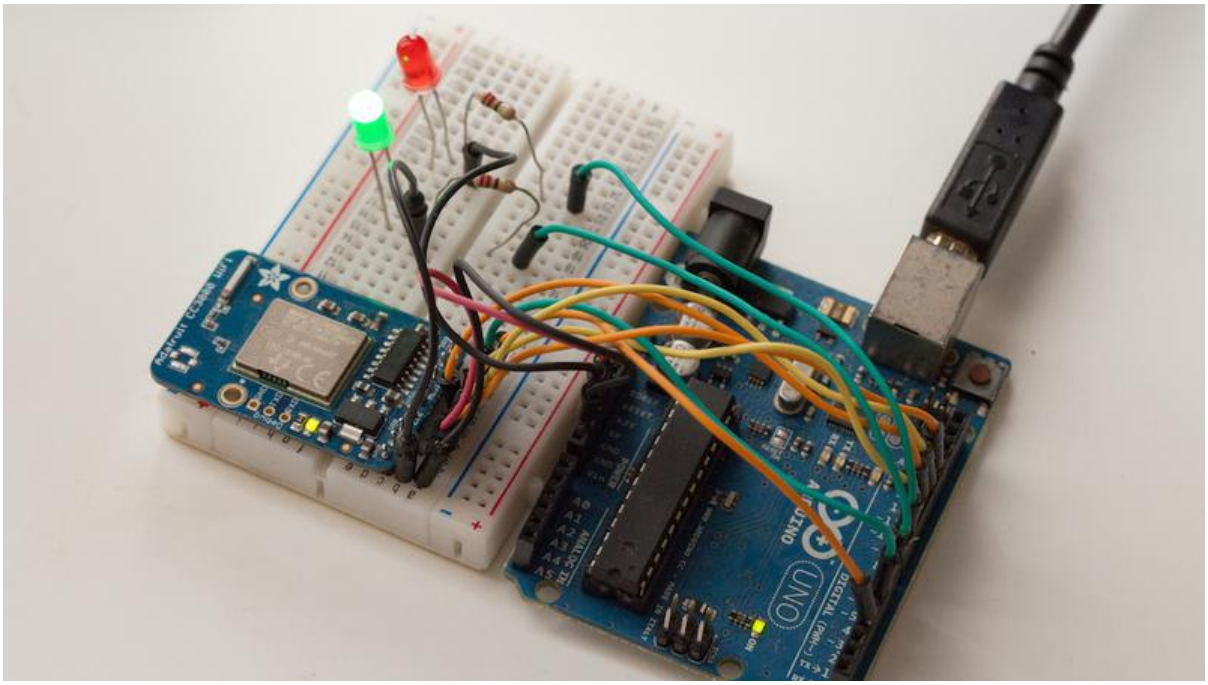
A REST API for Arduino & the CC3000 WiFi Chip

Created by Marc-Olivier Schwartz



https://learn.adafruit.com/a-rest-api-for-arduino-and-the-cc3000-wifi-chip

Last updated on 2022-12-01 02:10:20 PM EST

Table of Contents

Overview

I was recently playing with the Arduino Yun for a whole set of new projects, and I discovered that Arduino was providing an official REST API for the Arduino Yun, in the form of an Arduino sketch. And after using it for a while, I thought about a new idea: what not do the same for the Arduino platform in general?

But first, let see what a REST API really is. It's actually a concept that is widely used by many web applications like SaaS (Software as a Service). REST stands for REpresentat ional State Transfer, and is a communication architecture that was created back in 2000. It defines many constraints that the architecture has to follow, the most important features being:

- A client/server communication: a client sends a request to a server, and the server answers accordingly
- A stateless communication: every request has to contain all the information so the server can understand it, without having to rely on some information stored on the server
- A uniform interface to easily identify resources on the server

This allowed to standardise communication between web applications, and made them more scalable, faster, and simplified the development of more complex applications. And for our Arduino projects, it allows to standardise the communication between your Arduino and the external world via WiFi or Ethernet, and develop complex applications without having to modify your Arduino sketch every time.

Indeed, I have been using the CC3000 WiFi chip for a while, and the problem that I encountered while developing web applications using the CC3000 chip is that I had to create a new Arduino sketch for every application, that needs to be coordinated with the rest of the application, for example an interface running on my computer. For example, using this REST API, switching the state from a pin on the Arduino board can be done directly in the browser by typing the following URL:

http://arduino.local/digital/8/1 ()

With this REST API, it's easy to load a sketch once for all on your Arduino, and then only work on the interface on your computer that makes REST calls on your Arduino board. And for now, this kind of interface was only available on the official Arduino boards, like the WiFi & Ethernet shields, and the Yun.

But I wanted to have the same on my CC3000 WiFi chip: make a sketch that will create a web server on the Arduino board, and then accept REST commands from an external client, like from an interface running on my computer. That's why I created a dedicated library () that encapsulate these ideas, and even allows you to create your own functions that can be called from the API. And in this guide, we are going to see how to use this library.

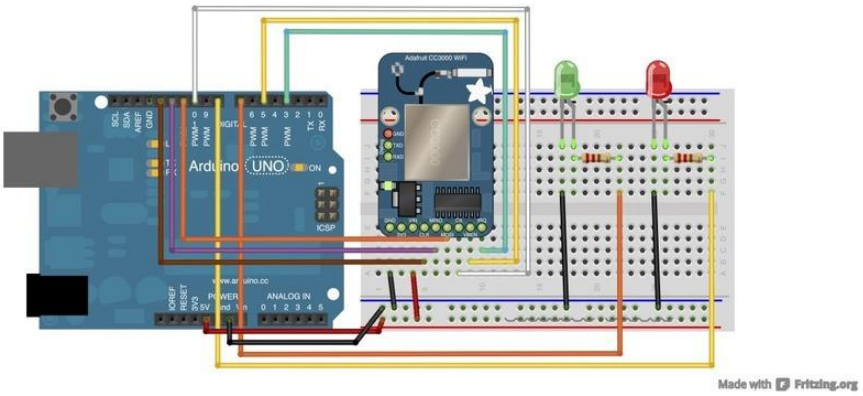
Hardware configuration

On the hardware side, you have a lot of options for this project. I tested it with an Arduino Uno board and an Adafruit CC3000 breakout board, but it should work with other Arduino boards like the Arduino Mega. You can also use the Adafruit CC3000 Shield to replace the breakout board.

I also added two LEDs to the project, just to visualise the effect of REST calls. But you can also add analog or digital sensors to the project to test other functions of the library.

To connect the breakout board, connect the IRQ pin of the CC3000 board to pin number 3 of the Arduino board, VBAT to pin 5, and CS to pin 10. Then, you need to connect the SPI pins to the Arduino board: MOSI, MISO, and CLK go to pins 11,12, and 13, respectively. Finally, take care of the power supply: Vin goes to the Arduino 5V, and GND to GND.

For the LEDs, I simply connected each of them in series with a 220 Ohm resistor, on pin number 7 and 8 of the Arduino board. The following picture summarises all the hardware connections:

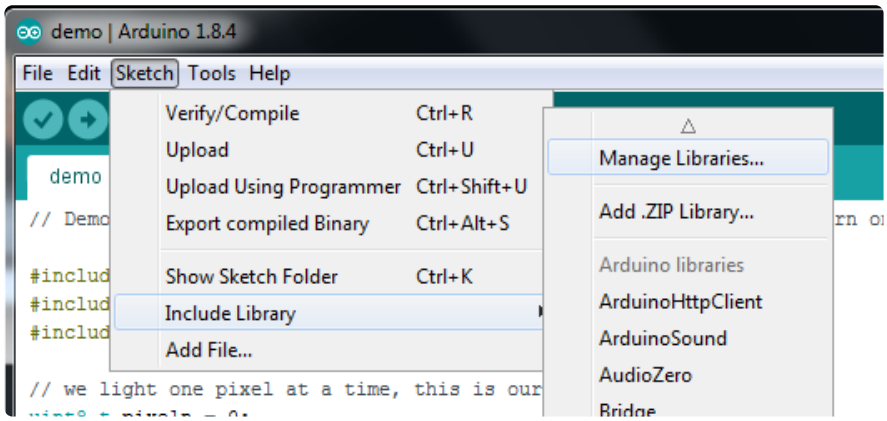


Installing the library

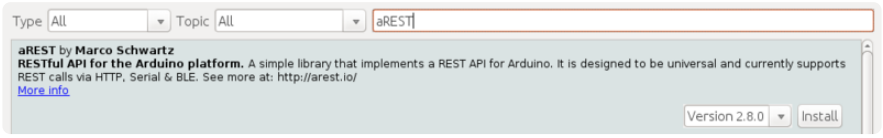
The first step is to get the library that will handle the REST calls, which is called aREST. It is hosted on GitHub and you can visit the page at the following link:

Get the code on GitHub

To install the library, first open up the Arduino library manager



Search for the aREST library and install it



We also have a great tutorial on Arduino library installation at: http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use ()

Basic functions

Now that everything is in place, we can test the project. We'll start with the basic functions of the REST API, which are the basic functions of Arduino: pinMode, digitalRead, digitalWrite, analogRead, and analogWrite.

The software part is actually not complicated at all: everything is already done for you in the WiFi example of the library. For that, simply open the file called "WiFi_CC3000.ino" in the examples folder of the aREST library. Save this file to another folder so you can edit it.

You will have to change these lines so the sketch can connect to your WiFi network:

```
#define WLAN_SSID       "yourSSID"
#define WLAN_PASS       "yourPassword"
```

For now, let's not look at the rest of the sketch, and simply compile the sketch & upload it to the Arduino board. You can now open the Serial monitor. If everything works fine, you should see that the board is connecting to your WiFi networks, and starts listening for incoming connections.

You can now head over to your favorite web browser. First, we have to set one of the LED pin as an output. Just type:

http://arduino.local/mode/7/o ()

This should set the pin as an output, and you should get the confirmation printed in your browser:

Setting pin D7 to output

If it doesn't work at this point, don't worry (yet!). The sketch is using the mDNS library so the board can be accessed using arduino.local. But you can also use the IP address of the board directly. To get the address, simply uncomment the "displayConnectionDetails" function in the sketch and look at the IP on the Serial monitor. You can then use this IP in place of the arduino.local in your browser.

We can then continue using the REST API and turn the LED on. For that, we need a digitalWrite function used on pin number 7. This can be done again in the browser by typing:

http://arduino.local/digital/7/1 ()

The LED should turn on and you should have the confirmation message in your browser:

Pin D7 set to 1

At this point, you should see how powerful the project is: you can now build your

projects on the client side only, without having the modify the Arduino sketch every time. You can have the sketch loaded once for all, and then simply change which REST call you make.

Let's try one more of the basic functions, for example analogRead(). To test it, simply connect a wire from the Arduino 3.3V pin to the analog input A0 of the Arduino board. As the Arduino Analog-Digital converter works with a 5V range over 1024 values, we should get something around "722" being readout by the project (3.3/5 * 1024). Simply type the following command in your browser:

http://arduino.local/analog/0/r ()

I got this printed in my browser:

{"value": 722, "id": "008", "name": "mighty_cat", "connected": true}

Here you can see that the value is return in a JSON container, so it can be used directly by other apps and languages, for example by PHP. You can also notice that other information is being returned in the JSON container, that identify the board with an id and a name. You can change these directly in the Arduino sketch.

Note that you can find all the information about the aREST library and the REST API on the GitHub repository of the project (), by reading the README file.

More features

But the aREST library actually includes more than just the basic Arduino functions. The first feature that we are going to see is the access to variables stored on the Arduino board. If you have a sensor that can be read directly using basic Arduino functions, like an analog temperature sensor, you can directly get the value from the sensor by calling the analog REST call, just as we saw in the previous section.

However, if you are using sensors that have a more complex behaviour, for example sensors that use a dedicated library, you want to store the measurement in a local variable first, before getting it with the REST API. The first step to get a variable with a REST call is to declare it in the sketch, with an alias name. Note that for the moment, only integer variables are supported by the API. This is done by the following piece of code in the exemple sketch:

```
rest.variable("temperature",&amp;temperature);
```

The, this value can be accessed by the REST API by typing the following command in your browser:

http://arduino.local/temperature ()

This is the answer you will get in your browser:

{"temperature": 24, "id": "008", "name": "mighty_cat", "connected": true}

Once again, you will note that the answer is in JSON format, so it can be processed by other programming languages like PHP.

The other feature from the API that you can use is the ability to define your own functions, so they can be called directly from a REST call. Imagine for example the case where you want to execute a sequence of command: it is much more efficient to encapsulate all these commands in a single function, instead of making several REST calls. Note that all functions must return an integer type, and take a String as the unique argument, which will contain the different parameters for the function. You can see for example the function defined in the example:

```
int ledControl(String command) {

  // Get state from command
  int state = command.toInt();

  digitalWrite(7,state);
  return 1;
}
```

Just as the variables, the functions have to be declared with an alias so they can be called by the API:

```
rest.function("led",ledControl);
```

The function can then be called by the REST API in your browser, using the name of the alias followed by the parameters:

http://arduino.local/led?params=1 ()

Once you sent this REST call, you should have the confirmation that the function has been executed:

Function led has been executed

Using these features, you can define your own variables and functions and call them

directly from the REST API. For now, the library only supports 2 variables and 2 functions for memory limitation reasons, but you can go inside the library files to extend these limits (at your own risks!).

Application: a basic web app

Now that we saw the basics of the REST API for Arduino, we can use that to build a simple web application that will run in your browser, with buttons to control the two LEDs that are connected to your Arduino board.

For this part, you'll need a working web server (like Apache) running on your computer, and you will need to put all the files at the root of your web server main folder.

You can find the complete code for this part by following this link:

Get the code on GitHub

First, to make sure that the library is working, go in a browser and set the two LED pins to outputs:

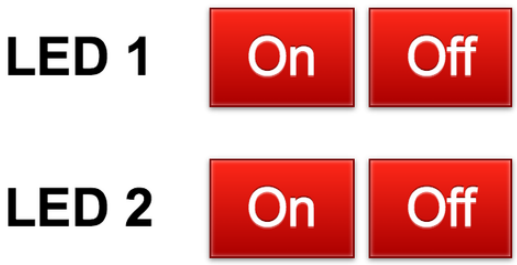
http://arduino.local/mode/7/o ()

http://arduino.local/mode/8/o ()

Once this is done, we are going to code the web app. The app is basically composed of three parts:

- An HTML page that contains the interface (I also added some CSS to make it look better)
- A JavaScript file that handles the commands coming from the interface
- A PHP file to communicate with the Arduino board by making REST calls

To give you some idea about what we want to build, here is the completed interface:



Let's talk about the HTML part first. The interface basically consists of these four buttons, which are defined by the following code:

```
&lt;button class="relayButton" type="button" id="1"
  onClick="buttonClick(this.id)"&gt;On&lt;/button&gt;
```

See this "this.id" argument that is passed on each click of the button? This is what we are going to use in the JavaScript file to know which command to send to the Arduino board using the REST API. For this particular button, here is how it is handled by the JavaScript file:

```
if (clicked_id == "1"){
  $.get( "curl.php", {
    pin: "7", state: "1"} );
}
```

You can see that every time this button is pressed, we call a file called "curl.php" which contains the functions to make the REST call to the Arduino board.

This PHP file starts by getting the pin & state to be sent to the Arduino board:

```
$pin = $_GET['pin'];
$state = $_GET['state'];
```

We then build the URL that will be called by the PHP script:

```
$service_url = 'http://arduino.local/digital/' . $pin . '/' . $state;
```

Then, we can create the cURL object that will do the REST call. cURL stands for Client URL Request Library, and this is what we will use to do like we would type the URL in a browser. This cURL object is initialised with:

```
$curl = curl_init($service_url);
```

And execute it, while storing the answer in a variable:

```
$curl_response = curl_exec($curl);
curl_close($curl);
```

This last line is optional for this app, but in case we want to get back some data from the Arduino board (for example if we called a variable on the board using the REST API) we can use the echo function to print this result:

```
echo $curl_response;
```

You can now go to your web server (usually on localhost) and open the HTML file. You can try to click on the different buttons: the state of the LEDs should change instantly, just like if you would type the REST URLs in your browser.

Future work

Using this library that implements a REST API for Arduino & the CC3000 WiFi chip, we saw that we can easily create web-based apps while having a standard way of communicating with your Arduino board. And this means that you don't have to reinvent the wheel at every new project you create using the CC3000 chip and Arduino.

Of course, you can build more complex applications that the simple app we saw in the previous section. You can use the functions to readout values from variables that are stored on the Arduino board, to get & display measurements coming from sensors. Or create your own functions that you then expose to the REST API, for example to control mobile robots.

In the future, the goal is really to build a universal REST API for Arduino. This library will be extended to other systems than just the CC3000 chip, for example to serial communications. This way, you can have hybrid systems composed of many different wireless hardware like XBee, WiFi or Bluetooth, all communicating transparently with a central interface, using the same REST commands.