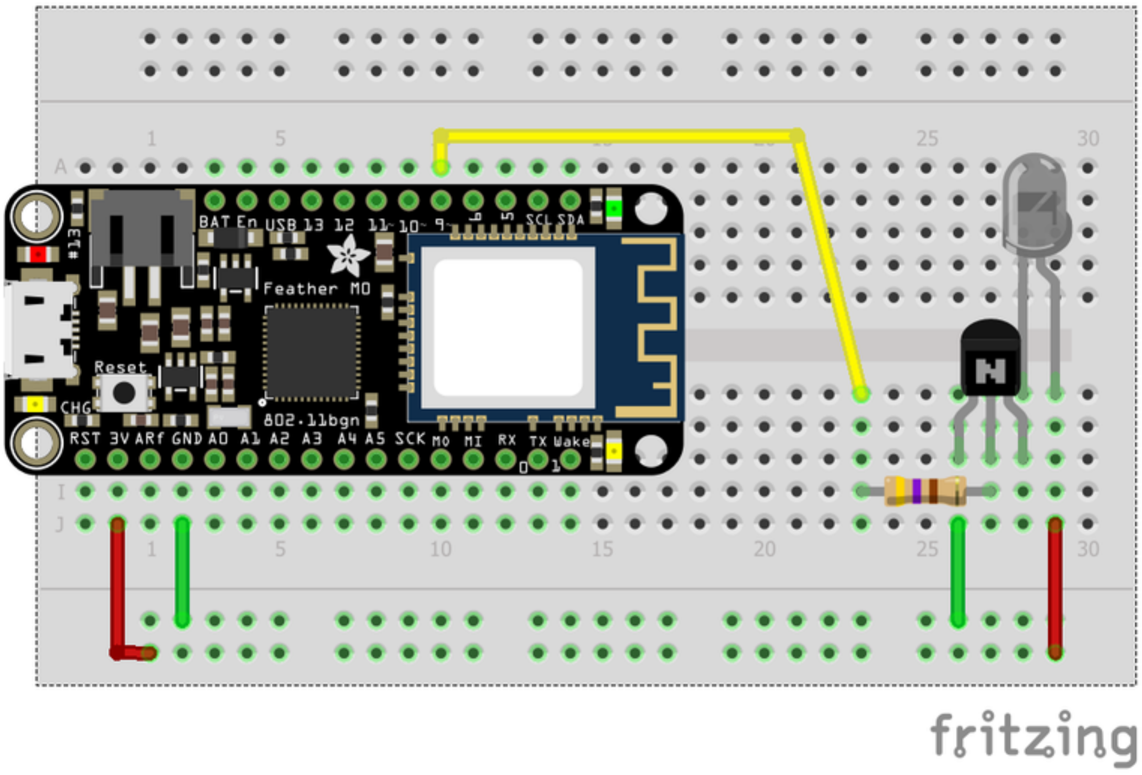
Internet of Things Infrared Remote

Created by Chris Young



BAT En USB 13 12 11 10 9 6 5 SCL SDA
Feather M0
802.11bgn
Reset
CHG
RST 3V ARf GND A0 A1 A2 A3 A4 A5 SCK MO MI RX TX Wake
fritzing

https://learn.adafruit.com/internet-of-things-infrared-remote

Last updated on 2023-08-29 03:41:08 PM EDT

Table of Contents

Overview

In this project we will show you how to create an infrared remote control your TV, cable box, DVD or other IR controlled device over Wi-Fi.

You control the device via a webpage that you can store on your PC and open directly or if you have web hosting capability it can be viewed by a tablet or phone. You layout the buttons and assign the functions however you want. Any function which can be transmitted by the protocols in the infrared library IRLib2 can be transmitted.

This project was designed for the Adafruit Feather M0 Wi-Fi board but also works with the Arduino MKR1000 both of which use the Wifi100 library and the ATWINC1000 Wi-Fi device.

Unfortunately at this time IRLib2 does not support ESP 8266 but if it eventually implements that capability, we will update this tutorial for use with that board.

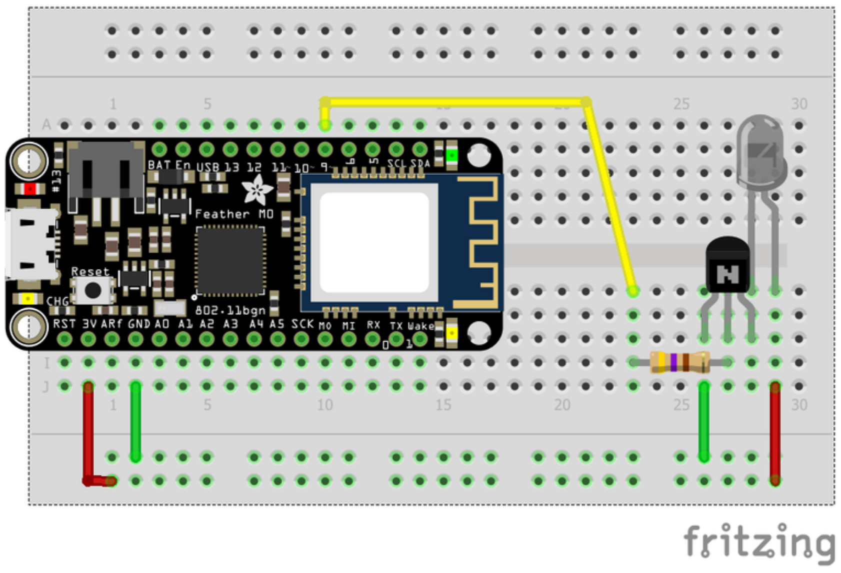
In addition to the board you will need an IR LED, an NPN transistor, and a 470 ohm resistor to create a simple LED driver circuit.

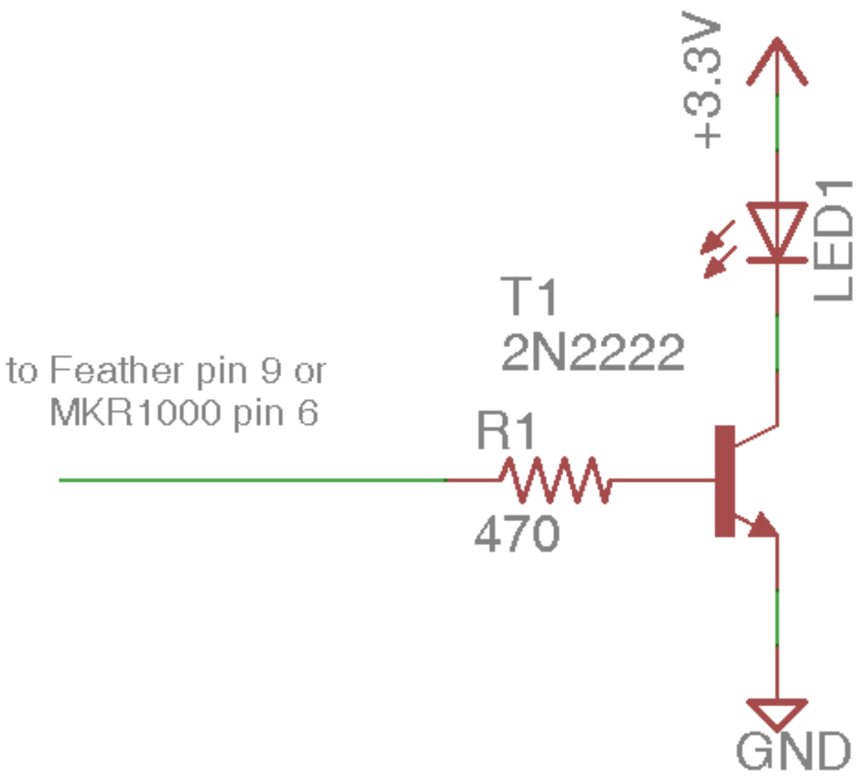
All of the necessary parts are available in the Adafruit store via links on the right.

Wiring

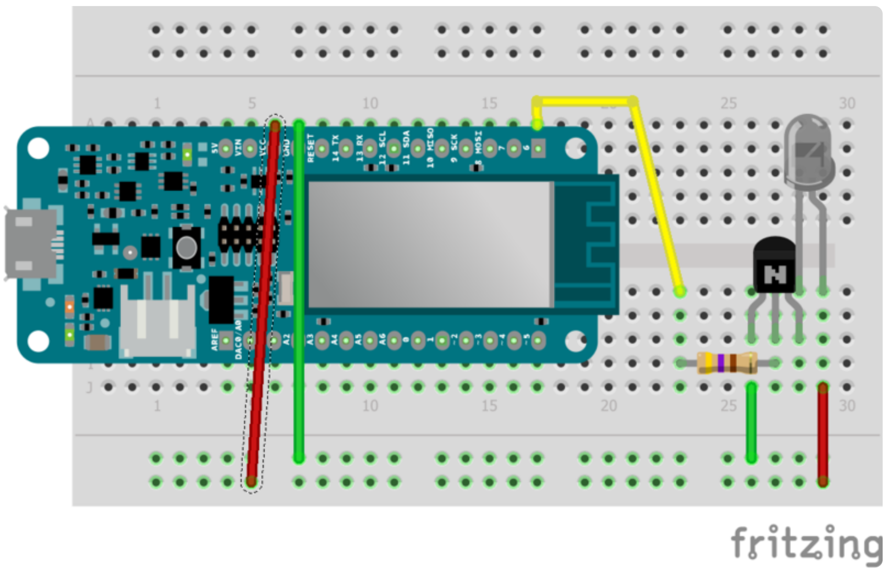
Typically the output of an Arduino type board does not have sufficient current to drive an LED so you need a driver circuit using a transistor. Here is a fritzing diagram and a schematic of how to wire up a simple driver circuit. A more complicated multiple LED driver circuit can be found in the documentation for IRLib2.

The base of the transistor is connected to pin 9 of the feather board (or pin 6 of the MKR 1000)using a 470 ohm resistor. The emitter is connected to ground and the LED is connected between the +3.3V and the collector.





The MKR1000 uses pin 6 as its default output pin rather than 9 on the Feather. These defaults can be changed in IRLib2. See it's users manual for details.



Installing Libraries

Whether using the Feather M0 Wi-Fi or the MKR1000 boards you will need to install the official Arduino WiFi101 library. Complete details on how to do this can be found in the tutorial for the Feather M0 Wi-Fi found here (). We recommend that you go through that tutorial and familiarize yourself with the Wi-Fi examples found there. Make sure that your firmware has been updated to the most recent version. Note: If you're running 19.5.2 there is no need to update to 19.5.4. Despite what the WiFi101 library says it should work just fine.

You will also need to install IRLib2 from github.

Installation of the IRLib2 library is as follows:

1. Visit the IRLib2 page on GitHib ().
2. Select the “Download ZIP” button, or simply click this link () to download directly.
3. Uncompress the ZIP file after it’s finished downloading.
4. The resulting folder should be named "IRLib2-master" and will contain 5 separate folders. That is because IRLib 2.x is actually a collection of 5 libraries that work together. Sometimes in Windows you’ll get an intermediate-level folder and need to move things around.
5. Copy all five folders into your Arduino library folder along side your other Arduino libraries, typically in your (home folder)/Documents/Arduino/Libraries folder. Libraries should not be installed alongside the Arduino application itself.
6. Re-start the Arduino IDE if it’s currently running.

This repository consists of a total of five libraries each of which must be in your arduino/libraries/ folder. So for example it should be installed as follows...

- arduino/libraries/IRLib2
- arduino/libraries/IRLibFreq
- arduino/libraries/IRLibProtocols
- arduino/libraries/IRLibRecv
- arduino/libraries/IRLibRecvPCI

Do not install them in a single folder such as this...

- arduino/libraries/IRLib2_master
  - IRLib2
  - IRLibFreq
  - IRLibProtocols
  - IRLibRecv
  - IRLibRecvPCI

Here's a tutorial () that walks through the process of correctly installing Arduino libraries.

More details on using IRLib2 can be found in my tutorial "Using an Infrared Library on Arduino ()".

Software Setup

The software for this project consists of 2 parts. There is an Arduino sketch that you upload to the device. There is also a webpage that you can either open on a PC to view in a browser or if you have a web hosting service available or a Web server on your PC or perhaps a raspberry pi you can host the webpage that way.

The necessary files can be found in the IRLib2/examples/IoT_IR folder. Or you can copy and paste from the source code listings below.

Configuring the sketch

The Arduino sketch consists of 3 files.

Here is IoT_IR.ino which is the source code in C++,

```
/*
 * IR IOT demo program
 * by Chris Young
 * use with Adafruit M0 Wi-Fi
 */

#include &lt;SPI.h&gt;
#include &lt;WiFi101.h&gt;
#include "WiFi101_Util.h"

#include &lt;IRLibSendBase.h&gt;            // First include the send base
#include &lt;IRLib_P01_NEC.h&gt;            // Now include only the protocols you
wish
#include &lt;IRLib_P02_Sony.h&gt;           // to actually use. The lowest numbered
#include &lt;IRLib_P05_Panasonic_Old.h&gt;  // must be first but others can be any
order.
#include &lt;IRLib_P07_NECx.h&gt;
#include &lt;IRLibCombo.h&gt;               // After all protocols, include this

IRsend My_Sender;

void processIR(WiFiClient client) {
  int protocol = client.parseInt();
  if (client.read() != '/') return; //Need more. If not there then ignore
  unsigned long code= client.parseInt();
  //If next character is a '/' then we will parse number of bits
  //otherwise assume bits are zero
  int bits =0;
  if (client.read() == '/') {
    bits = client.parseInt();
  }
  client.print("{\"command\":\"irsend\"");
  client.print(",\"protocol\":"); client.print(protocol);
  client.print(",\"code\":");  client.print(code);
  client.print(",\"bits\":");  client.print(bits);
  client.println('}');
  My_Sender.send(protocol, code, bits);
}

void setup() {
  WiFi101_Setup(); //moved all the setup code to a separate tab for clarity
}

void loop() {
  // listen for incoming clients
  WiFiClient client = server.available();
  if(client.available()) {
    //char c = client.read();//skip initial "/"
    #if(0) //use for debugging
      String command=client.readString ();
      Serial.print('"');Serial.print(command);Serial.print('"');
    #else
      String command= client.readStringUntil('/');//Skips over "Put" to the start
of the commands
      command= client.readStringUntil('/');
      if(command == "irsend") { //is this IR send command
        processIR(client);
      } //add an else if there are other kinds of commands
    #endif
    client.stop();
  }
  delay(50);//Poll every 50ms
}
```

WiFi101_Util.h is an include file containing all of the Wi-Fi setup code. Make sure that SERIAL_DEBUG is set to 1 so that you can ensure that your board is working properly and you can obtain your IP address. Once you have everything configured and are using the board under normal circumstances you would set this value to 0.

```
/*
 * Moved all of the Wi-Fi initialization and debugging code here for clarity
 */
#include "arduino_secrets.h"
///////please enter your sensitive data in the Secret tab/arduino_secrets.h
char ssid[] = SECRET_SSID; // your network SSID (name)
char pass[] = SECRET_PASS; // your network password (use for WPA, or use as key for
WEP)
int keyIndex = 0;          // your network key Index number (needed only for WEP)

//Set to 1 if you are going to open the serial monitor
#define SERIAL_DEBUG 1
IPAddress ip;
int status = WL_IDLE_STATUS;
// Initialize the WiFi server library
WiFiServer server(80);

#if SERIAL_DEBUG
  void printWifiStatus() {
    // print the SSID of the network you're attached to:
    Serial.print("SSID: ");
    Serial.println(WiFi.SSID());

    // print your WiFi shield's IP address:
    Serial.print("IP Address: ");
    Serial.println(ip);

    // print the received signal strength:
    long rssi = WiFi.RSSI();
    Serial.print("signal strength (RSSI):");
    Serial.print(rssi);
    Serial.println(" dBm");
  }
#endif

void WiFi101_Setup(void) {
  #if defined(ARDUINO_SAMD_FEATHER_M0)
    //Configure pins for Adafruit ATWINC1500 Feather
    WiFi.setPins(8,7,4,2);
  #endif

  #if SERIAL_DEBUG
    // Start Serial
    Serial.begin(115200);

    // check for the presence of the shield:
    if (WiFi.status() == WL_NO_SHIELD) {
      Serial.println("WiFi shield not present");
      // don't continue:
      while (true);
    }

    String fv = WiFi.firmwareVersion();
    if ( fv != "1.1.0" )
      Serial.println("Please upgrade the firmware");
  #endif
  // Attempt to connect to Wifi network:
  while ( status != WL_CONNECTED) {
    #if SERIAL_DEBUG
        Serial.print("Attempting to connect to SSID: ");
        Serial.println(ssid);
    #endif
    // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
    status = WiFi.begin(ssid, pass);
    // Wait 10 seconds for connection
    delay(10000);
  }
  // Start the server
  server.begin();
  ip = WiFi.localIP();
  #if SERIAL_DEBUG
    // Print out the status
    printWifiStatus();
  #endif
}
```

The file arduino_secrets.h where you will enter your Wi-Fi SSID and password. You will have to edit this file to include your values.

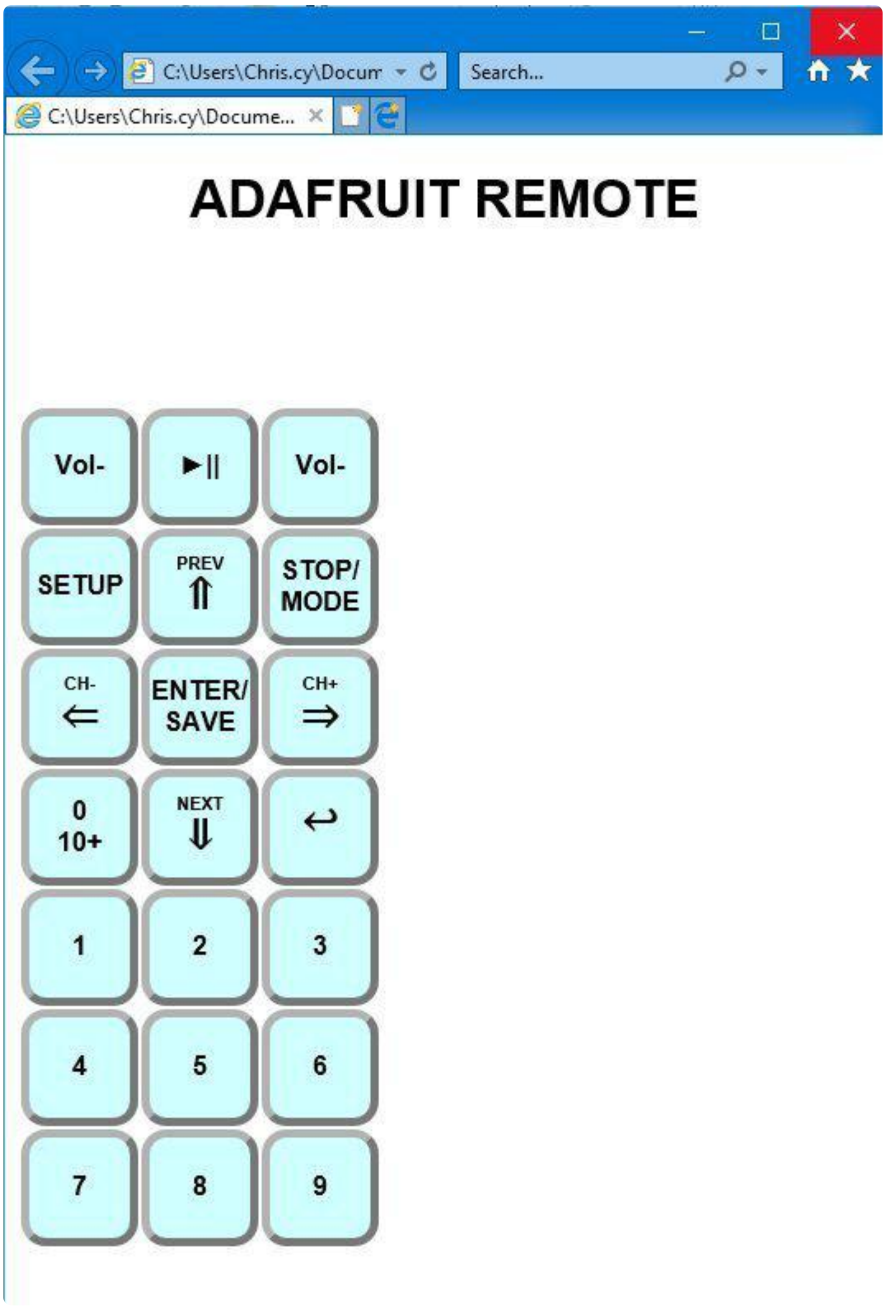
```
#define SECRET_SSID ""
#define SECRET_PASS ""
```

Upload the sketch and open your serial monitor. Take note of the IP address that is printed. You'll need that information later.

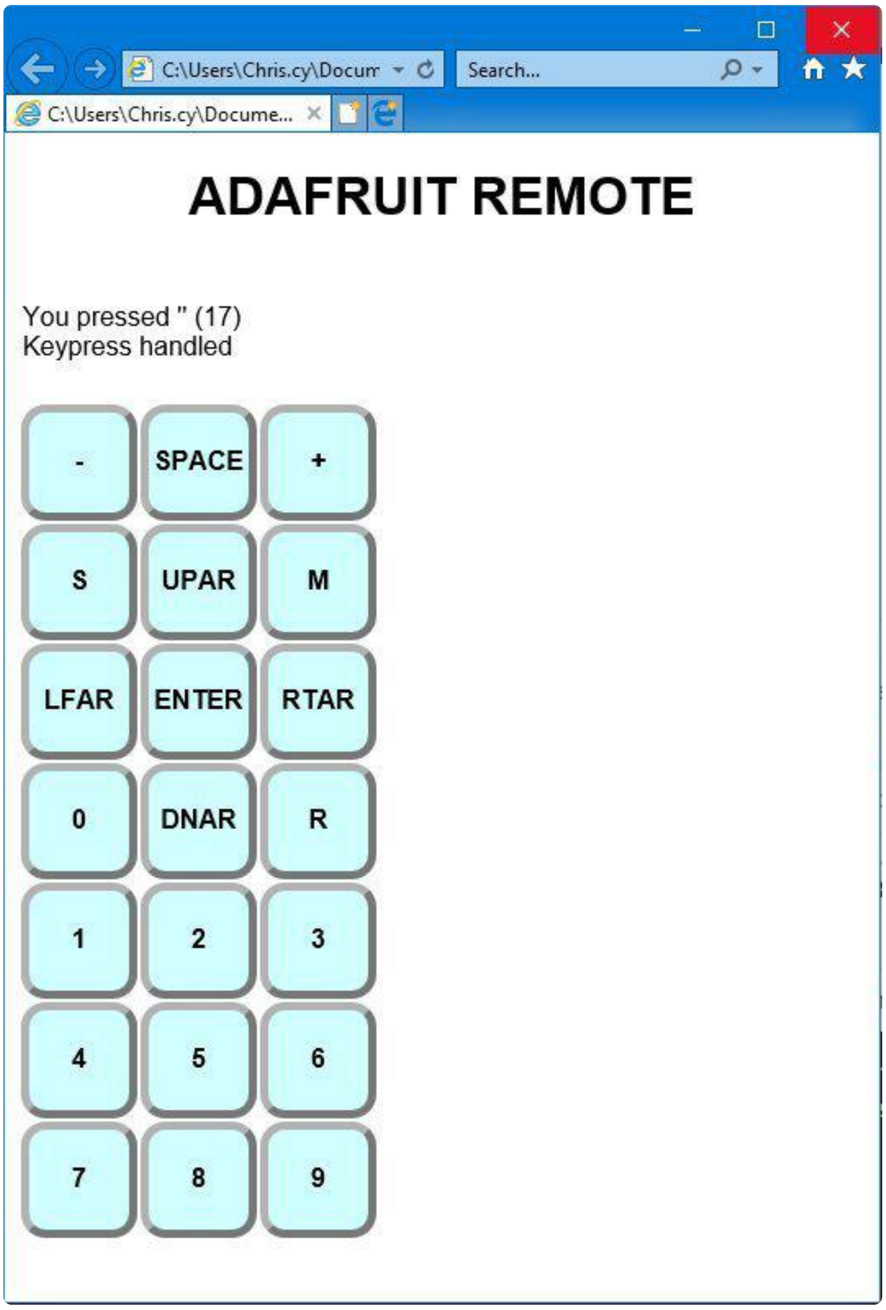
Configuring the HTML file

Open the file adafruit_remote.html using a text editor. At approximately line 21 edit in the URL value that you got from serial monitor. Then save the file and open it in any browser. If you use Internet Explorer it will ask you to enable active X controls. Other browsers we have tested such as Google Chrome, Mozilla Firefox, and Opera browser do not have this requirement. Also Internet Explorer will automatically resize the window and dock it in the upper left corner of your screen. The other browsers do not allow this to be down under JavaScript control.

You will see a screen that looks like this.



This remote has been preconfigured to emulate the Adafruit Mini-Remote product number 389 (). When you click on any of the buttons it will transmit the proper code via your IR LED. If you have another Arduino set up with a receiver you can detect the transmitting codes. In addition to mouse clicks, this webpage also responds to keyboard presses. If you want to see which keystroke activates which button, You should press the "escape" key on your keyboard and the display will toggle to reveal this...



Customizing your remote

It is unlikely that you have a device that responds to these codes unless you have built one yourself designed to use this remote. So you will want to customize the webpage to create your own custom remote . Open the file in a text editor and and save as a different filename. Then we will edit the button definitions which begin at line number 26. This is a JavaScript definition that defines the buttons. Here is a portion of the existing code.

```
var Button= [//object containing all buttons
    [//object containing row 0
        [1,0xfd00ff,0, "Vol-",189],
        [1,0xfd807f,0, "►||",32],//play pause
        [1,0xfd40bf,0, "Vol-",187]
    ],
    [//row 1
        [1,0xfd20df,0, "SETUP",83],
        [1,0xfda05f,0, "&lt;span class='Tiny'&gt;PREV&lt;/span&gt;&lt;br /
&gt;&lt;span class='Big'&gt;⇑&lt;/span&gt;",38],//up arrow
        [1,0xfd609f,0, "STOP/&lt;br /&gt;MODE",77]
    ],
```

Although we have only included 3 buttons per row, you can have as many buttons in each row as you want and will fit on your webpage. Each entry of the button consists of five values. The first value is the protocol number used by IRLib2. In this example we are using protocol 1 which is NEC protocol. The second number is a hex value that is the actual data transmitted. The third value is the number of bits. Note that some of protocols do not require this value so you can enter zero as we have in this example. The third value is a string enclosed in quotes that will be displayed. Note that this can include HTML formatting such as linebreaks or span definitions to change the style such as size and color. The fifth value is the keyboard scan code for the keypress that you want to assign to that particular button. If you do not know the proper value you can open up the webpage and press a key and it will display the value for you.

To determine the proper protocol number and code for your particular remote you will need to set up an Arduino with a receiver and use the dump.ino sample sketch from IRLib2. Point your remote at the receiver and record the protocol number, the data value, and the number of bits. For further details on how to determine the proper protocols and data values for your remote, we recommend you read our other tutorial "Using an Infrared Library on Arduino ()"

For further examples of how you might configure your buttons see our other example file "cable_and_tv.html" which has been configured for my own cable box which is a Scientific-Atlanta/Cisco cable box from BrightHouse/Spectrum cable company which uses protocol number 5 Panasonic_Old and a Samsung TV which uses protocol 7 NECx.

NOTE: This sketch will only work if the device is connected to your computer with the serial monitor open. Once you have everything working, you can change the SERIAL_DEBUG to 0 in the WiFi101_Util.h file. Then the device will no longer be dependent upon serial output. It can be disconnected from your computer and operated by battery or USB power supply.

How it works

Everything you need to build and configure this project has already been explained in this tutorial so you can stop now if you want and enjoy the project. But if you want to dig deeper and understand how the code really works, in the remaining sections of this guide we will explain a little bit of the programming principles behind the project.

This is not intended to be a complete tutorial on all of the programming practices implemented in this project. It is just designed to give you a peek under the hood on how it works. Perhaps it will wet your appetite to explore these programming practices further. We have provided links throughout which can provide you with further information on these topics.

Creating a table on the fly

If you look at the HTML source code of "adafruit_remote.html" near the very bottom you will see the following HTML code defining a table called "RemoteTable". You will notice that this table is empty. However when it is displayed, it contains rows and columns of buttons for your remote. Where do the buttons come from? The answer is we create them using JavaScript. Here is the section of the HTML code.

```
&lt;table id="RemoteTable"&gt;
&lt;/table&gt;
&lt;script&gt;myInitialize(); &lt;/script&gt;
```

This table definition is followed by a script command to call "myInitialize()". That is a JavaScript function that initializes various JavaScript variables and particularly cause a function called "ShowRemote()" which takes the data from the variable "Buttons" near the top of the page and converts it into rows and columns of the table. It uses JavaScript commands such as "innerHTML" and "appendChild(...)" to insert HTML tags into your webpage as it is loaded. Each cell of the table contains a parameter "onclick='DoButton(r,c)" so that when you click on that particular button it calls the JavaScript function "DoButton(r,c)" with the row and column number of the button which you clicked. It in turn calls SendButton which extracts the data from the Buttons structure and puts together the information to be sent to your Arduino device.

The Arduino sketch implements something called a REST API interface. We will discuss that in the following sections.

Understanding REST APIs

This implementation uses what is called a REST API interface. REST stands for REpresentational State Transfer. It is not really a protocol but it is a design strategy for transferring data using HTML. Basically you use the data of the URL that you are getting or putting to communicate with a remote device. You can find more information on REST design at http://www.restapitutorial.com/lessons/whatisrest.html ( ) or other similar websites.

In our implementation we are using the URL to transmit the protocol number, data value, and number of bits. So for example if your device is at IP address 196.168.1.105 you could simply browse to this web URL.

`http://192.1 68.1.105/irsend/2/12345/20`

The first thing after the IP address is the word "irsend". This would probably be unecessary because all of our commands to this device are in the form of an infrared send command. However we have implemented it this way so that if you wanted to send other commands to this device you can modify the program to do something different depending upon this first item after the IP address. Then separated by slashes we have the protocol number which in this case is 2, the data value in decimal, optionally followed by a slash and the number of bits. Note that we normally record data values in hex but our interface as we have designed it must be in decimal.

One of the problems with simply accessing that URL with a browser is that sometimes it will only work once. That is because your browser tries to cache the results. It will not actually access the URL again and thus the device is not triggered. There are some things we could do to the arduino sketch that would tell the browser not to cache the data and force it to fetch every time but from my research, there are a variety of methods to do that which may or may not always work. A simpler solution is instead of doing an HTML GET we can do an HTML PUT and not have to deal with the problem. Also from a theoretical point of view we are sending data to the device, so a PUT makes more sense than a GET. However to do a PUT we need a bit of fancy JavaScript. We can't just point our browser to a particular URL.

Before we talk about the JavaScript used to do the HTML PUT, let's take a peek at what that PUT command looks like when it is received by your board. Use the IoT_IR sketch with the SERIAL_DEBUG set to 1 and your serial monitor open. At line 47 there is a #if(0) that you should change to #if(1) to see what's really being sent. On your serial monitor you will see something like this.

```
""PUT /irsend/1/16582903/0/ HTTP/1.1
Accept: */*
Accept-Language: en-US
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:
11.0) like Gecko
Host: 192.168.1.131
```

```
Content-Length: 0
Connection: Keep-Alive
Cache-Control: no-cache
```

It is the job of the code in your Arduino sketch to capture this data using "client.readString()" or "client.read()" or "client.parseInt()" calls to extract the various parts of the URL. The main loop makes sure that it begins with "irsend" and then it calls the function "processIR(client)" to process the rest of the URL and get the protocol number, data value, an optionally number of bits.

In the next section we will talk about how we used JavaScript to create the HTML PUT command.

JavaScript HTML PUT with callback function

I've been a computer programmer for over 40 years but a few years ago when I first learned how to do JavaScript HTML programming using callback functions, it nearly fried my brain. There was a paradigm shift that took me a while to understand. I was always taught that when you execute a programming function, it starts with the first line of code and executes them in sequence top to bottom with possibly conditional statements and/or loops thrown in the middle. But because JavaScript is interpreted and not precompiled, things work bit differently. When dealing with HTML communication you pass information to the remote device about what you want it to do. But then you also pass it what is called a "callback function". This is a piece of code that will be executed only after the remote device responds to your request. You have no idea when that response is going to come. So even though the function definition is included in your code at a particular point in the source code, it is not going to get executed immediately.

It helps to think of a callback function somewhat like an interrupt handler. There's nowhere in your in your code that explicitly calls this function. Rather the function gets called when a certain event occurs. In this case when the response comes back from the HTML device to which you have sent a request.

We said that we made a call to the function "getRest(url,cb)". The first parameter is rather simple. It would look something like "http://196.168.1.105/irsend/2/12345/20/". The second parameter "cb" is the callback function.

This function DOES NOT get executed when you call "getRest". It looks like it does but it doesn't. It only gets called when your device acknowledges that it has received the REST command. In this case the function takes the response text returned by your device, converts it into a JSON formatted string and uses the innerText command to display it on your webpage. Each time you click a button on the remote you can see the text that is returned.

For example the webpage might display something like this

`{"status":200,"statusText":"OK","data":{"command":"irsend","protocol":1,"code":16609423,"bits":0}}`

The contents of the "data" field was created by the Arduino sketch in the "processIR" function using "client.print()" commands.

Inside the getRest() function is where we do the actual HTML PUT command. The heart of the command is the "XMLHttpRequest()" object which we create in the variable "Request". We use various methods and variables of that object to set up the request and then eventually transmit it. The first variable we set is the "Request.onreadystatechange". Again this variable is a callback function definition. This code does not yet executed where we have defined it. Rather we have created batch of code that will get called every time the ready state of our request changes. Inside this function we check for the ready state to be equal to 4 which means that the request has been completed.

The "Request.open("PUT", url, true)" method tells it that this is a HTML PUT command and passes it the REST URL we want to put. Finally we "Request.send()".

For more information about XMLHttpRequest we suggest the following resource https://www.w3schools.com/xml/xml_http.asp ()

We haven't intended to make this a complete tutorial on how to use JavaScript and callback functions and all the other intricacies of this implementation but we didn't want to give you a peek under the hood, it works. We suggest you use some of the links we have provided if you want to explore these topics further.