Internet Connectivity

Introduction

With the explosion of computing power and connectivity, the Internet has become a primary means of storing, processing, and delivering data. Cloud computing, social networks, search engines, the "Internet of Things" – each of these require sharing and storage of information, some of it in large quantities (so-called "big data"). This document lays out the fundamentals of connecting a microcontroller to an online database system.

The flow of data will happen in several steps, and tutorials and sample code will be provided to help you understand each piece of the puzzle. You'll start with simple, "Hello, world!" applications and work up to a website that displays key statistics of your system (eg, temperatures) and allows you to *control* your greenhouses from the Internet.

Objectives

Upon successful completion of this lab, the student will be able to:

- Use a microcontroller to invoke scripts for retrieving data from a database,
- Use a microcontroller to invoke scripts for storing data in a database,
- Transfer data (bi-directionally) between an Arduino and a web server, and
- Implement a simple web interface for setting control parameters.

Preparation

- Show up.

Communication from a microcontroller

Each group will be provided the following:

- An ESP8266 WiFi module from SparkFun. In the interest of full disclosure, this is the first time the class will use the ESP8266. In previous editions we've used the CC3000 from Adafruit, which was the beginning of the end of my patronage thereof.[1] The ESP8266 is

[1]Gory details provided upon request.

a popular chip, but comes with it's own quirks and mediocre libraries. Your instructor has re-written some of the libraries, but there can always be unforeseen issues – your patience is appreciated.

The ESP8266 communicates by means of a UART – a "serial" link on your microcontroller. To avoid having to rely on the Software Serial library, it makes sense to switch to a more powerful chip that has more memory and better communication options, so we'll switch to the SAMD21 mini from SparkFun. An added benefit is that both the SAMD21 and the ESP8266 are 3.3V systems, so no level shifting will be needed.

Setting up the Arduino IDE

The first thing you'll need to do is set up your Arduino environment to work with the SAMD21.

Procedure

1. Point you browser to the SAMD21 Hookup Guide and follow the instructions there. After you complete the "Setting Up Arduino" section, do the Analog Input and Ouput example sketch: Remember when we talked about PWM? This chip has a built-in *digital to analog converter*, which allows you to make true, variable voltages, not just simulate them with PWM.
2. You may have noticed that uploading files takes a *long* time with this chip. That is because the default bootloader isn't a very good one. Your instructor has found a better bootloader, which he can load on there with some special hardware. If you want a faster upload, come see him when he has a free moment (ha!) and he'll load the new one on.

Basic connectivity

The next thing to do is make sure that you can connect the SAMD21 to the Internet. In this case, you'll communicate with the WiFi module using "raw" `AT` commands, which constitute an API for communicating with the module. There are many `AT` commands one can send to the chip, but we'll only need a few to accomplish what we want to do. For most commands, there is a response, which may be as simple as acknowledging the command with an `OK` or as complex as returning a webpage response. Some common examples are:

Command	Response	Action
AT	OK	attention
AT+RST	OK	reset
AT+CWLAP	list of WiFi networks, then OK	list access points

Table 1: A small sampling of AT commands.

You can find a fuller list of `AT` commands here and a more detailed reference here. There's also a summary on the ESP8266 pinout diagram linked above.

Procedure

1. Consult the pinout diagrams for the SAMD21 and the ESP8266 and make the following connections:
   - Connect VCC on the SAMD21 to VCC and CH_PD on the ESP. VCC in both cases is 3.3V. **Do not connect anything to Vin (RAW) on the SAMD21 board** – that is 5V. CH_PD is just the chip enable pin, which must be HIGH for the chip to operate.
   - Connect GND to GND.
   - Connect RX on each board to TX on the other.

   N.b., you'll need a pin "spreader" to be able to fit the WiFi module into a breadboard. Luckily, I have some.
2. Download `ESP8266_Serial_Passthrough.ino` from collab, store it in a directory in your sketchbook, and load it into the SAMD21 chip. All this program does is pipe text from the SerialMonitor (`SerialUSB` on the SAMD21 chip) to the ESP and vice versa.
3. Open the SerialMonitor and verify that it is set to send both a carriage return (CR) and newline (NL) with each entry (there's a drop down box near the bottom).
4. Type `AT` in the SerialMonitor. You should see a response, "OK". If not, check your wiring.
5. Type `AT+CWMODE=1` to set your WiFi module to station mode.
6. In order to connect to the `wahoo` network, we'll need to register your chip's MAC address with UVa. Before you do that, however, enter the following command:

   ```
   AT+CWLAP
   ```

   You should see a list of possible access points, including `Cavalier` and `Welcome_to_UVA_Wireless`. Probably others. Note that `wahoo` is hidden, so you won't see that particular network.
7. Consult the list of commands linked above to determine what command is used to get the MAC address of your device. Enter that command, write down the MAC address, and come see your instructor, who will enter the device into UVa's registry (you could register it, but I'll likely be here longer than you, so better it's on my account).
8. Now find the command to connect to a network ("access point") and connect to the `wahoo` network. What is your IP address?
9. Finally, find the option to automatically (re-)connect to a network. This will make your life easier, since the chip will connect by itself on power up – no waiting for you to tell it to connect.

The ESP8266 library

There are a number of libraries out there for interacting with the ESP8266. Your instructor pored over several options over Spring Break, carefully considered the advantages and disadvantage of each, and decided they all mostly stink. So he took the SparkFun library, trimmed it down substantially, and made it work with the SAMD21. You can download the library, `ESP8266_thin`, from collab.

You'll use the library to manage all of the `AT` commands and their responses – you only have to call a small set of functions to interact with the AWS server, for example. You'll start by reading a trivial webpage with your SAMD21 + ESP8266 combo, and work up to calling PHP scripts on the AWS server to insert data into your database. **If you have trouble with any of the tutorials, see an instructor sooner rather than later.** The library is reasonably well tested, but not guaranteed to be flawless.

1. Download the example WebClient code, `ESP8266_WebClient.ino` from collab, store it in its own directory in your sketchbook, and load it onto the SAMD21 chip. The provided code should contact `www.example.com` and spit out the contents of the website (in text format, of course).

Sending data to a database

Once you've demonstrated basic connectivity, the next step is to send information to a database. You'll do this by sending commands to the WiFi module that emulate the URLs you have used in the past to invoke PHP scripts. The PHP script will use the GET protocol to parse the data. With the GET protocol, the data are included in the http request string (essentially, the URL) with the following format:

```
http://<hostname>/<scriptname>?<variable1>=<value1>&<variable2>=<value2>...
```

The `?` character tells the script that a list of `variable=value` pairs is coming, each separated by the `&` character. An alternative method is to package data first and send it using the POST protocol. This is most often done with large datasets, such as from a web form, or when you need to conceal a password.

1. In the WebClient sketch, change the `destServer` to:

   ```
   ec2-34-209-142-24.us-west-2.compute.amazonaws.com
   ```

2. Change the `httpRequest` to

   ```
   GET /~gcl8a/addtoroster.php
   ```

   Add lines in the `ClientDemo()` function to add the following fields to the `GET` statement (n.b., I've had hit-and-miss luck with spaces – best not to use them for this tutorial).

   - id = [your computing id]
   - firstName = [your first name]
   - lastName = [your last name]
   - gradYear = [your year of graduation]
   - favoritePizza = [your favorite pizza]

   Upload the sketch to your Arduino and run it. Unless you tell it not to, the code will still call the server every 15 seconds, but don't worry! I've made your computer id a `PRIMARY KEY`, so it won't add you over and over.

3. When complete, point a browser to this script and check if your name is there. If it is, well done. If not, find an instructor.

4. Repeat the above for each member of your team.

Show an instructor that you have entered the data.

Instructor initials: ____________________________

Debugging PHP scripts

Debugging PHP scripts is a pain. It's very easy to make syntax errors – especially with quotation marks – and the scripts often don't fail nicely. For example, you may put a debug `echo` statement early in the script, hoping that will prove your script made it that far, but even if the error is after the `echo`, it usually won't print. One way to find the offending statement is to comment out the majority of the script and add the statements back line-by-line until you find which statement is causing the problem. If you `ssh` to the server directly (easily done on a Mac; it requires additional software on a Windows machine), things will go much faster because you can edit the script directly instead of re-uploading the file each time you make a change.

One problem that we see frequently is that students don't know if the problem is in their script or their Arduino sketch. It is quite easy to test a PHP script directly just by using a web browser. Simply point your browser to the script and manually enter all of the fields – that's all your Arduino is doing, anyway.

Whenever you have troubles writing to a database, in general, you should first confirm that you have a working PHP script using steps similar to that above before you start worrying about your Arduino sketch.

Sending greenhouse data to a database

In the end, you will combine your most recent Arduino greenhouse sketch with the WebClient sketch to send data from the integrated system. For now, you can just send dummy data to show that you can communicate with your PHP script.

1. Using the `clientDemo()` function as a start, write a function, `SendReadingToDB(...)` that takes `sensor_id` and `value` as arguments and calls the `insert.php` script you made in a previous assignment with the appropriate arguments. To make your lives easier, I've overloaded `tcpSendPacket()` to take a `String` as an argument. This allows you to write statements like:

   ```
   float val = 3.14;
   esp8266.tcpSendPacket(String("&value=") + String(val));
   ```

2. Since you don't have a temperature sensor connected, just set your sketch to periodically call the routine with the `sensor_id` of your inside temperature sensor and a typical value.
3. Upload the sketch and open the Serial Monitor. Let the sketch run for a minute or two.
4. Using the `mysql` command line, `SELECT` all of the readings from `sensor_readings` to show that the data were written correctly.

Show your working system to an instructor.

Instructor initials: ______________________

Retrieving information from the database

The last thing we'll do is get data *from* your database and use it in your Arduino sketch. Specifically, you'll make a web page that allows you to control the set-point of your greenhouse remotely. First, let's verify that your Arduino can receive information:

1. Edit the WebClient sketch to invoke a PHP script at

   `http://ec2-34-209-142-24.us-west-2.compute.amazonaws.com/~gcl8a/get_readings.php`

   a copy of which can be obtained (for later) on the AWS server by typing:

   ```
   cp /home/gcl8a/public_html/get_readings.php .
   ```

   This script retrieves the last 40 rows of data from HVAC database for my house and echoes it to a web page. (It's not formatted to be pretty, but it has both the value and timestamp for the chosen sensor). It takes as arguments an `id` (14 is the indoor temperature) and a `keyword` (tornbjerg) so that it has very basic security.

2. Upload your new sketch and open the Serial Monitor. If all went well, you should see the last 40 indoor temperatures at my house (along with a mess of html tags and other data).

   If not, see an instructor.

Once that works, you'll need to make a webpage that accepts user input and stores the value to a database that your Arduino can access. A copy of `changesetpoints.html` has been put in your professor's directory on the AWS server. This page allows the user to enter a desired set-points (for heating and cooling), a keyword for basic security, and sends them to a PHP script to put the set-points in a database. You can get a copy by moving to your `public_html/` directory and typing

```
cp /home/gcl8a/public_html/changesetpoints.html .
```

You'll need to:

1. Create a table called `setpoints` that holds the desired set-points, `SPheat` and `SPcool`. These are different from the recorded set-points in the `sensor_readings` table. Be sure to add an `id` field and a `timestamp` to keep track of when the set-point was updated.

2. Create a PHP script, `set_setpoints.php`, that accepts the input from the form, creates the appropriate SQL query, and executes it. Note that the form sends data via a `POST` call – make sure your PHP script uses `_POST` instead of `_GET` when you're ready to try from the form (but you might want to leave it as a `GET` call for initial debugging).

3. Create another PHP script, `get_current.php` that reads the `setpoints` table and returns the current set-points, ie, the most recent record.

4. Using the WebClient sketch as a template, periodically poll the server, parse the return string, and print the set-point to the Serial Monitor. Note that your PHP script will return much more than the set-points themselves; it will be useful to enclose the set-points in a tag, eg. `<setp>25,35</setp>`. Then you can search each line using the `startsWith()` and `endsWith()` methods in Arduino.

If all goes well, you should be able to change the set-point from a standard web browser and see the changes printed in the Serial Monitor.

Putting it all together

Let's see...you now have an Arduino that monitors and controls the temperature in your greenhouse, and the infrastructure to view and change parameters from a web browser (at the moment, stored data can only be accessed in text form – we'll add graphics later). It's time to put it all together. Your task is to build a system with the following functionality:

- As before, temperature control with a 1C hysteresis band.
- A heating set-point and a cooling set-point that can be changed from the Internet.
- Stores interior and exterior temperatures, current set-point, and power to the heater in a database. Also, add a flag for the position of the lid. Write the data every 30 seconds.
- Continue to write temperature and set-point data to the LCD, as before.