CURIE Academy, Summer 2014
Lab Notes 3: "Smart Door" IoT System

Prof. Christopher Batten
School of Electrical and Computer Engineering
Cornell University

In the first two labs you explored the field of computer engineering both from the hardware perspective and the software perspective. In this lab, you will put hardware and software together to create your first basic IoT system: a "smart door" that includes an IoT device to send the door status to the cloud and a different IoT device to poll the cloud and display the door status using a small light. These lab notes provide a brief survey of background information relevant to understanding the purpose and context for the lab.

Figure 1 illustrates the overall template that we will be using in all of our IoT systems. Each IoT system is comprised of an IoT input device, IoT cloud, and IoT output device. The IoT input devices will have various input modules attached that can sense what is going on in the environment (e.g., light, temperature, motion, force, current, pulse, liquid) and be able to upload data into the cloud. We will be using Xively as our IoT cloud service. Each IoT output device will be able to download data from the cloud and will have various output modules attached to display data (e.g., LEDs, piezo buzzer, mini-printer) or react in some way (e.g., servo, relay for controlling appliances). In these lab notes we will briefly discuss the IoT device and the IoT cloud. By the end of this lab you should have enough understanding of IoT systems to begin work on your design project.

1. IoT Device: Arduino with WiFi Board

Figure 2 illustrates the specific IoT devices we will be using in our IoT systems. Note that similar devices are used for the IoT input devices and the IoT output devices; the only difference is the code and the attached input/output modules. Each IoT device is contained within a plastic enclosure and contains an Arduino board, a small breadboard for prototyping, a WiFi module for connecting to the cloud, an LCD panel for displaying output, a 9V battery, and an on/off switch. Although you will occasionally run your IoT device off of the 9V battery, you should primarily leave your IoT device plugged into the host computer through the USB cable to avoid draining the 9V battery too quickly.

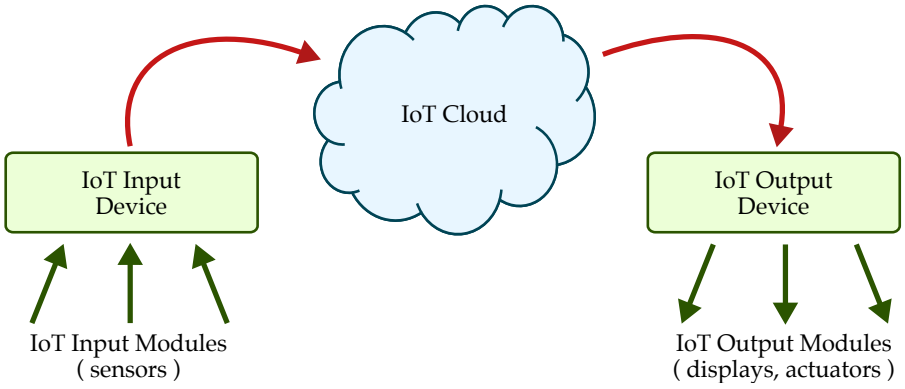

Figure 1: Diagram of IoT System Template

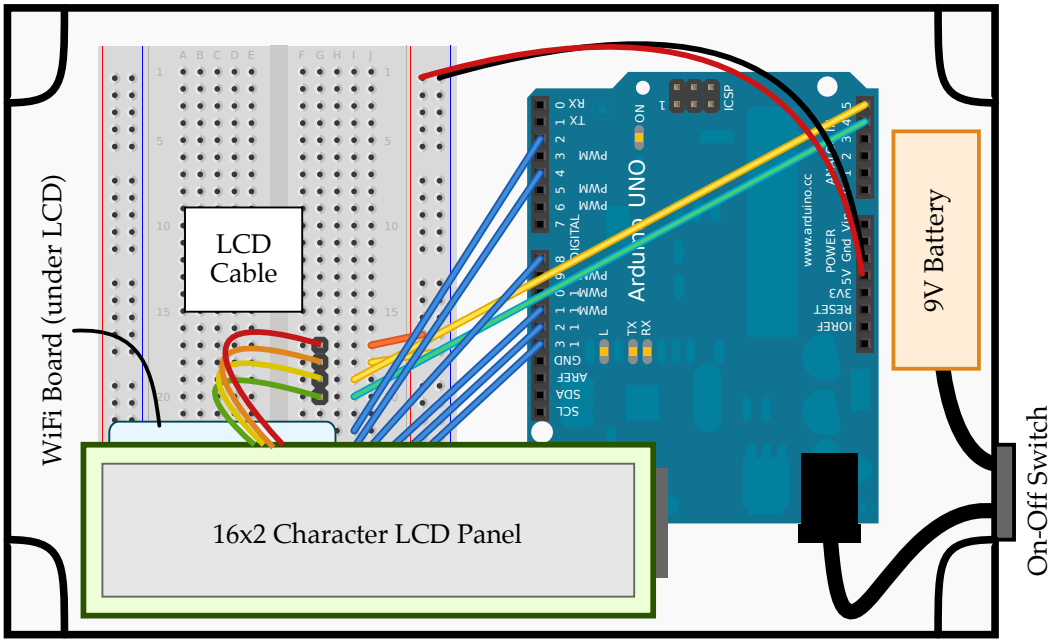

Figure 2: Arduino-Based IoT Device

From Figure 2 you can see that digital pins 2, 4, 8, 11, 12, 13 are used by the WiFi module. **NOTE: Digital pins 0 and 1 cannot be used because they are used by the serial monitor!** This leaves digital pins 3, 5, 6, 7, 9, and 10 for you to use in this lab and in your projects. From Figure 2 you can also see that analog pins 4 and 5 are used by the LCD panel. This leaves analog pins 0, 1, 2, and 3 for you to use in this lab and in your projects.

The LCD panel can display two rows of 16 characters. If the LCD panel is difficult to read it may mean your 9V battery is running low and needs to be replaced. It might also mean that the contrast potentiometer on the LCD panel needs to be adjusted. Ask an instructor to help you tune the contrast potentiometer if necessary.

The IoT input/output modules will usually require you to use the breadboard prototyping space. You can use what you learned from Lab 1 to build simple circuits using resistors and/or LEDs. Note how the cable from the LCD panel is attached to the breadboard using double-sided male header. We then use the breadboard to connect power and ground, and we can use additional wire to connect to pins on the Arduino. We will use a similar approach when attaching cables from input/output modules. The plastic enclosure has a hole near the breadboard for sensor cables. **For the design project you are free to modify the enclosure as necessary.** We can drill more holes to mount sensors or actuators.

2. IoT Cloud: Xively

A variety of different IoT cloud service providers are starting to enter the marketplace. These service providers enable IoT devices to connect to a centralized server for uploading and downloading data. For this lab and our design projects, we will be using Xively (`http://www.xively.com`). We have created a shared Xively account for all students to use with username `curie2014@csl.cornell.edu`. The password will be distributed in lab. After logging into Xively click on the *Develop* tab in the

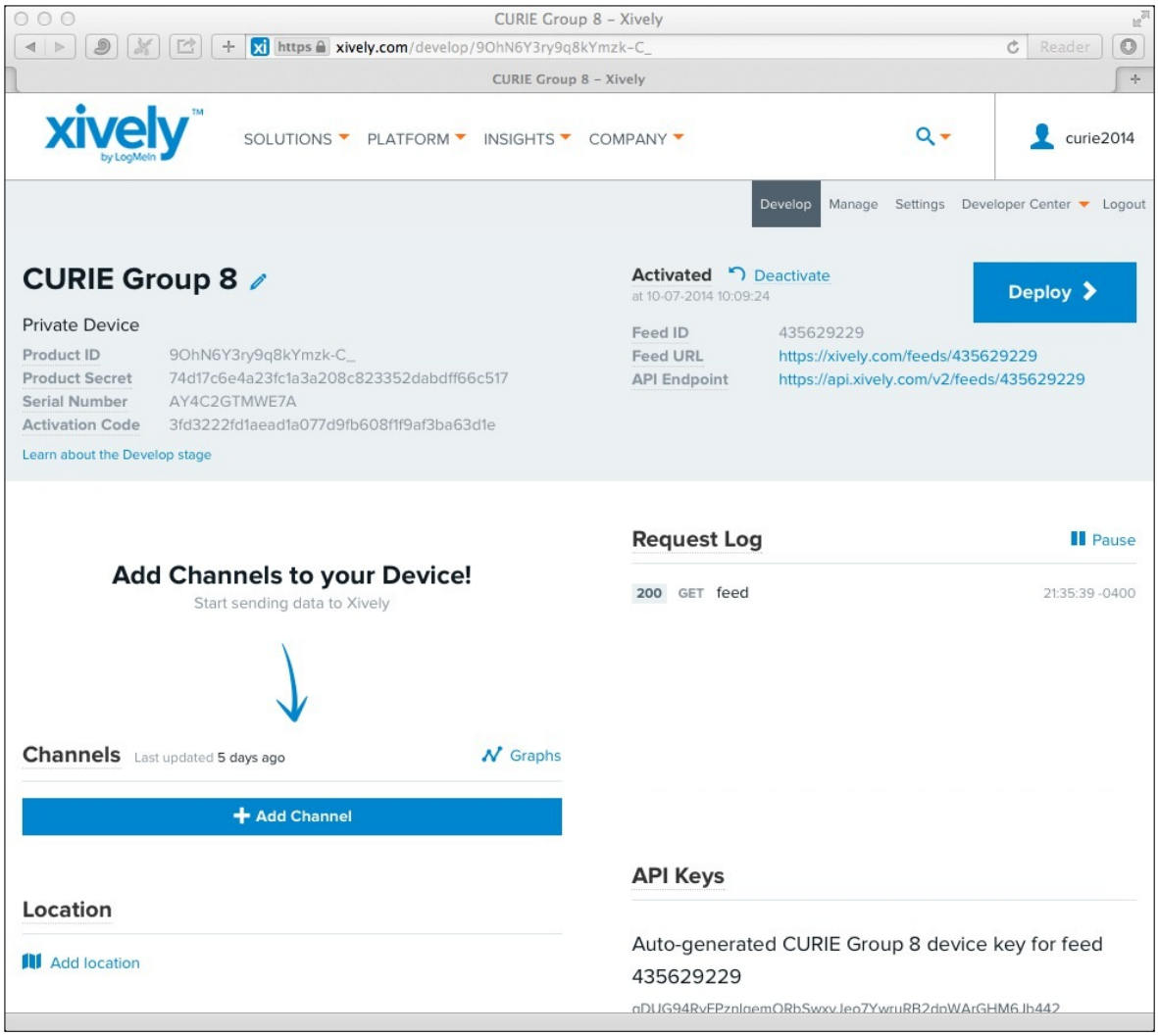

Figure 3: Initial Xively Page With No Channels

upper right-hand corner. You will then see a list of the CURIE groups. Click on your group number. You should now see something similar to Figure 3.

You will use Xively "channels" to communicate between your IoT device and the cloud. You can add new channels to Xively by simply clicking on the *Add Channel* button. Each channel has a name and a current value. Note that you can set the current value either by having your IoT device upload a new value, or you can also change the current value directly through the Xively web interface. Figure 4 shows the Xively page after we have added a new channel and uploaded some data. Each channel can either be set to just show its current value or to show a plot of how the channel value has changed over time. The request log area will show all requests your device makes to either upload or download data.

We will use special routines in the code running on our IoT input and output devices to communicate with the cloud. Table 1 shows the routines that are currently available for you to use (we may add more as needed). In the `setup` function, your code should call `curie_cloud.begin(CURIE_CLOUD_GROUP(N))`

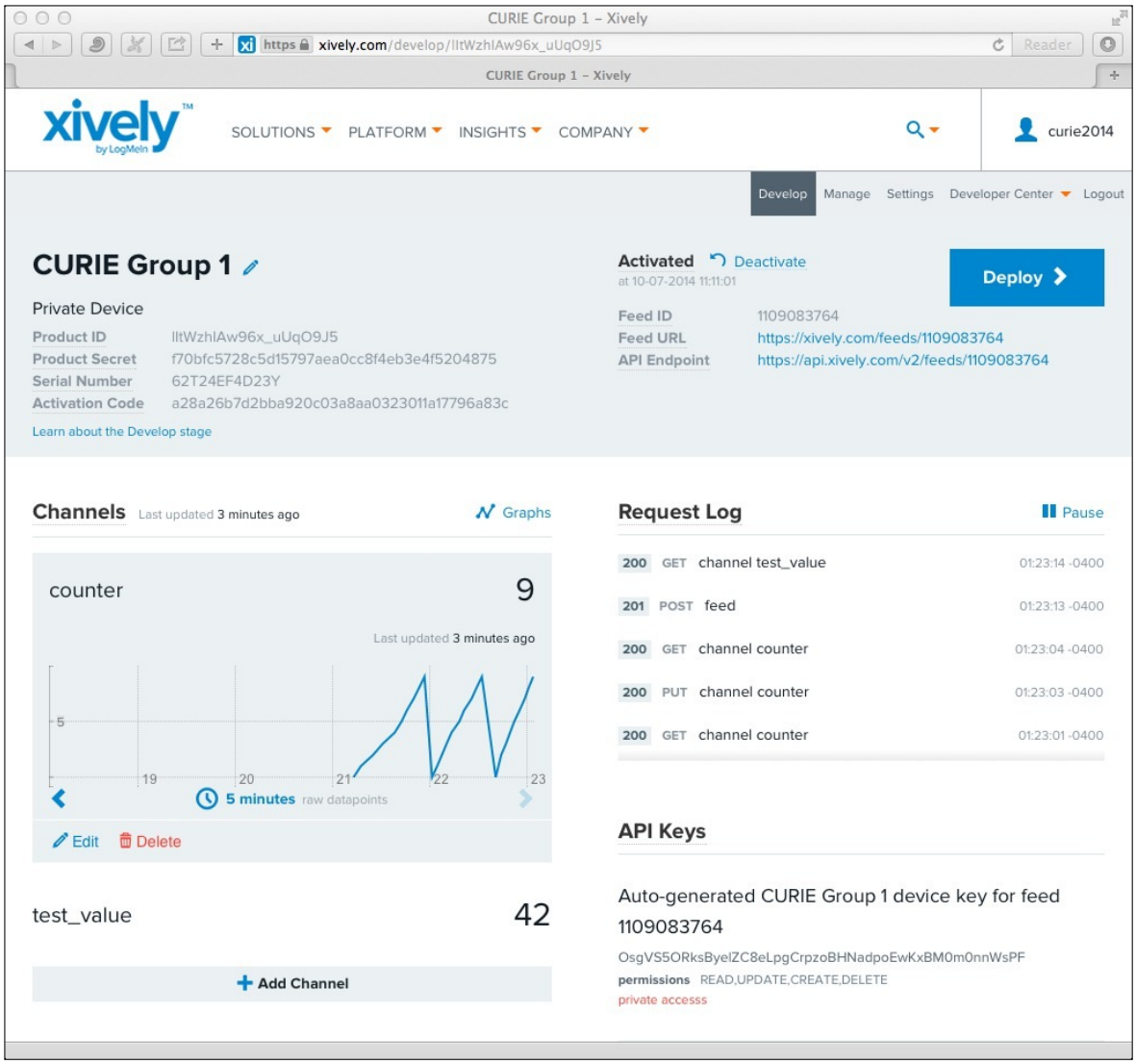

Figure 4: Initial Xively Page With Two Channels

where `N` is your group number. Then you can use the various `send` and `recv` functions in the `loop` function to send data to the cloud and receive data from the cloud.

Figure 5 illustrates a simple program that uploads a counter to the "counter" channel in the cloud and downloads data from a "test_value" channel from the cloud. Lines 3–5 load in some libraries that the program needs to communicate with the cloud and the LCD panel. Line 11 initializes the cloud; notice how we have indicated that this device is for CURIE Group 1. Before running this code you would need to setup two new channels in the Xively workspace for CURIE Group 1: one called "counter" and one called "test_value". Line 21 sends data into the cloud and Line 23 receives data from the cloud. Lines 25–26 display the data received from the cloud on the LCD panel.

If you run this code while it is attached to the serial monitor, then you will see some debug information as the Arduino initializes the WiFi board and connects to the cloud. The LCD panel will also display debug information when the device is being initialized. You should also see some debug

`curie_cloud.begin(CURIE_CLOUD_GROUP(N));`	Initialize the cloud, N = group number
`curie_cloud.send_int( "channel_name", value );`	Send integer to given channel in cloud
`curie_cloud.send_float( "channel_name", value );`	Send float to given channel in cloud
`curie_cloud.send_str( "channel_name", value );`	Send string to given channel in cloud
`value = curie_cloud.recv_int( "channel_name" );`	Receive integer from given channel in cloud
`value = curie_cloud.recv_float( "channel_name" );`	Receive float from given channel in cloud
`value = curie_cloud.recv_str( "channel_name" );`	Receive string from given channel in cloud

Table 1: Curie Cloud Routines

```
1  // We need to include three libraries
2
3  #include <Curie.h>
4  #include <Wire.h>
5  #include <SPI.h>
6
7  // The setup routine runs once when you press reset
8
9  void setup()
10 {
11   curie_cloud.begin( CURIE_CLOUD_GROUP(1) );            // Setup the cloud
12   curie_lcd.clear();                                    // Clear the LCD panel
13 }
14
15 // The loop routine runs over and over again
16
17 void loop()
18 {
19   for ( int i = 0; i < 10; i++ ) {
20
21     curie_cloud.send_int( "counter", i );               // Send i to counter channel
22     delay(4000);                                        // Wait 4 seconds
23
24     int value = curie_cloud.recv_int( "test_value" );   // Receive data from test_value channel
25     curie_lcd.setCursor(0,0);                           // Display test value on LCD panel
26     curie_lcd.print(value);
27     delay(4000);                                        // Wait 4 seconds
28
29   }
30 }
```

Figure 5: Simple Program to Send/Receive Data To/From the Cloud

information on the serial monitor every time data is sent to the cloud. This debug information can be useful to ensure that the device is still communicating with the cloud. You should be able to see the counter value changing over time.

Notice that we add a four second delay after we access the cloud. Xively limits how many times you can access the cloud per minute to avoid overloading their servers. The limit is 25 accesses per minute calculated as a rolling average over three minutes. This means if you try and access the cloud more than 25 times in a minute, Xively will block further accesses for a few minutes. You will need to wait before trying again.