CNT5517 - Section 1G92. CIS4930 - Section 015E

Mobile & Pervasive Computing

Spring 2017

Professor Sumi Helal

Lab 3 — IOT-Enabled Metronome with ARM mbed

Due Date: 11:55pm February 27, 2017.

ARM mbed and the FRDM K64F allow one to easily program a microcontroller while simultaneously preparing the device for the Internet of Things. In this lab, you will create a simple metronome using the buttons and LEDs on your FRDM board, and then create a web interface to interact with the device remotely.

Part One — Embedded Application & RESTful API

Background
A metronome is a device that creates a visual or audible pulse at consistent intervals. A common measurement for the speed, or tempo, of a metronome is Beats per Minute (BPM). A metronome at 60 BPM will pulse once every second, at 120 BPM twice every second, etc.

A common way to set the speed of modern metronomes is for the user to repeatedly press a button at the desired tempo. The metronome will calculate the average time between the presses and set the BPM accordingly. You will emulate this feature using one of the K64F's built-in buttons.

Requirements
For the first part of this lab, you will create a metronome-style feature on your mbed board. The metronome should have two modes; a "play" mode, which will blink an LED at a given tempo, and a "learn" mode, in which the user will repeatedly press a button to set the tempo. The two modes will be toggled between with the other onboard button.

In the "play" mode, the *green* LED will blink repeatedly at the previously specified tempo. On startup, this should be the default mode, with a tempo of 0, meaning the LED will always be off.

In the "learn" mode, the *red* LED will pulse every time the user presses the button, and the time of the tap will be recorded. Once the user is done tapping (i.e., the user

presses the mode button again), the BPM should be calculated and the new value should be reflected in the "play" mode. At least 4 taps are required to create a valid BPM measurement.

To make your metronome a Thing, your board will expose the following rest endpoints (using *mbed-client* M2M objects) under a **Frequency** device (**3318**):

- **Set Point Value (5900):** The current metronome BPM. This should be readable through GET, and writable through PUT.
- **Min/Max Measured Value (5601/5602):** The highest and lowest BPM the user has set the metronome to (not 0). These should be readable.
- **Reset Min/Max Measured Value (5605):** Reset the max and min to their undefined values through a POST request.
- **Units (5701):** To be fully compliant, your Thing should expose the units of measurement for its generic sensors. This should be readable as "BPM".

For more information on LwM2M and device/resource codes, refer to http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html

Getting Started

Import a new project in the mbed online compiler. Click on "import from URL" and enter the following: https://github.com/wquist/metronome. The project should import the required libraries and be buildable. Look at the headers and sources located in the top level of the project.

The header/source for *frdm_client* contains the code to authenticate and connect your board to the ARM Device Connector. Nothing in this class should need to be changed; it is complete, and already set up in the main function.

The header for *metronome* contains the definitions for the functionality of the metronome itself. You will need to write the implementation of this class. The metronome should *start_timing()* in the "learn" mode, and *stop_playing()* upon return to the "play" mode. While timing, each *tap()* should record the current time (through the use of mbed's *Timer* object). The *m_beats* array can be used to implement a running average – only the most recent *N* taps are recorded.

The file *main.cpp* contains most of the functionality involving the LEDs, buttons, and IoT capabilities. Near the top of this file, you will notice a *#define IOT_ENABLED*. Commenting this out will allow you to develop the metronome functionality without being required to connect with an Ethernet cable. The LED and button objects, as well as their callbacks, have been predefined. Note that the button callback functions are somewhat limited - network requests cannot be performed inside of them. For example, updating the BPM in a GET request (with *set_value)* must be done in the main thread, not inside the button callback.

Some utility functions are also provided in *utils.hpp*. *utils::pulse* allows you to turn an LED on and off in a short flash. *utils::entropy_seed* generates a more random seed for *srand*. This is necessary for randomizing network parameters inside *mbed-client*.

With the starting code and *IOT_ENABLED*, the blue LED will be active upon startup, and turn off once your board has connected successfully.

Once completed, you should be able to interact with your connected board through the mbed Device Connector, under the "API Console" tab. Remember to get your unique *security.h* header from the "Security Credentials" tab. Review the Lecture 10 slides for more information.

Part Two —Web Application & Introduction to Node.js

Background

The Device Connector exposes your embedded REST API to the Internet. However, the Connector works differently compared to a normal website's API – rather than sending the results of a request in the response, the Device Connector creates its own REST request directed back at your endpoint with the data.

Handling this style of request/response solely on a client-side site is difficult, especially considering the limitations of JavaScript cross-domain communication. Rather, we will use Node.js to create a backend/server along with our front-end webpage. This also allows us to use *mbed-connector-api-node*, ARM's library for communicating with the Device Connector.

Requirements

For this part of the lab, you will create a web application to interact with your embedded service, using Node.js and HTML5. The application will allow you to easily interact with your REST endpoints.

Therefore, your web application should allow you to:

- Be notified when the BPM on the board changes by automatically updating the BPM text.
- Set the BPM on the board.
- View/Reset the minimum and maximum values recorded by the board (the values should be updated automatically similar to the BPM value).

In addition, the application will provide the following metronome features:

- Play a clicking sound at the speed specified on the board.
- Show a simple animation by blinking the background at the specified BPM, similar to how the LED flashes on the microcontroller.

Getting Started

Download and install Node.js from https://nodejs.org/en/ (this will install the *npm* and *node* command line utilities). Download the part-two archive and decompress it. Navigate to the resulting folder in a command line and run *npm install*. Once the command completes, run *node app.js* and navigate to http://localhost:8080. You should see a basic webpage.

Before beginning, you should edit the *.env* file in the project directory (it may not be visible in your file browser, as it starts with a dot). Set the *ACCESS_KEY* line to your Device Connector access key, which can be created/viewed under the "Access Keys" tab at https://connector.mbed.com/. Set the *ENDPOINT* line to your board unique id, which can be viewed under the Device Connector's "API Console" tab.

The two main files of interest are *app.js* and *views/index.html*. In a Node application, *app.js* defines the server-side, and the HTML "views" are the client-side web pages.

Client-Side

Looking in *index.html*, you will see the majority of the page design is already complete. Towards the bottom of the file, you will see the JavaScript for communicating with the server. This is where all of your client-side code will be written. The action of the "GET" button has already been defined.

The two halves of this Node application communicate through *WebSockets*, which basically allow you to send messages between the client and server. The *emit(message, data)* function is used to send a message to the server, and the *on(message, callback(data))* function is used to handle messages received from the server.

For animating and playing sounds, look into the *setInterval* function and the *Audio* HTML5 object (http://www.w3schools.com/html/html5_audio.asp), respectively.

Server-Side

In *app.js*, you will see the code for creating the corresponding server. This half uses ARM's Node API to communicate with the device connector. The main point of interest in this file is the *listen* function (defined towards the bottom of the file), which specifies two callbacks: one for a new socket connection (when your webpage is loaded and connects to the server with WebSockets) and one for asynchronous notifications from the Device Connector.

Like the front-end, the server uses socket *on* and *emit* functions to communicate. In the example "get-bpm" socket action, the server requests GET data from the mbed API using *getResourceValue*. The API also provides similar functions for the other REST methods. Refer to https://github.com/ARMmbed/mbed-connector-api-node/blob/master/docs/API.md for more information.

Listening to sockets is useful for knowing when the user requests an action. However, for this lab, the application should also be able to automatically update the BPM value as it changes on the board. This is what the second callback in *listen* is used for: investigate the *putResourceSubscription* function and the *notification* object in the mbed API.

Submission Details

For part one, submit a *.zip* file of all sources you modified. For part two, submit a *.zip* of the project directory **after** deleting the *node_modules* folder. If you delete the *node_modules* folder and want to run your application again, you will have to run *npm install* first.