Demo example in Java for Android

Introduction

AndroidExample01 is a simple demonstrational example of using the SRM module with a LED light, button and buzzer written in the programming language Java for the Android platform.

When developing new applications for SRM modules for the Android platform it is helpful to see code examples of how to interact with their REST API. This demonstration is built around the scenario where an application developer is given an unconfigured device powered by the SRM module. The device contains a SRM module without a boot configuration script, but has a LED light attached to a GPIO port, a button to another GPIO port, and a buzzer/speaker to a PWM port. To use the application first enter all needed parameters and click on the "INIT" button to remotely configure the SRM module and prepare controllers for easy interaction with the LED light, button, and buzzer. Afterwards you can use these controllers to turn the LED light on and off, read the button state, and turn the buzzer on and off. Although this is a simple example, it demonstrates the power of SRM modules and is easily understand and extend.

The purpose of this example is not to show the most simple code for interacting with the SRM module, but to provide a clean example with security aspects and best programming practices in mind. Each function is implemented separately to make the code more understandable. It implements a pooling strategy, a more power-efficient pushing strategy will be presented in SRMDataPlatform.

List of main files for this delivery:

- *"app/src/main/java/srm/srmexample01/MainActivity.java" -- Demo example for Android*
- *"app/src/main/java/srm/srmlib/" -- Supporting library*

Other files and resources:

- *"app/src/main/res/layout/activity_main.xml" -- Layout of demo example for Android*
- *"app/src/main/res/drawable/" -- Shapes and icons*
- *"app/src/main/res/mipmap-*/" -- Launcher icon*
- *"app/src/main/res/values/*.xml" -- Constants*
- *"app/src/main/AndroidManifest.xml" -- Application manifest*
- *"*gradle*" -- Build scripts for Android Studio*
- *"README.pdf" -- Getting started instructions*

Setup

Development environment for this demonstration is assumed to be:

- *Ubuntu 16.04 LTS*
- *Android Studio 2.3.1*
- *Android 6.0 SDK (API 23)*

First you need to download and extract *Android Studio*, then install *Android SDK* under menu option *Tools/Android/SDK Manager*. Afterwards you can build and install this Android example on your Android device as usual.

For higher HTTPS security levels place the certificate contents of your CA as string into *"app/src/main/java/srm/srmlib/CaCertPem.java"*.

Hardware requirements:

- an unconfigured SRM module
- a LED light attached to GPIO port 25 *(/phy/gpio/25)*
- a button attached to GPIO port 14 *(/phy/gpio/14)*
- a buzzer/speaker attached to PWM port 1 *(/phy/pwm/1)*

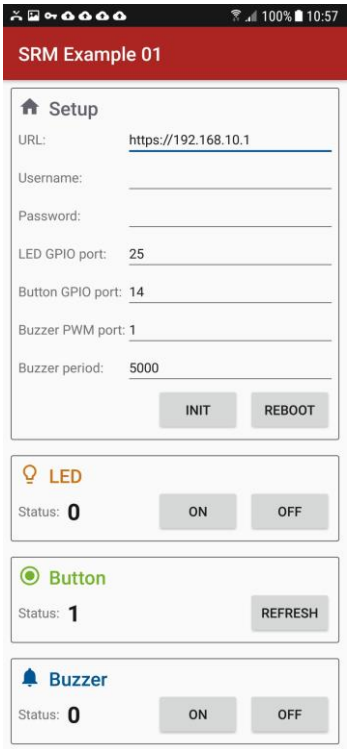
Power on the SRM module and connect over WiFi to the access point it exposes. When connected, the REST API interface should be accessible over:

https://192.168.10.1/ui

Make sure the SRM module is in an unconfigured state by rebooting it before running this example.

Usage

When this example is installed on an Android phone it appears as the "SRM Example 01" application. First you need to enter the parameters for your SRM module and click on button "INIT". When the SRM module is successfully initialized, all buttons become enabled and you can remotely control your SRM-enabled device.



Code description

Main code is contained within *"app/src/main/java/srm/srmexample01/MainActivity.java"* and it uses the supporting library *"app/src/main/java/srm/srmlib/"*.

We initialize the application in the method *onCreate()* where corresponding click handlers for all buttons are registered. Each handler is represented by an *AsyncTask*, which is the preferred programming practice for handling multi-threaded execution of tasks. More precisely, network operations should not be performed in the UI thread and UI updates should not be performed in background threads. First the *onPreExecute()* retrieves needed data from the UI in UI thread, then *doInBackground()* executes the network REST API calls in a background thread, and finally *onPostExecute()* updates the UI with results in UI thread. Functionality for each sensor and actuator was intentionally implemented separately to make the code more understandable. To simplify interaction with SRM modules and handle HTTPS security levels we prepare *SRMClient* objects from the supporting library where needed.

Initialization of the SRM module (in class *SetupTask*) begins by preparing the root URL with provided username and password for accessing the SRM module and a *SRMClient* object to simplify the initial interaction.

```
URL url = SRMClient.urlBuilder(this.url, null, this.username, this.password, null,
    null, null, null, null);
mClient = new SRMClient(url, this.httpsCheck, null, null, null, this.verbose);
```

Next the GPIO port for the **LED light** gets allocated and configured with direction out. To simplify further interaction with the LED light, we also prepare a *SRMClient* object with the direct REST API link to the LED light state *(https://192.168.10.1/phy/gpio/25/value)*

```
mClient.post("/phy/gpio/alloc", String.valueOf(ledGpio));
data = "{\"dir\": \"out\", \"mode\": \"floating\", \"irq\": \"none\",
    \"debouncing\": 0}"
mClient.put(String.format("/phy/gpio/%d/cfg/value", ledGpio), data);


URL ledUrl = SRMClient.urlBuilder(url, null, username, password, null, null,
    String.format("/phy/gpio/%d/value", ledGpio), null, null);
mLed = new SRMClient(ledUrl, httpsCheck, null, null, null, verbose);
```

Next the GPIO port for the **button** gets allocated and configured with direction in. To simplify further interaction with the LED light, we also prepare a *SRMClient* object with the direct REST API link to the button state *(https://192.168.10.1/phy/gpio/14/value)*.

```
mClient.post("/phy/gpio/alloc", String.valueOf(buttonGpio));
data = "{\"dir\": \"in\", \"mode\": \"floating\", \"irq\": \"none\",
    \"debouncing\": 0}";
mClient.put(String.format("/phy/gpio/%d/cfg/value", buttonGpio), data);


URL buttonUrl = SRMClient.urlBuilder(url, null, username, password, null, null,
    String.format("/phy/gpio/%d/value", buttonGpio), null, null);
mButton = new SRMClient(buttonUrl, httpsCheck, null, null, null, verbose);
```

Next the PWM port for the **buzzer/speaker** gets allocated and configured. To simplify further interaction with the buzzer, we also prepare a *SRMClient* object *(https://192.168.10.1/phy/pwm/1/value)*.

```
mClient.post("/phy/pwm/alloc", String.valueOf(buzzerPwm));
data = String.format("{\"mode\": \"pulsewidth\", \"period\": \"%d\"}", buzzerPeriod);
mClient.put(String.format("/phy/pwm/%d/cfg/value", buzzerPwm), data);
```

```
URL buzzerUrl = SRMClient.urlBuilder(url, null, username, password, null, null,
    String.format("/phy/pwm/%d/value", buzzerPwm), null, null);
mBuzzer = new SRMClient(buzzerUrl, httpsCheck, null, null, null, verbose);
```

Afterwards you can use these controllers to check **LED light** state or turn it on and off (in class *LedTask*):

```
data = mLed.get(null).content;
mLed.put(null, "1");
mLed.put(null, "0");
```

Or read the **button** state (in class *ButtonTask*):

```
data = mButton.get(null).content;
```

Or turn the **buzzer** on and off (in class *BuzzerTask*):

```
mBuzzer.put(null, "300");
mBuzzer.put(null, "0");
```

Rest of the code handles the user interface initialization and updates.