setPixelColor(p, 0,255,0);  //set the number of lights, and if >96 lights, => green
      pixels.show();
    }
   }
 }

 else{
    for(int n = 0; n <= 144; n++){
       pixels.setPixelColor(n,128,0,128);   // when over goal, lights will be purple
       pixels.show();
    }
  }
}
```

Note, pixel color is set like this:
pixels.setPixelColor(pixel number, red value, green value, blue value)

Change the values for goal and total and play around with it a bit to make sure that it works.

Now that you know that that works, you want to be able to send commands via your computer's serial port while the code is running on the arduino. This will eventually lead to the rio sending the serial commands to light up the lights. The serial port reads in characters, so you must concatenate them into a string, then parse the string into total and goal and then run the lights code based on that.
Here is the code:

```
#include <Adafruit_NeoPixel.h>         //makes sure to include the neopixel library
#ifdef __AVR__
#include <avr/power.h>
#endif
#define PIN            3               //the pin number of the data wire
#define NUMPIXELS      144    //the number of neopixels on the strip

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
```

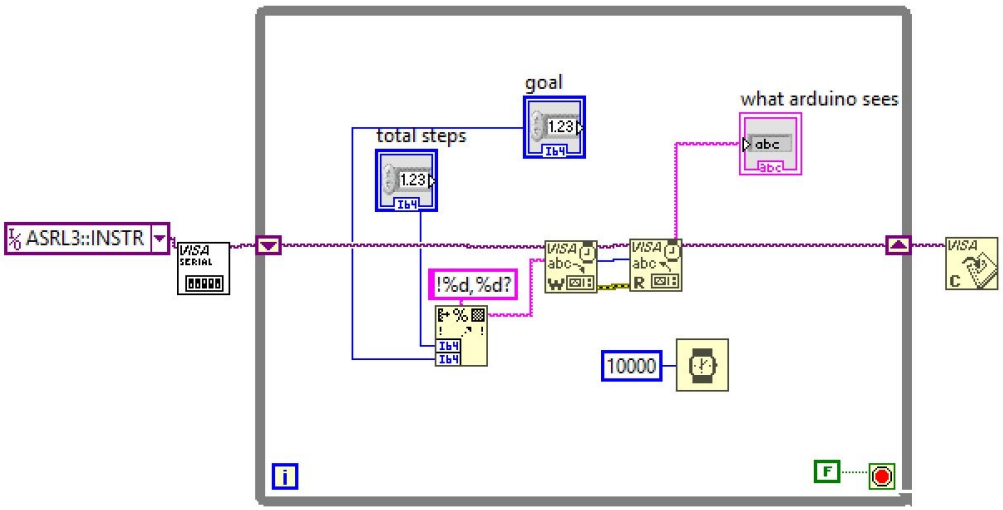
```
float goal; //goal
float steps; //current steps
int k;
bool reading = 0;
String data = "";
int g;
String stepString;
String goalString;

void setup() {
  pixels.setBrightness(30);       //turns down brightness (the lights are very bright)
  strip.begin();                  // This initializes the NeoPixel lights (makes sure they are all off).
  strip.show();

  Serial.begin(9600);             // opens serial port so the arduino can talk to rio

}

void stepCount(String input){     //function that parses “!#,#?” into numbers
  for(int s = 0; s <= input.length(); s++){
    if(input.charAt(s) == ','){
      g = s-1;
    }
  }

  stepString = input.substring(0,g+1);
  goalString = input.substring(g+2,input.length());
    steps = stepString.toInt();
    goal = goalString.toInt();
    steps = steps/10;
    goal = goal/10;
}

void loop() {
  // read  string in the form "! step number, goal number ?"
    if (Serial.available()) {                 //check to see if data is sent through serial
      char letter = Serial.read();
      if(reading == 0){
        if(letter == '!'){                    //if it wasn’t reading the characters and it sees ! start reading
           reading = 1;
        }
      }else{
        if(letter == '?'){                    //sees end of string
```

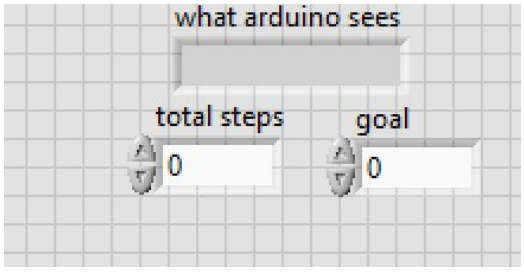
```
        reading = 0;            //stops making a string
        stepCount(data);        //parses string into goal and steps
        lights();               //runs lights (from before)
        Serial.print(steps);    //returns the step count (just so you can see that it works)
        data = "";              //resets the string to empty so it is ready for a new string
      }else{
        data = data + letter;           //makes the string using the characters coming in
      }
     }
    }
}

void lights() {
                Same code from before
  }
```

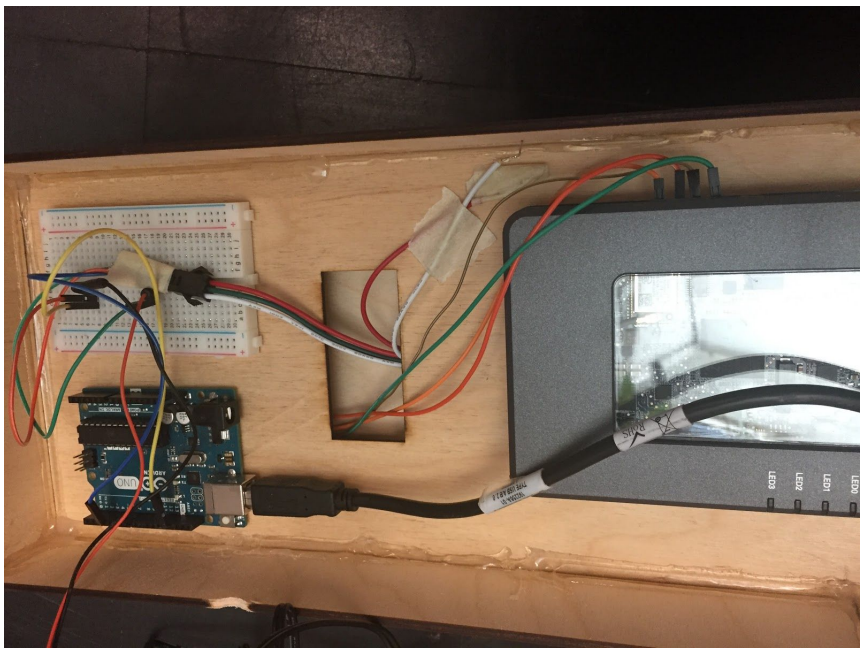
You should test this out by typing into the COMM command on the arduino application. Type in "!{steps},{goal}?" where {steps} is the total steps for the day and {goal} is your test goal. Make sure all this is working consistently before moving on.

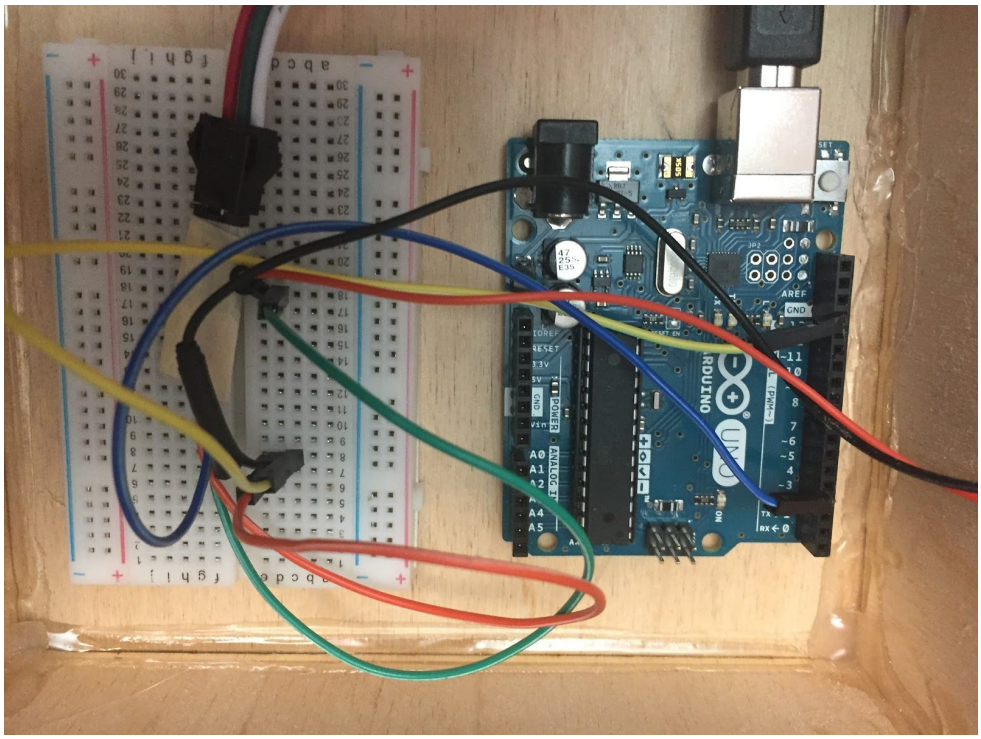
STEP 5: Connecting a myRIO to the arduino

Now, you need to write labVIEW code that will tell the myRIO to talk to the arduino and tell it to light up its lights. You want to have the myRIO open its serial port, start a while loop that is sending the arduino information and when you want it to stop, it closes the serial port. You will want to have controls on the front panel of labVIEW so that you can continuously change the goal and current step values and see the lights change (the same thing you were doing in the COMM command in the last step). Your code and front panel should look something like this



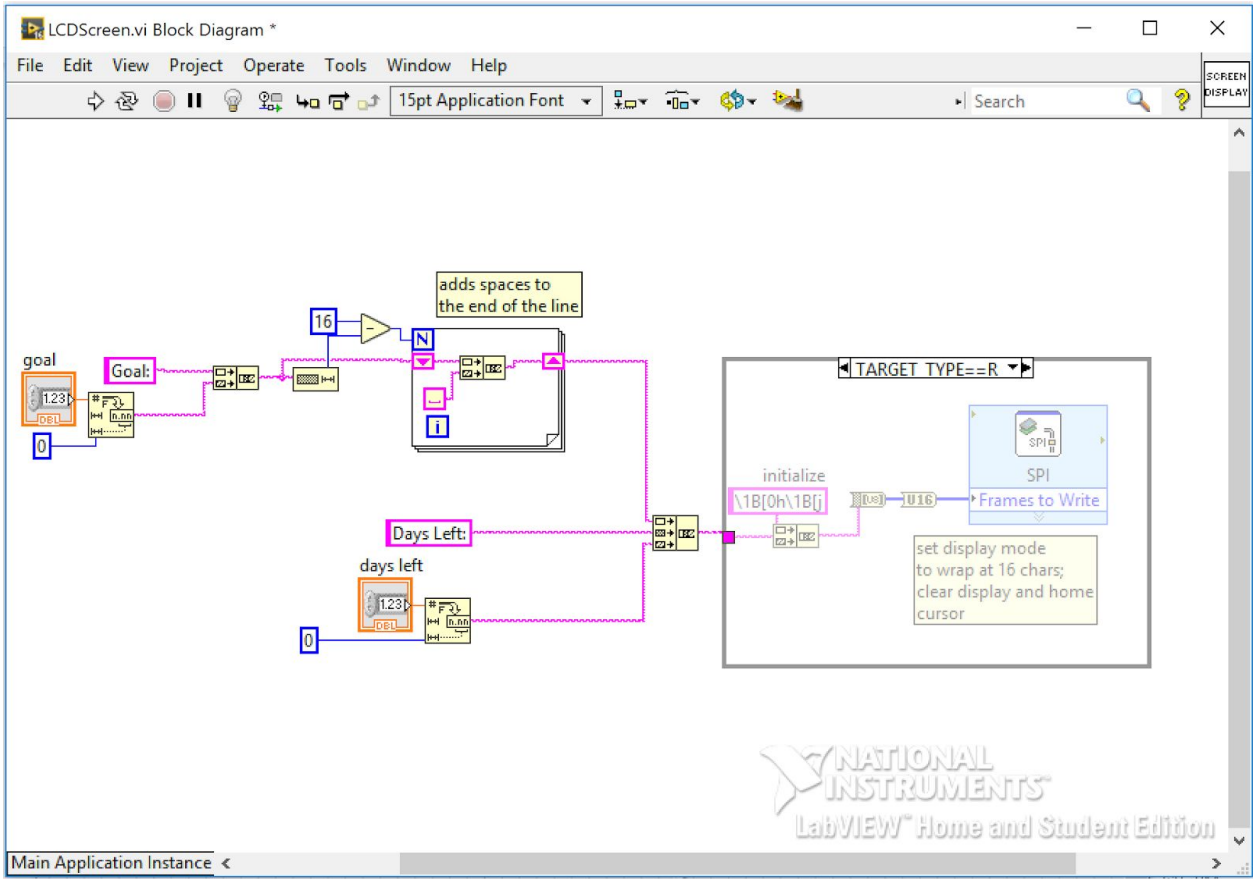

what arduino sees
total steps
goal
0
0

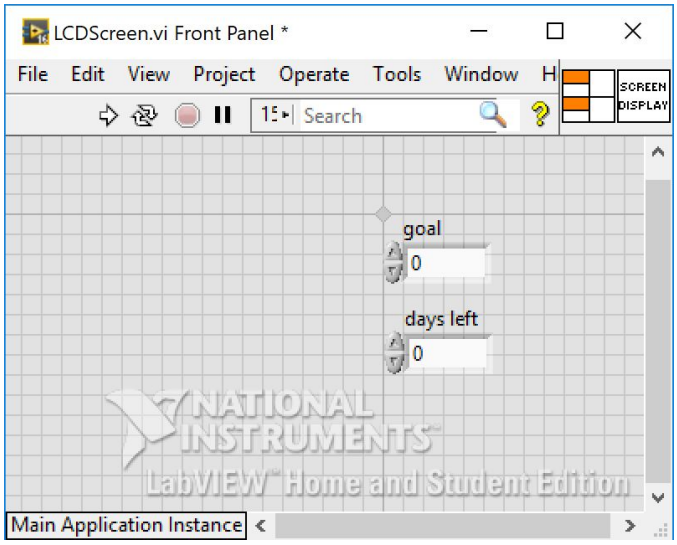




STEP 6: Adding an LCD Screen

Since the goal can change pretty frequently, it is nice to have a display that can show what the current goal is and how many days the office has left to complete that goal. This is where it is nice to have an LCD screen that just says “goal: # [enter] days left: #”. This code is pretty simple. You need to put a string into a function that displays it on the LCD screen. The string needs to be a concatenated string of “goal:”,”{the goal}”, “days left:”, and “{the number of days left}”. This is easy to do in labVIEW, because there is a string concatenation function in labVIEW. The one tricky part of this is making it so that “goal: #” and “days left: #” are on different lines. The way to do this is is to find the length of the string “goal:#” and then add 16 minus that length number of spaces, and then add “days left:” and “{number of days left}”. The who string is then fed into the function that displays the string on the LCD screen. Since you added the specific number of spaces, the length of the number for the goal doesn’t matter.





STEP 7: Putting it all together

It would be much easier to put all the steps together so that all you have to do is put in the person's name and the access code you get from the redirect url when they allow your application access to their account.

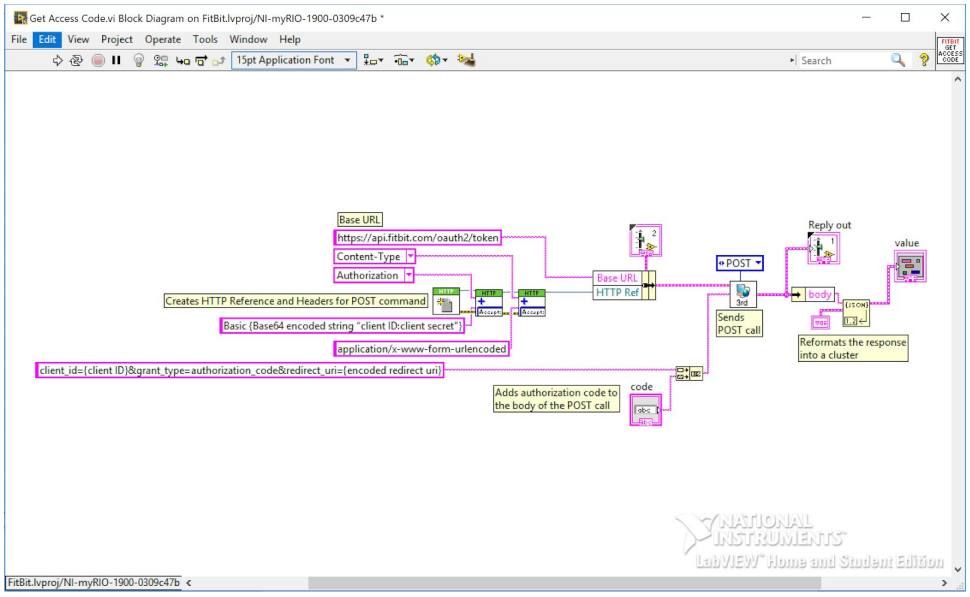
You will want to make many sub vis that do specific tasks. I will talk through the sub vis in order of when they would be used.

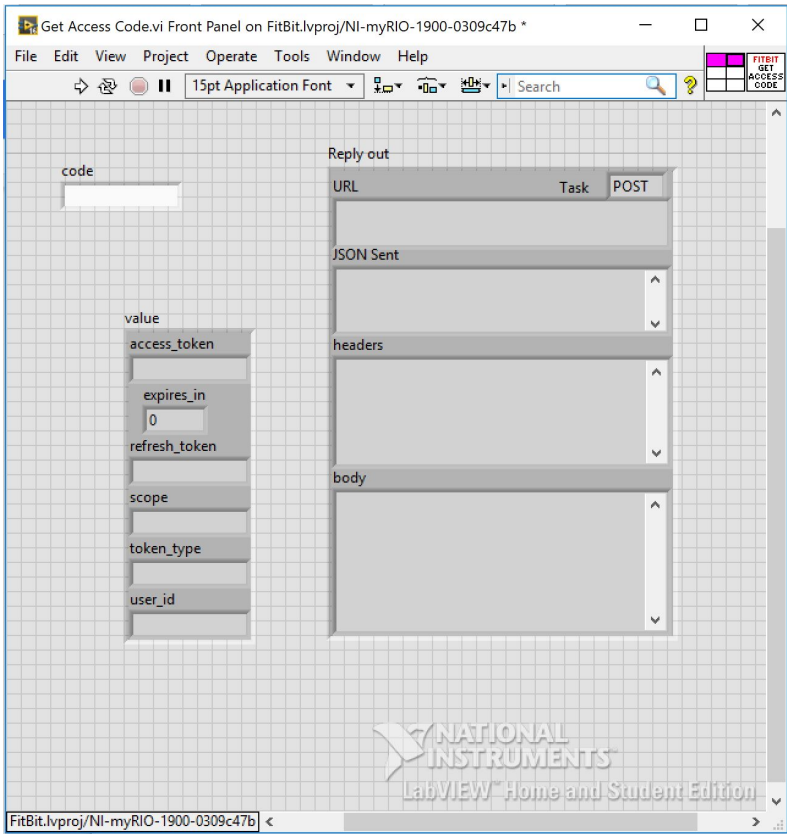
(1) Turning access code into access token

Inputs: access code

Outputs: access token, refresh token, user ID

This sub vi should take the access code, make the POST command to the fitbit api with the hard coded values of the headers, then parse the json response and output the access token, refresh token and user ID.



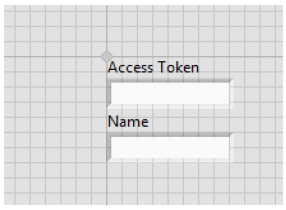


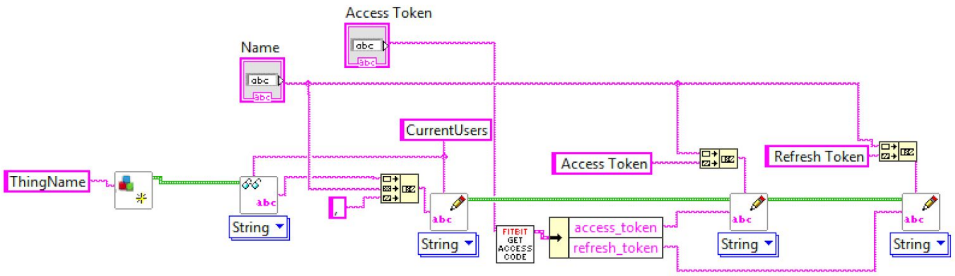
(2) Adding the new user to thingworx

Inputs: access code, person's name

This sub vi takes the access code, and puts it through (1). It also takes the person's name and creates two properties in Thingworx with it "PersonsName_AccessToken" and "PersonsName_Refresh_Token" and saves the access token and refresh tokens to it respectively. It also adds the person's name to the list of names in the property "CurrentUsers"

Code looks like this:



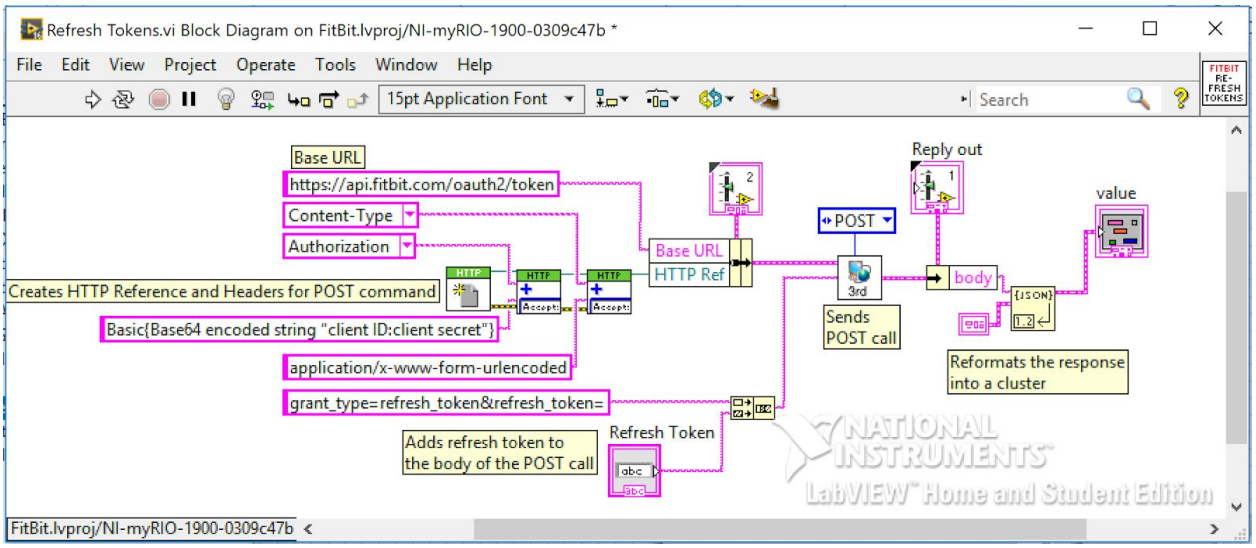


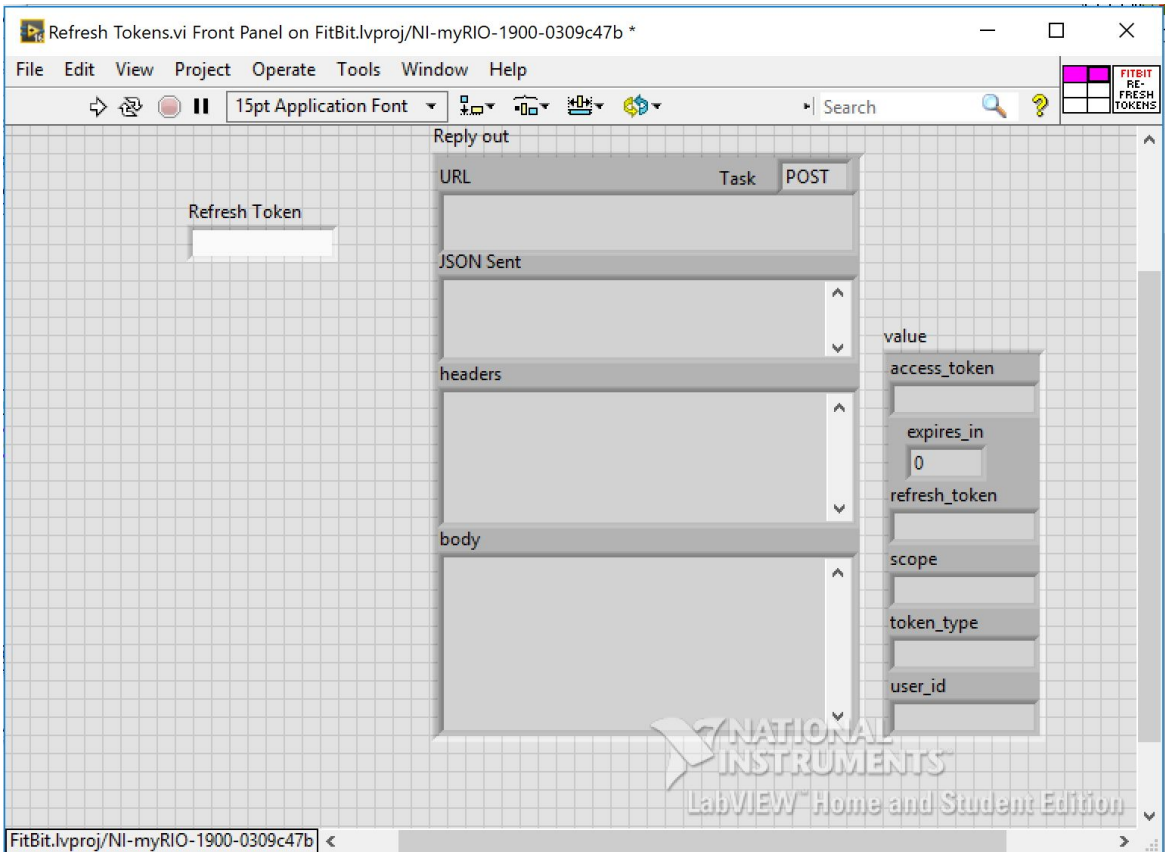
(3) Turning refresh token into access token

Inputs: refresh token

Outputs: access token, refresh token, user ID

This sub vi should take the refresh token, make the POST command to the fitbit api with the hard coded values of the headers, then parse the json response and output the access token, refresh token and user ID. (Very similar to (1), but making the POST to a different URL and with slightly different headers)



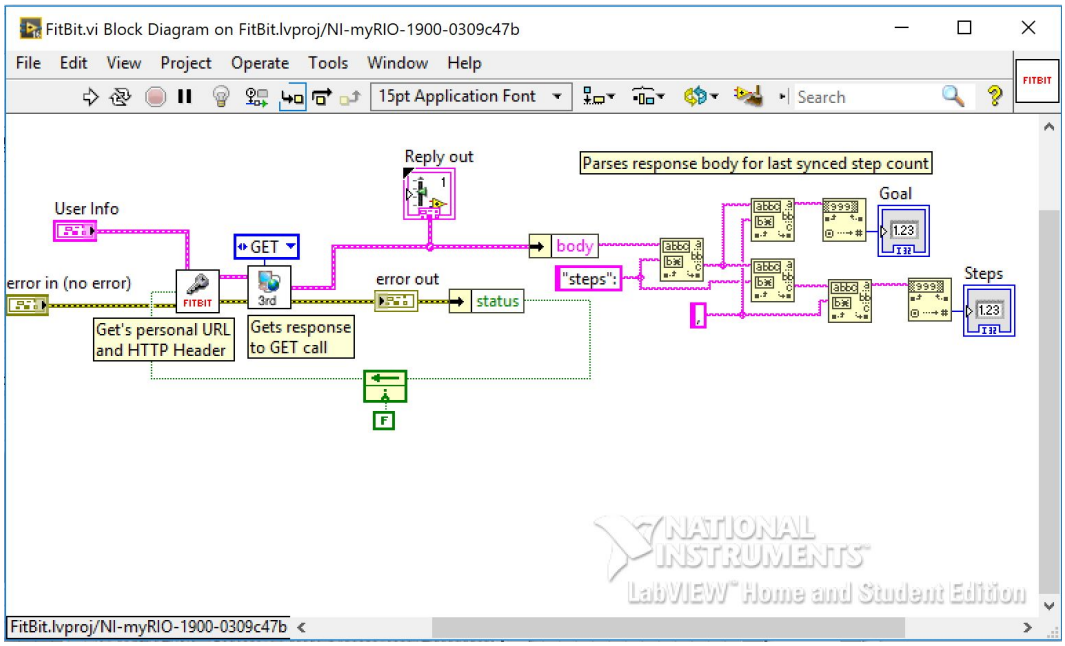


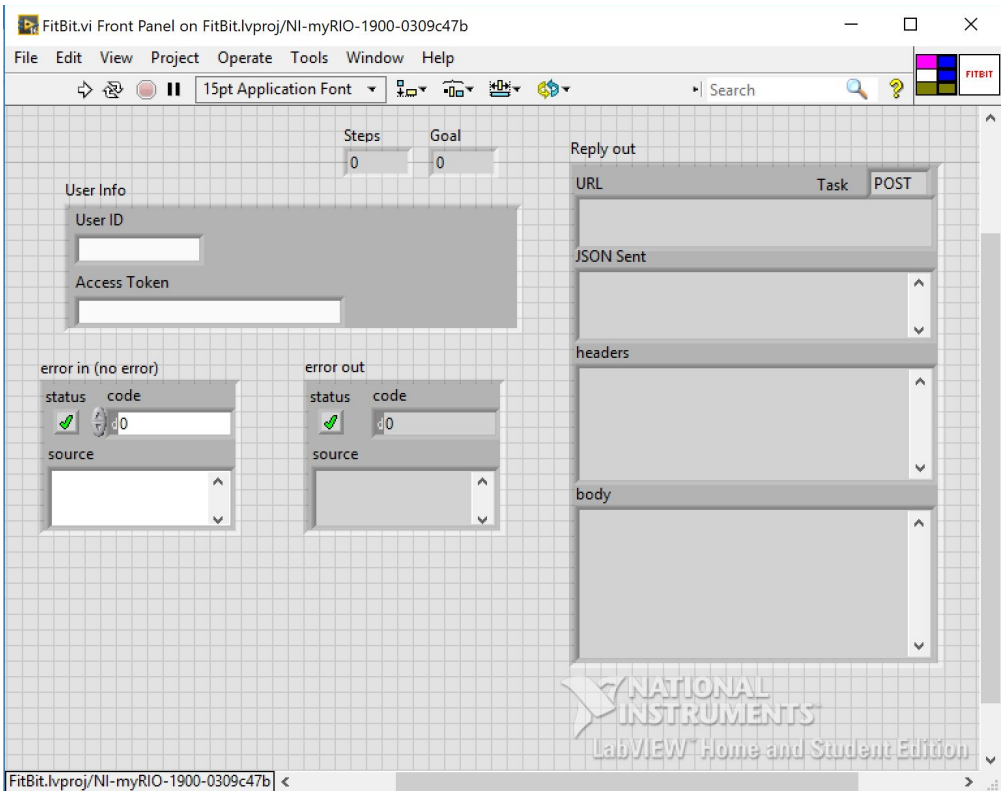
(4) Getting Steps From Fitbit

Inputs: access token, user ID

Outputs: step count

This one takes the user ID and adds it to the fitbit api base url that you use for making the GET command to. It also adds the access token to header. The code makes the GET command, parses the response and outputs the step count for that person.



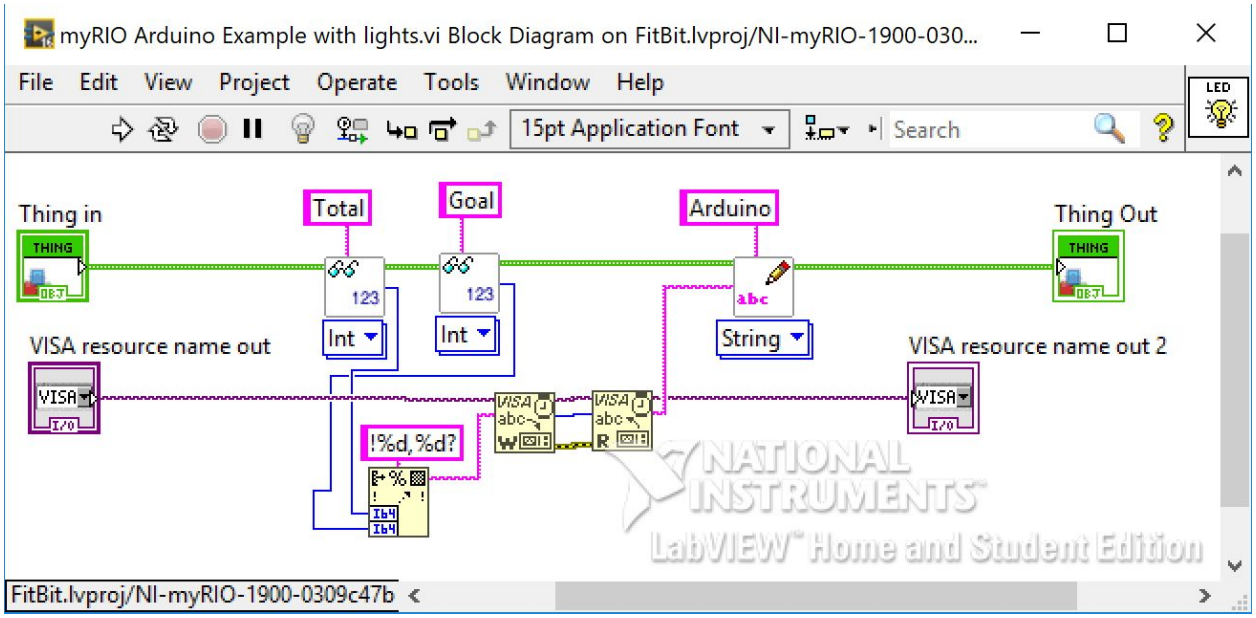


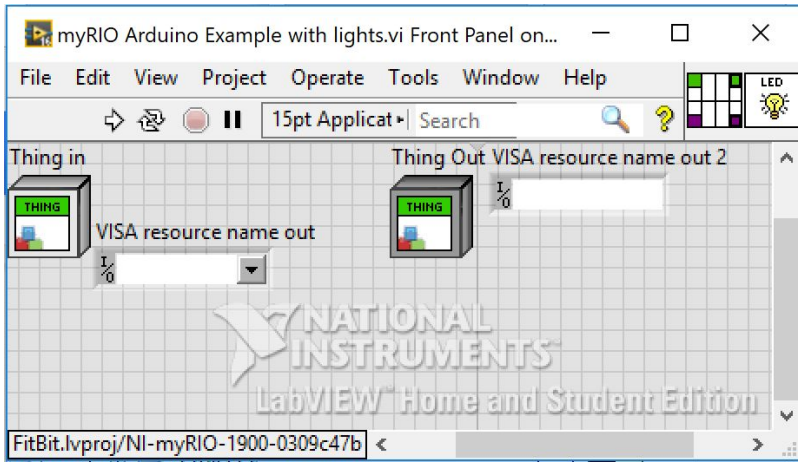
(5) Lights

Inputs: serial port, steps, goal

Outputs: serial port

This sub vi is the vi that we wrote in a previous step where you can input the steps and goal and it has the RIO tell the arduino the numbers and the arduino lights up the lights. It outputs the serial port, so that you can have it in a chain with other vis that may need to open serial connections.





(6) Screen Display

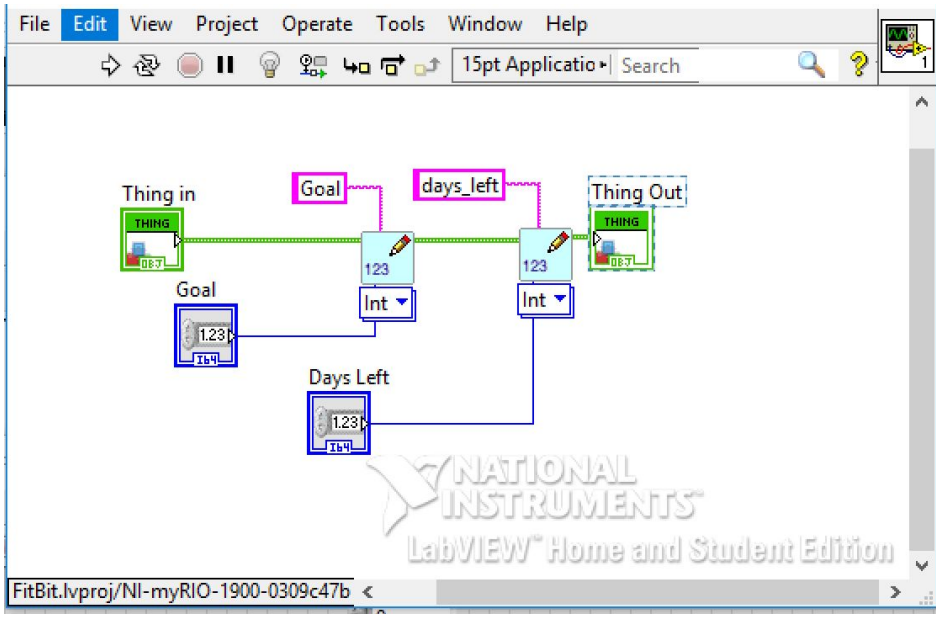
Inputs: days left, goal

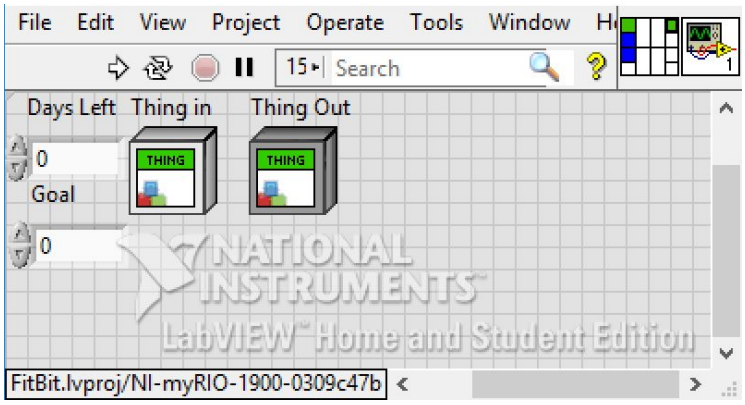
This sub vi makes an LCD screen display the goal and the number of days left to complete the goal.

(7) Change Goal

This is a vi that runs on its own. You add the goal that you want and the number of days that the office has to complete that goal to controls on the front panel. It takes this information,

and writes the new goal and days left to complete to thingworx, overriding the previous values. It also clears the total, so that it restarts the light display and count.

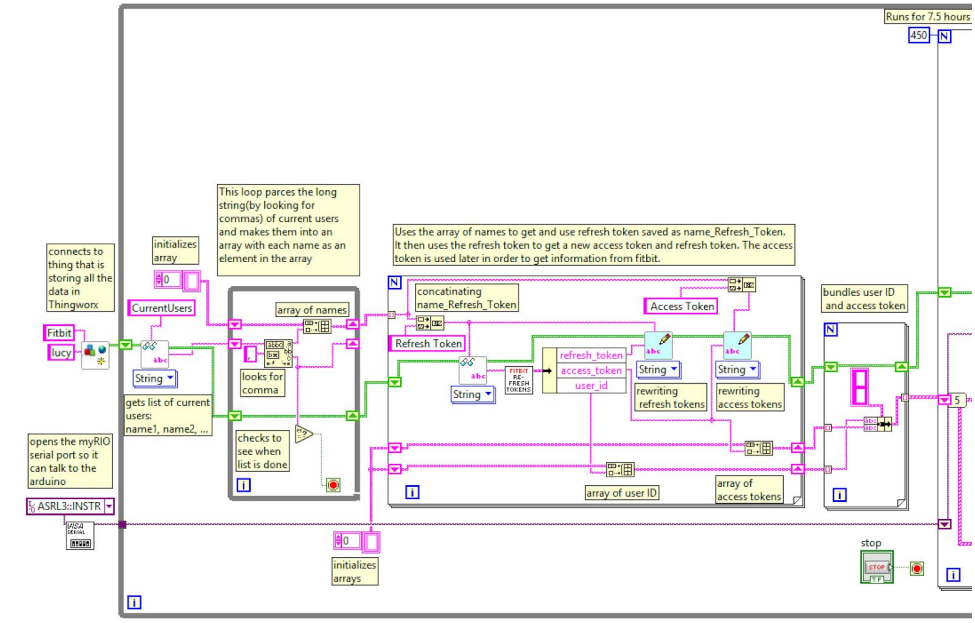


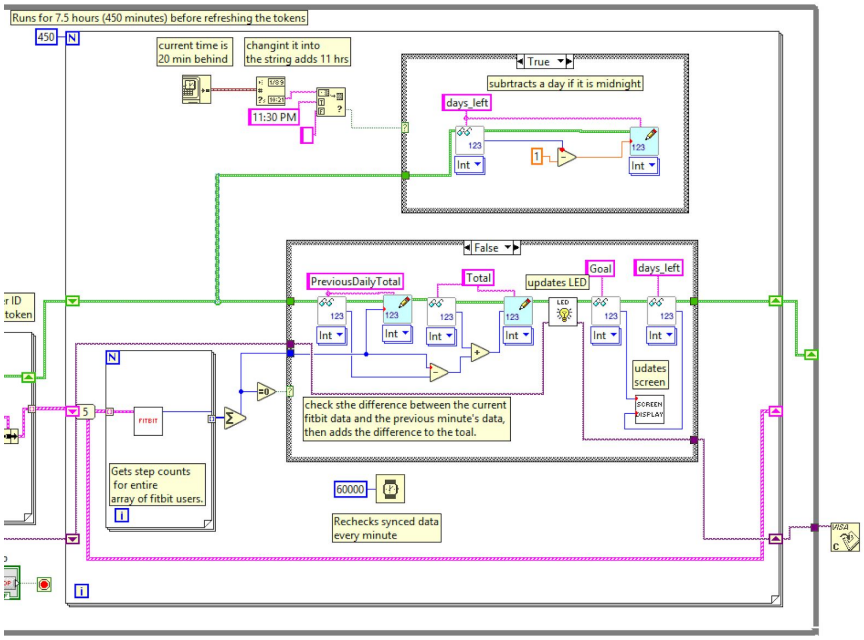


Once all the sub vis are made, you can start adding them into one big vi.Outside the while loop you want to start a thing and open the serial port. Then you will want to run a while loop that runs forever. In that loop you want to grab the list of names from the “CurrentUsers” property, parse through them and make an array of the names of people. Then send that through a for loop in order to read the “name_Refresh_Token” and send it through the refresh token sub vi and then overwrite the “name_Access_Token” and “name_Refresh_Token”. As well as creating an array of clusters which contain the access token and user ID. After going through the for loop, it goes into another for loop which runs once every minute for 450 minutes (7.5 hours). That way it uses the access token for 7.5 hours and then goes back to the beginning of the while loop and uses the refresh token and gets a new access code and keeps going.

Inside the for loop that runs for 7.5 hours, you want to have a for loop that goes through the array of clusters and sends each group of user ID and access token through the step count vi to get the steps. Then sum all the steps to get the total. Update the total in thingworx. Send the total into the lights vi. Also grab the goal from the “goal” property in thingworx and send that

into the lights vi. Also send the goal to the screen display. Pull the “days_left” property value and send that into the screen display vi. Make sure to wire the array of clusters to the end of the for loop and change it into a shift register so that the next time it loops, it can send the access token and refresh token to the step vi again, and continue the cycle.





STEP 8: Making the box

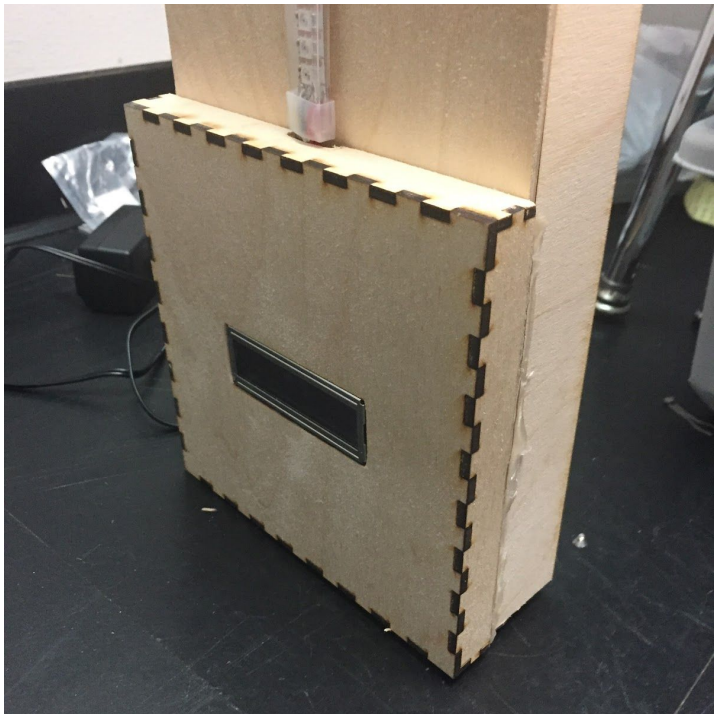
Make sure that all the code works before making a polished box for it. I made the box by taping together 2 ⅛ thick pieces of 24x6 in wood. (6 in sides together to make a piece 48x6 in).

If you have a laser engraver you can write the title of your step counter at the top along with motivational saying to the side of where the LED strip would be. You and also just do this in Sharpie. Place the LED strip on the wood with the wires that connect to the breadboard at the bottom. Cut a hole at the bottom that allows the wires to go through it so that you can have the rio, arduino and breadboard on the back of the wood. Also put the LCD wires through that hole and have the LCD screen on the front of the wood. Cut 2 12x2 in pieces and 4 6x2 in pieces and 2 5.75x2 in pieces. Lining the 6x12 in pieces with the corner of the back of the 48x6 in piece where the 6 in part is along the edge of the 48 in part, hot glue the piece down to create a 2 in wall. This wall will hold the 48x6 in wood away from the wall allowing the RIO, arduino and breadboard to be behind it without people see it, and without it resting on those pieces of hardware. Next to the 6x2 in piece, hot glue the 12x2 in piece (again lined up with the edge of the 6x48 piece). This creates more stability where the 2 6x24 in pieces meet. Then glue another 6x2 in piece next to that to create an entire side. Do this again on the other side. Now, the 5.75x2 in pieces should fit snuggly between those two side walls at the top and bottom. Glue those in. Add reinforcement where needed. Now you should have an open top box. Feel free to tape the RIO, arduino and breadboard inside the box.

Flip the box back over so that you are looking at the front. Tape down the LED strip where you want it (double sided tape works well).

Make a open box (5 faces missing the second 6x6 face)) whose outside dimensions are 6x6x1 in. This will hold the LDC screen. Cut a 3x1 in hole out from the middle of the 6x6 face. The LCD should fit right in. You may need to solder the back of the LCD screen and cut the prongs on the front to make it fit snuggly. Now glue that box to the front of the 48x6 in piece of wood, under the LED strip. Try to cover the hole you made for the wires to go through with the box.

Add screw eyes to the top, so that you can put string between the screw eyes and hang it from the wall. See pictures.




So close
Keep
Going


So close
Almost
there
Keep
Going
Come
On
Halfway
there!
Getting
there
You got
this
Let's
go!
Get
Walking
Ready,
Set,
GO!

STEP 9: Using it

You email the link (https://www.fitbit.com/oauth2/authorize?response_type=code&client_id={client ID}&redirect_uri={redirect URI}&scope=activity) to the person who wants to add their fitbit data to the display. They need to email you back the redirect page's url. You then take the code from that, put it in the front panel of the "adding new user to thingworx" vi along with their name, and click run. This will add the person to the list of people and when the main code that is running on the rio loops back to the beginning and checks all the people, the new person will be added. Note: you can speed this up by unplugging the rio from the wall, waiting 10 seconds and plugging it back in. The rio take a bit of time to reboot, so be patient.

To change the goal, go to the Changing Goal vi and put in the new values. Then run the code. The old values will be overwritten.