

GadgetBridge, Bluetooth Learning And Simulation Demo

A demonstration of the Bluetooth Low Energy (BLE GATT) device learning and simulation on Android.

BLE GATT is an acronym for Bluetooth Low Energy Generic Attribute Protocol. You can learn more at bluetooth.com.

This demo uses the [GadgetBridge](#) open source application written for Bluetooth fitness bracelets. It supports Xiaomi Mi Band, Pebble, Amazfit and other devices.

Development and testing of such application involves physical manipulation with the the bracelet for some scenarios which is usually a major obstacle in development and (especially automated) testing.

There are 3 ways how to experience this demo:

1. **Out-of-box Simulation With Mobile Center**

Use the ready-made simulation model of *Xiaomi Mi Band 2* with minimal prerequisites (Mobile Center and a smartphone with Android 5.0+) to demonstrate simulation of the fitness bracelet. You don't need the bracelet, it is being simulated.

2. **Out-of-box Simulation Without Mobile Center**

Use the ready-made simulation model of *Xiaomi Mi Band 2* with minimal prerequisites (ADB and a smartphone with Android 5.0+) to demonstrate simulation of the fitness bracelet.

3. **Learning The Simulation Models**

Walk through the learning, editing and running the simulation. You need the actual *Xiaomi Mi Band 2* fitness bracelet to record its behavior.

Out-of-Box Simulation With Mobile Center

Lets follow this high level workflow for a quick simulation demonstration without the physical device:

1. Downloading the GadgetBridge application to Mobile Center
2. Installing Packaged Application To Mobile Phone
3. Simulating the bracelet to test the application without physical access to the device

Prerequisites

- Mobile Center 3.20 or newer with SV Lab integration enabled for your mobile phone (See Mobile Center documentation).
- [Java JDK 1.7 or newer](#)
- A smartphone with Android 5.0 Lollipop or newer with [USB debugging enabled](#)

Step 1: Downloading The GadgetBridge Application To Mobile Center

The GadgetBridge APK is distributed via the f-droid appstore. To [download the APK from](#), pick the *Download APK* option (rather than *DOWNLOAD F-DROID installer*).

The current version of APK was [nodomain.freemyorgadget.gadgetbridge_148.apk](#) at the time of writing this documentation but feel free to try a newer one.

Download the file to your disk and upload it to Mobile Center.

Step 2: Installing Packaged Application To Mobile Phone

Open you mobile phone in Mobile Center, click the 'Application Interactions' icon and install the packaged version of GadgetBridge application. Close the phone.

Step 3: Simulation

Mobile Center 3.20 user interface does not have the functionality to start the simulation but you can leverage the `sv-capture` tool to launch the simulation of chosen scenario.

There are simulation models bundled with the demo containing four scenarios:

- *'pair'*
describing the communication when the GadgetBridge app is run for first time and it has to be paired with bracelet
- *'connect'*
describing the communication when the GadgetBridge app is connecting to the bracelet whenever it disconnects (i.e. after app restart)
- *'syncData'*
modeling synchronization of past 24 hours of activity data from the bracelet using synthetic data generated in the model
- *'activity'*
simulating user walking with the bracelet

First simulate the *'pair'* scenario when the GadgetBridge is not yet connected to the bracelet. Provide your Mobile Center server URL, access key and mobile phone name with `-mn` argument or mobile phone ID with `-mi` parameter. The `-f NONE` argument allows us to misuse the sv-capture tool for launching simulation by disabling the default behavior of the tool which forces services to forwarding mode by default in order to learn some real responses. The `-as` argument selects the application scenario to run:

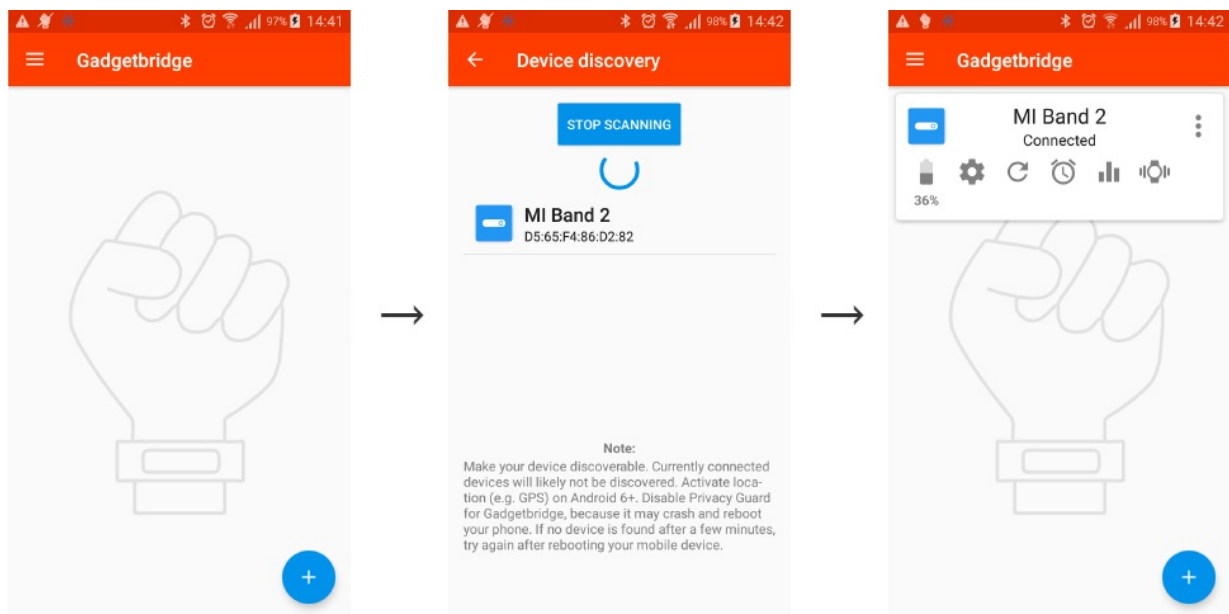
```
../../bin/sv-capture.sh -ms http://localhost:8080 -ma "client=oauth2-..." \  
                        -mn SM-G800F \  
                        -lab ./src/main/resources/sv-lab.json \  
                        -vsl ./src/main/resources/demo \  
                        -f NONE -as pair
```

In Windows:

```
..\..\bin\sv-capture.bat -ms http://localhost:8080 -ma "client=oauth2-..." ^  
                        -mn SM-G800F ^  
                        -lab ./src/main/resources/sv-lab.json ^  
                        -vsl ./src/main/resources/demo ^  
                        -f NONE -as pair
```

```
...  
Press <Enter> to quit simulation.  
12:16:27 INFO [SV_RUNLOG] Lab transition NOT_DEPLOYED -> INITIALIZED has  
succeeded.  
12:16:27 INFO [SV_RUNLOG] Lab transition INITIALIZED -> STARTED has succeeded.  
12:16:27 INFO [SV_RUNLOG] Scenario has started: pair
```

Check the console for the `Scenario has started: pair` message, it can take a while first time you run the demo due to compilation of simulation models. Then launch the GadgetBridge application on you smartphone. Tap the plus symbol to discover and add a new bracelet. The app will detect the simulated bracelet. Tap it to connect:



Open the bracelet detailed view tapping the bar graph icon and an empty graph appears. The application has not yet synchronized with the bracelet data.

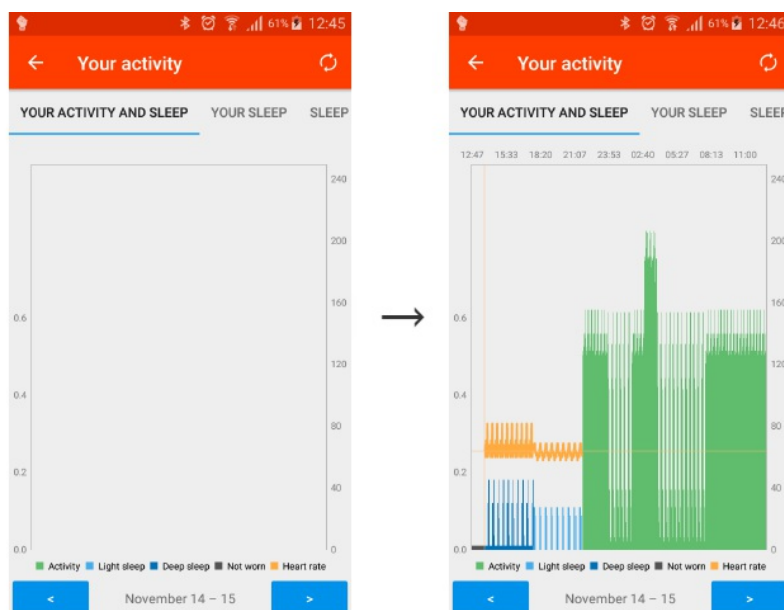
Stop the *'pair'* scenario simulation by pressing <Enter> in the console where you've run it and simulate the *'syncData'* scenario:

```
../../../../bin/sv-capture.sh -ms http://localhost:8080 -ma "client=oauth2-..." \
-mn SM-G800F \
-lab ./src/main/resources/sv-lab.json \
-vsl ./src/main/resources/demo \
-f NONE -as syncData
```

In Windows:

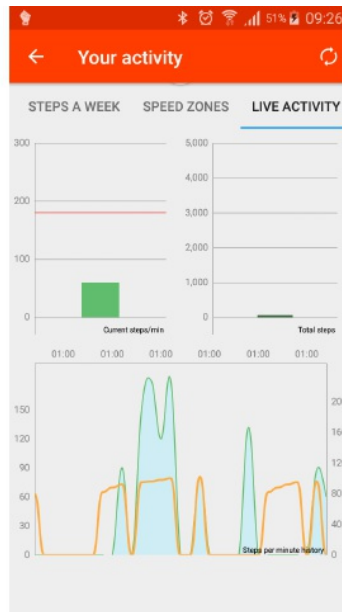
```
..\..\bin\sv-capture.bat -ms http://localhost:8080 -ma "client=oauth2-..." ^
-mn SM-G800F ^
-lab ./src/main/resources/sv-lab.json ^
-vsl ./src/main/resources/demo ^
-f NONE -as syncData
```

Now tap the synchronization icon in the top right corner and watch the 24 hours of synthetic data generated by SV appear in the graph:

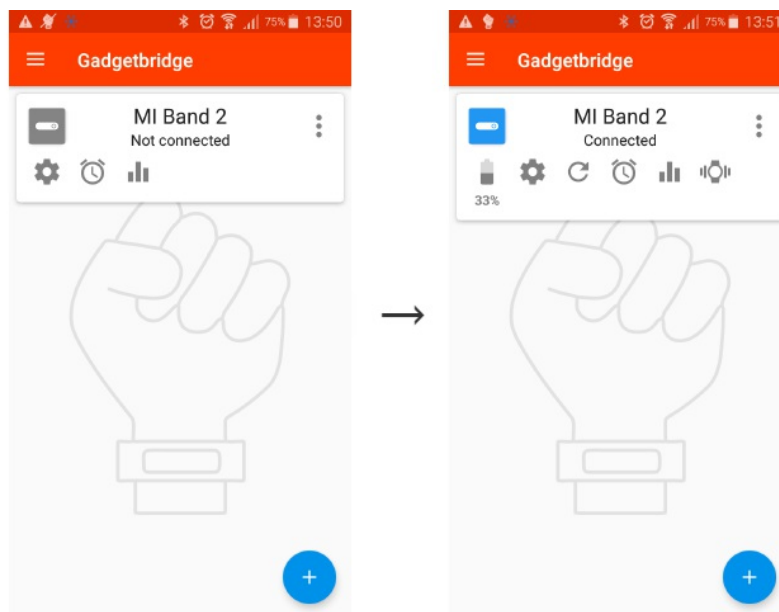


Now stop the simulation by pressing <Enter> in the console and simulate the *'activity'* scenario in a similar way.

Scroll to the rightmost *LIVE ACTIVITY* tab in the mobile application. The graphs will start to move and show the user walking with the bracelet around the office:



Run the *'connect'* scenario in a similar way whenever the bracelet gets disconnected from the application (i.e. when the application is restarted), then tap the gray 'Mi Band 2' card to reconnect:



Out-of-Box Simulation Without Mobile Center

This workflow is a little bit more complex than using Mobile Center since you will have to perform the configuration and app instrumentation steps yourselves:

1. Downloading the GadgetBridge application
2. Configuring the smartphone to talk to SV Lab
3. Packaging the GadgetBridge Android application using code that enables virtualization of smart devices connected over BLE GATT and installing the app on the phone
4. Simulating the bracelet to test the application without physical access to the device

Prerequisites

- [Java JDK 1.7 or newer](#)
- Either Mobile Center or the ADB tool to upload APKs to the smartphone. The ADB tool can be installed either with the [Android Studio](#) or the light-weight [Minimal ADB And Fastboot](#) distribution
- A smartphone with Android 5.0 Lollipop or newer with [USB debugging enabled](#)
- A *SV Lab Server* running on local host (launched with `server-start.sh|.bat`)

Step 1: Downloading The GadgetBridge Application

The GadgetBridge APK is distributed via the f-droid appstore. To [download the APK from](#), pick the *Download APK* option (rather than *DOWNLOAD F-DROID installer*).

The current version of APK was [nodomain.freeyourgadget.gadgetbridge_148.apk](#) at the time of writing this documentation but feel free to try a newer one.

Save the file to the demo directory and rename it to `gadgetbridge.apk` .

Step 2: Android Connector Configuration

Now you need to configure the Android phone to connect to your PC running the simulation. There has to be a virtual lab running to test the connection. You can for example use the *sv-capture* tool to bring it up:

```
../../bin/sv-capture.sh -u https://localhost:8445/api/ -c MOBILE_API \
                        -p 9009 -o ./sandbox
```

In Windows:

```
..\..\bin\sv-capture.bat -u https://localhost:8445/api/ -c MOBILE_API ^
                        -p 9009 -o ./sandbox
```

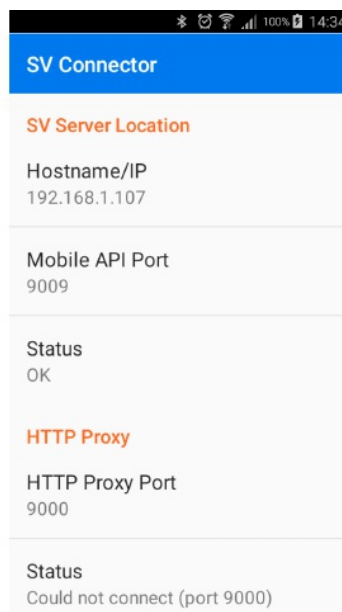
Now install the *SV Connector* application on the phone. If you work with a smartphone connected to Mobile Center, upload the `../../bin/sv-android-connector-5.1.1.apk` file and install it to the phone using Mobile Center. Alternatively, use the ADB tool from Android SDK to install the application on a phone connected with USB cable:

```
adb install ../../bin/sv-android-connector-5.1.1.apk
```

In Windows:

```
adb install ..\..\bin\sv-android-connector-5.1.1.apk
```

Launch the *SV Connector* application on your phone and enter the IP address or host name of your PC. The application tests the connection to running lab and it should show the OK status:



Now **stop the virtual lab pressing the <Enter> key** in the console window where you've run the *sv-capture* tool.

Step 3: Android Application Virtualization

The SV Lab needs to hook-in to the Android application to be tested so it can learn and simulate the Bluetooth API calls used to communicate with the bracelet. It is performed by packaging the mobile application with the Mobile Center's APKInfuser tool using code that enables the virtualization of smart devices connected over BLE GATT. While the Mobile Center performs this step automatically upon APK upload, you have to apply the instrumentation yourself when you have chosen not to use Mobile Center.

Get the ApkInfuser tool from the [Microfocus AppDelivery Marketplace](#). Download the '*APKInfuser - Android app packager 3.10*' package and extract the ZIP file and copy contents of the **Android Tools** archive subdirectory to the SV Lab **bin** directory.

Then execute following command in demo directory:

```
../../bin/MCAndroidEnabler.sh -custom ../../bin/sv-ble.json \  
../../bin/sv-ble.dex -signdebug gadgetbridge.apk
```

In Windows:

```
..\..\bin\MCAndroidEnabler.bat -custom ..\..\bin\sv-ble.json ^  
..\..\bin\sv-ble.dex -signdebug gadgetbridge.apk
```

Now you have the packaged application in the **gadgetbridge.apk.signed.debug.apk** file. Install it to the phone via Mobile Center or using the ADB tool:

```
adb uninstall nodomain.freeyourgadget.gadgetbridge  
adb install gadgetbridge.apk.signed.debug.apk
```

The first of the two commands is only needed when you have had the GadgetBridge application installed on your phone. Note that it will remove any application data and settings.

Step 4: Simulation

There are simulation models bundled with the demo containing four scenarios:

- *'pair'*
describing the communication when the GadgetBridge app is run for first time and it has to be paired with bracelet
- *'connect'*
describing the communication when the GadgetBridge app is connecting to the bracelet whenever it disconnects (i.e. after app restart)
- *'syncData'*
modeling synchronization of past 24 hours of activity data from the bracelet using synthetic data generated in the model
- *'activity'*
simulating user walking with the bracelet

First simulate the *'pair'* scenario when the GadgetBridge is not yet connected to the bracelet:

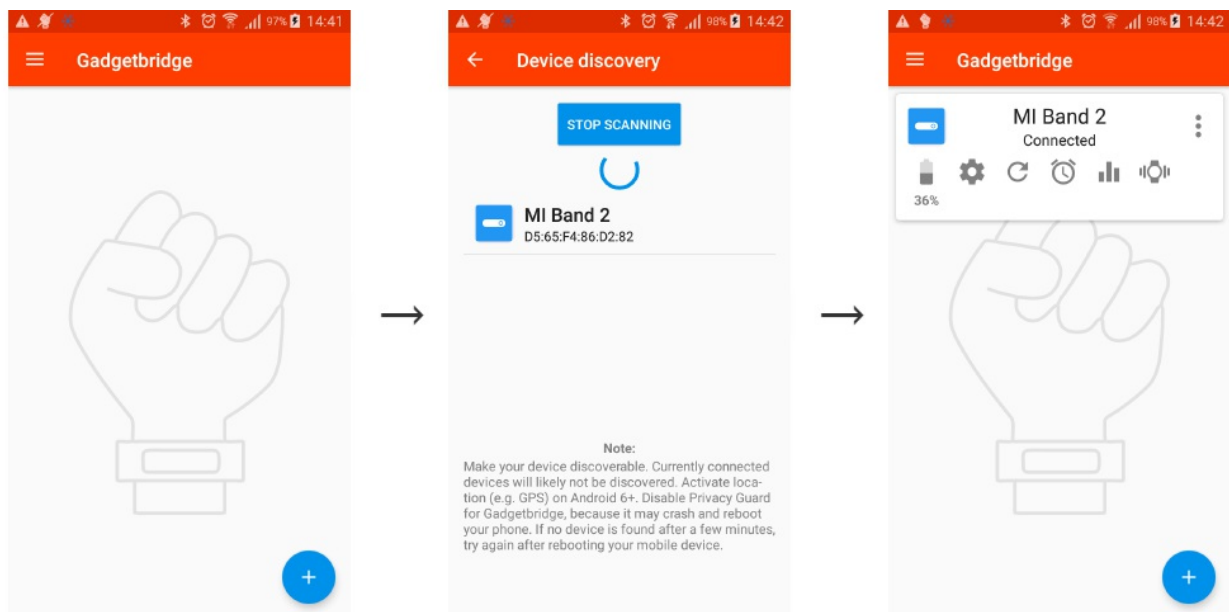
```
./run-with-server.sh pair
```

In Windows:

```
run-with-server.bat pair
```

```
...
Press <Enter> to quit simulation.
12:16:27 INFO [SV_RUNLOG] Lab transition NOT_DEPLOYED -> INITIALIZED has
succeeded.
12:16:27 INFO [SV_RUNLOG] Lab transition INITIALIZED -> STARTED has succeeded.
12:16:27 INFO [SV_RUNLOG] Scenario has started: pair
```

Check the console for the **Scenario has started: pair** message, it can take a while first time you run the demo due to compilation of simulation models. Then launch the GadgetBridge application on you smartphone. Tap the plus symbol to discover and add a new bracelet. The app will detect the simulated bracelet. Tap it to connect:



Open the bracelet detailed view tapping the bar graph icon and an empty graph appears. The application has not yet synchronized with the bracelet data.

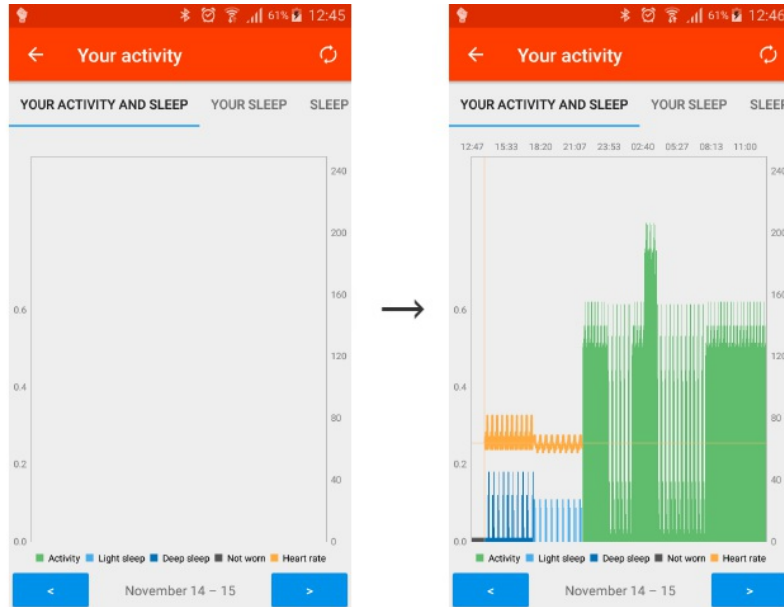
Stop the *'pair'* scenario simulation by pressing <Enter> in the console where you've run it and simulate the *'syncData'* scenario:

```
./run-with-server.sh syncData
```

In Windows:

```
run-with-server.bat syncData
```

Now tap the synchronization icon in the top right corner and watch the 24 hours of synthetic data generated by SV appear in the graph:



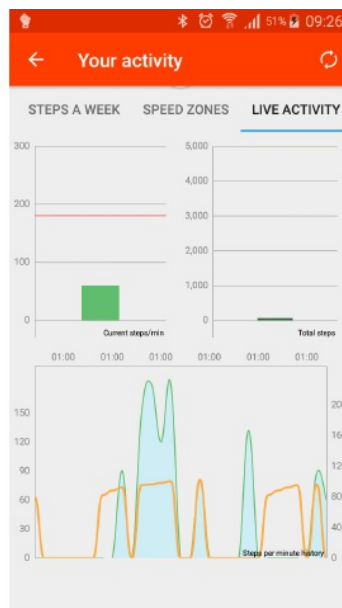
Now stop the simulation by pressing <Enter> in the console and simulate the 'activity' scenario:

```
./run-with-server.sh activity
```

In Windows:

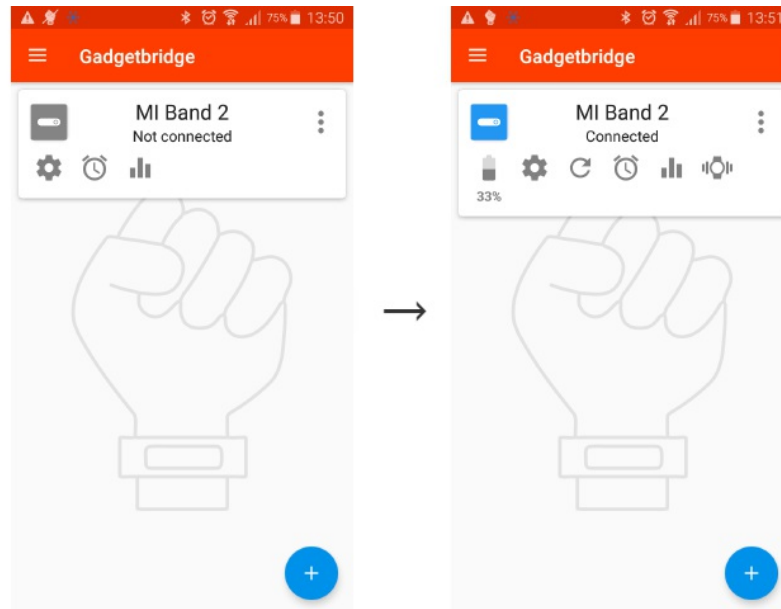
```
run-with-server.bat activity
```

Scroll to the rightmost *LIVE ACTIVITY* tab in the mobile application. The graphs will start to move and show the user walking with the bracelet around the office:



Use the 'connect' scenario in a similar way whenever the bracelet gets disconnected from the application (i.e. when the application is restarted). Launch the scenario and tap the gray 'Mi Band 2'

card to reconnect:



Learning The Simulation Models

When you work with a new device or when you create new tests you have to create simulation models with new scenarios describing desired behavior of the device during test cases.

With Bluetooth devices, the most common way is to learn each scenario while interacting with real device and then add it to your simulation models.

The workflow for this demonstration is similar to the out-of-box simulation with or without Mobile Center described in previous chapters so please follow the detailed description there.

The only difference in the extra learning step performed before the simulation where you learn the simulation models as described below instead of using the models included with demo.

The Mobile Center workflow will become:

1. Downloading the GadgetBridge application to Mobile Center
2. Installing Packaged Application To Mobile Phone
3. **Learning scenarios of interaction with real *Mi Band 2* bracelet and refactoring them in a runnable simulation model**
4. Simulating the bracelet to test the application without physical access to the device

and the workflow without Mobile Center now becomes:

1. Downloading the GadgetBridge application
2. Configuring smartphone to talk to SV Lab
3. Packaging the GadgetBridge Android application using code that enables virtualization of smart devices connected over BLE GATT and installing the app on the phone
4. **Learning scenarios of interaction with real *Mi Band 2* bracelet and refactoring them in a runnable simulation model**
5. Simulating the bracelet to test the application without physical access to the device

Prerequisites

- [Java JDK 1.7 or newer](#)
- Either Mobile Center or the ADB tool to upload APKs to the smartphone. The ADB tool can be installed either with the [Android Studio](#) or the light-weight [Minimal ADB And Fastboot](#) distribution

- A smartphone with Android 5.0 Lollipop or newer with [USB debugging enabled](#)
- A *SV Lab Server* running on local host (launched with `server-start.sh|.bat`)
- Xiaomi [Mi Band 2](#) sport bracelet

Learning

Lets start learning the scenario where the GadgetBridge application is pairing with the bracelet.

When you don't use Mobile Center, start learning using the *sv-capture* tool:

```
../../bin/sv-capture.sh -u https://localhost:8445/api/ -c MOBILE_API \
                        -p 9009 -o ./sandbox
```

In Windows:

```
..\..\bin\sv-capture.bat -u https://localhost:8445/api/ -c MOBILE_API ^
                        -p 9009 -o ./sandbox
```

The tool brings up a virtual lab with virtual service ready for learning.

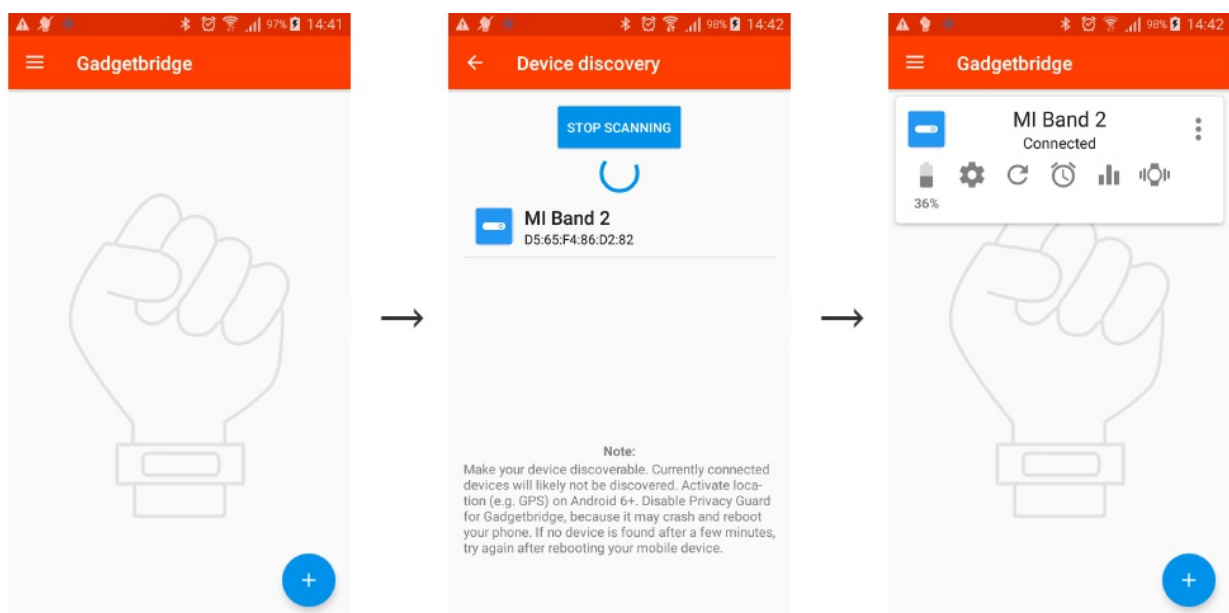
With Mobile Center, specify the connection, authentication token and mobile phone to use in place of Lab Server URL:

```
../../bin/sv-capture.sh -ms http://localhost:8080 -ma "client=oauth2-..." \
                        -mn SM-G800F -c MOBILE_API \
                        -p 9009 -o ./sandbox
```

In Windows:

```
..\..\bin\sv-capture.bat -ms http://localhost:8080 -ma "client=oauth2-..." ^
                        -mn SM-G800F -c MOBILE_API ^
                        -p 9009 -o ./sandbox
```

Now launch the GadgetBridge application on your smartphone and tap the plus button to add a new bracelet to the app. It should discover your bracelet. Tap on it to pair and connect. You can see the captured messages in the *sv-capture* tool.



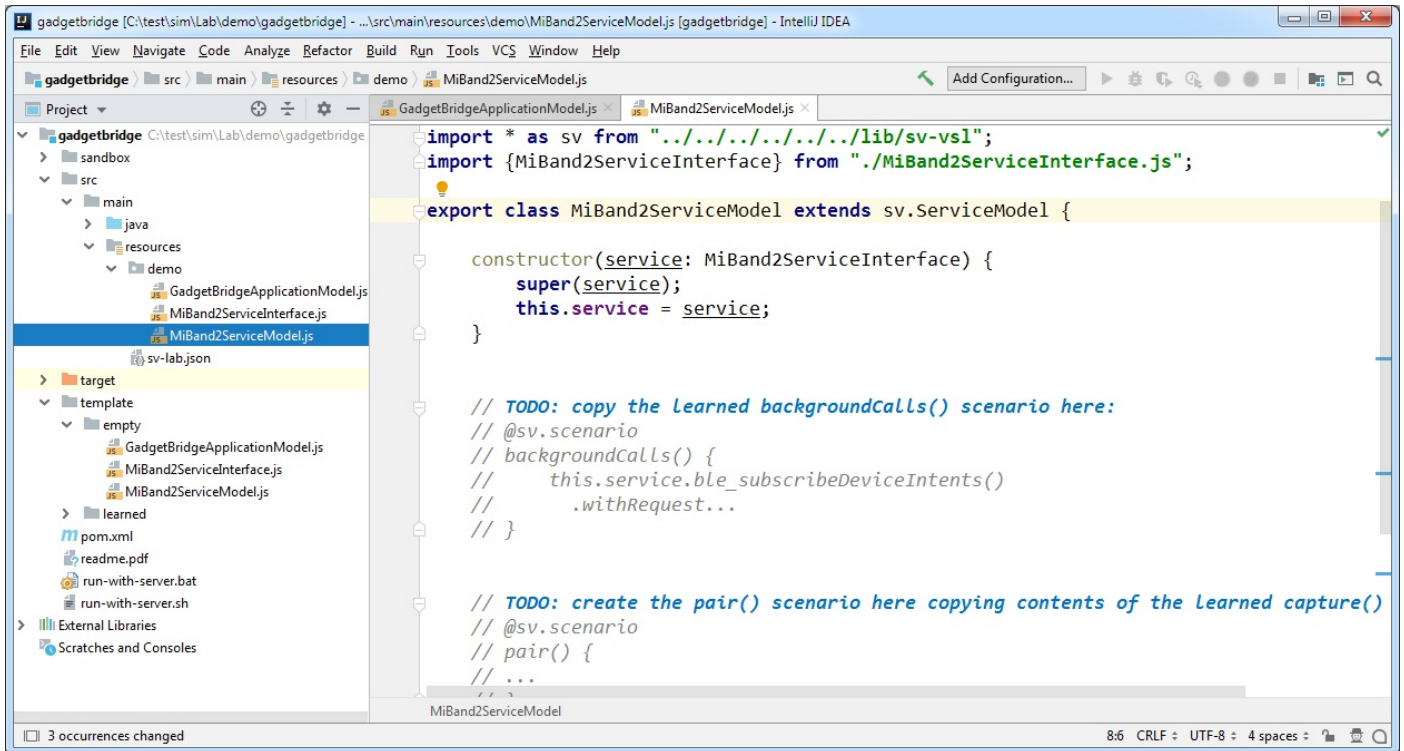
Finish the learning by pressing <Enter> in the *sv-capture* tool. The resulting VSL files containing the learned scenario are written to the sandbox directory:

```

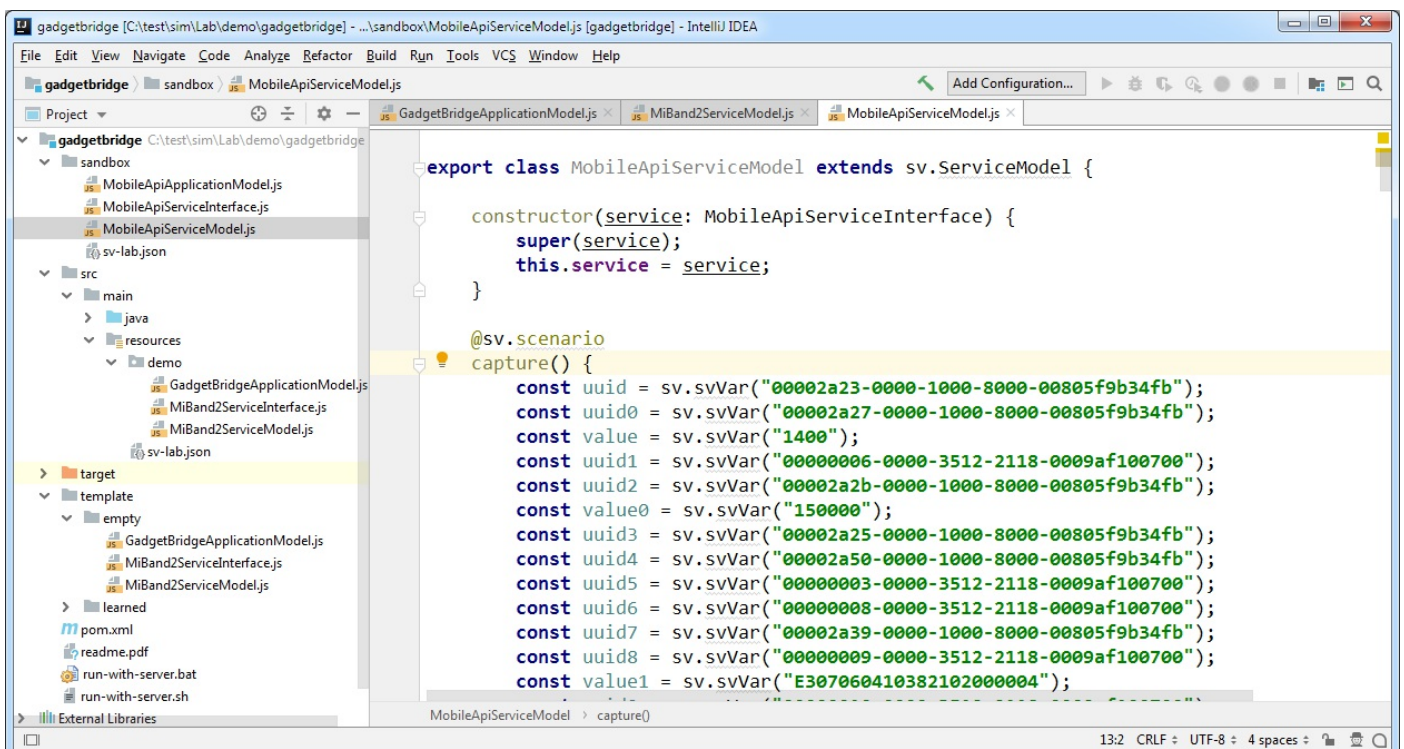
--sandbox
MobileApiApplicationModel.js
MobileApiServiceInterface.js
MobileApiServiceModel.js

```

You will now use the learned data in the simulation model. Open the demo `pom.xml` project in your IDE. Copy the 2 empty model files from the `template/empty` directory to `src/main/resources/demo`. They will be our *target models* for the refactoring:

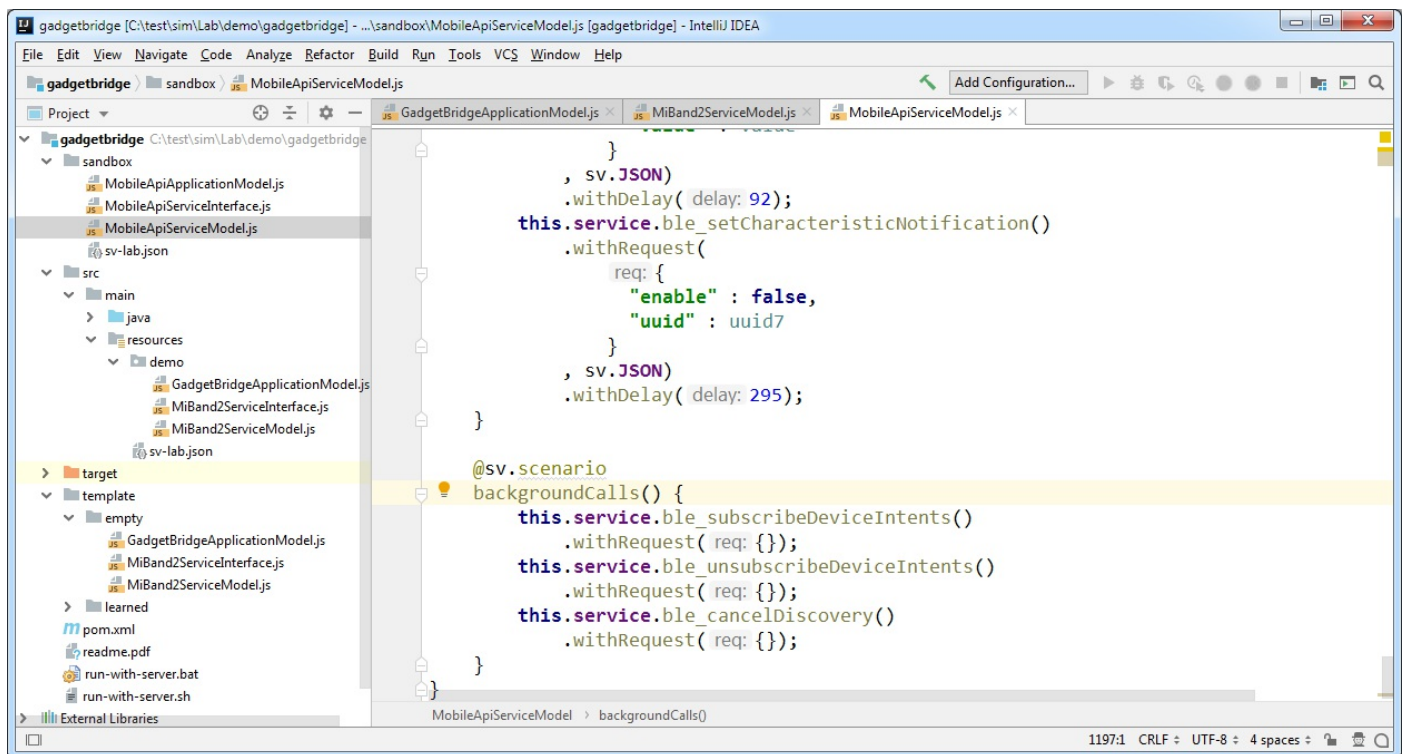


Open the learned `sandbox/MobileApiServiceModel.js` and watch the the learned scenarios. The `this.service` property represents our Bluetooth device (the bracelet).



Now, there are basically 2 threads running in the GadgetBridge application. One is periodically checking

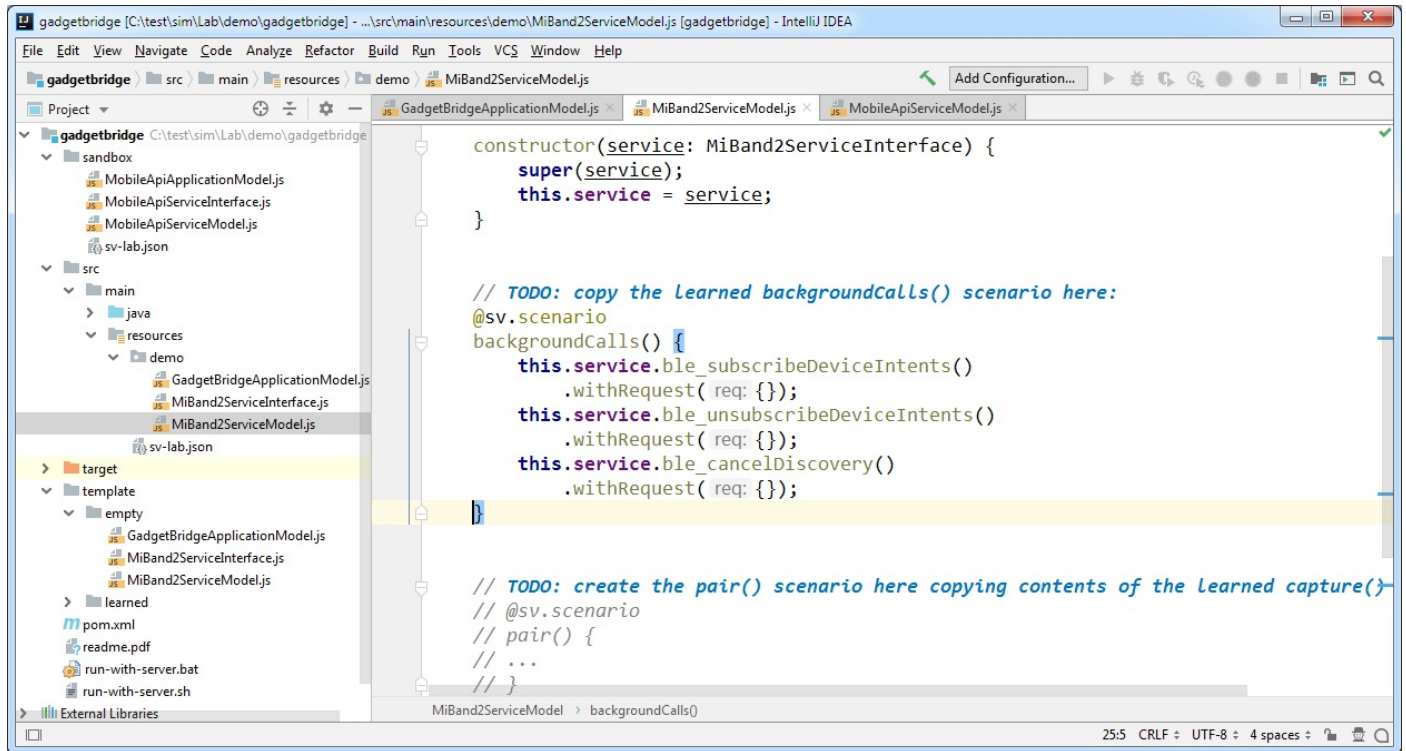
Bluetooth devices bonded to the phone and the other is talking to the bracelet according to user interaction. The *sv-capture* tool has split the calls to 2 scenarios. While the `capture()` scenario contains the main scenario data, the `backgroundCalls()` at the end of the model contains the operations which are usually called at the background:



```
}, sv.JSON)
.withDelay( delay: 92);
this.service.ble_setCharacteristicNotification()
.withRequest(
  req: {
    "enable" : false,
    "uuid" : uuid7
  }
), sv.JSON)
.withDelay( delay: 295);
}

@sv.scenario
backgroundCalls() {
  this.service.ble_subscribeDeviceIntents()
    .withRequest( req: {});
  this.service.ble_unsubscribeDeviceIntents()
    .withRequest( req: {});
  this.service.ble_cancelDiscovery()
    .withRequest( req: {});
}
```

Copy the learned background scenario to the *target* service model (`src/main/resources/demo/MiBand2ServiceModel.js`):

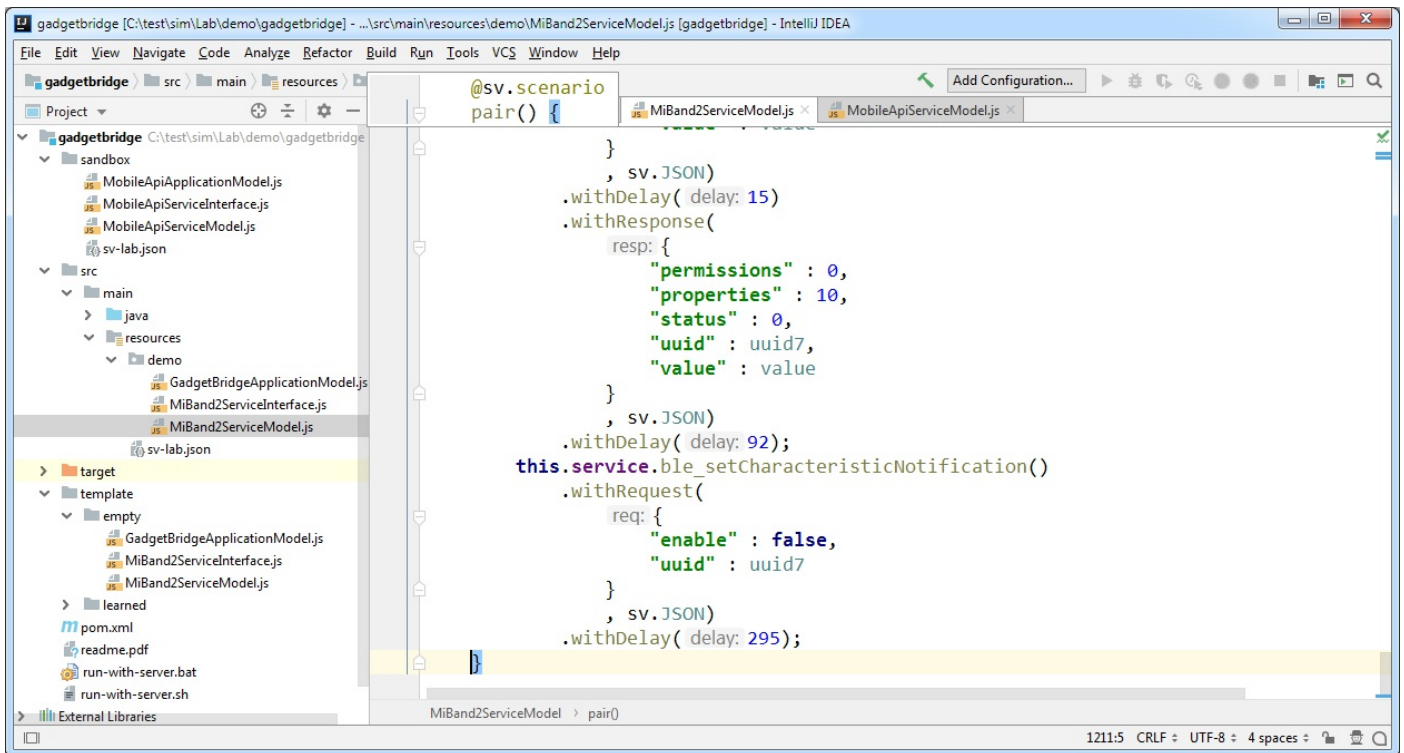


```
constructor(service: MiBand2ServiceInterface) {
  super(service);
  this.service = service;
}

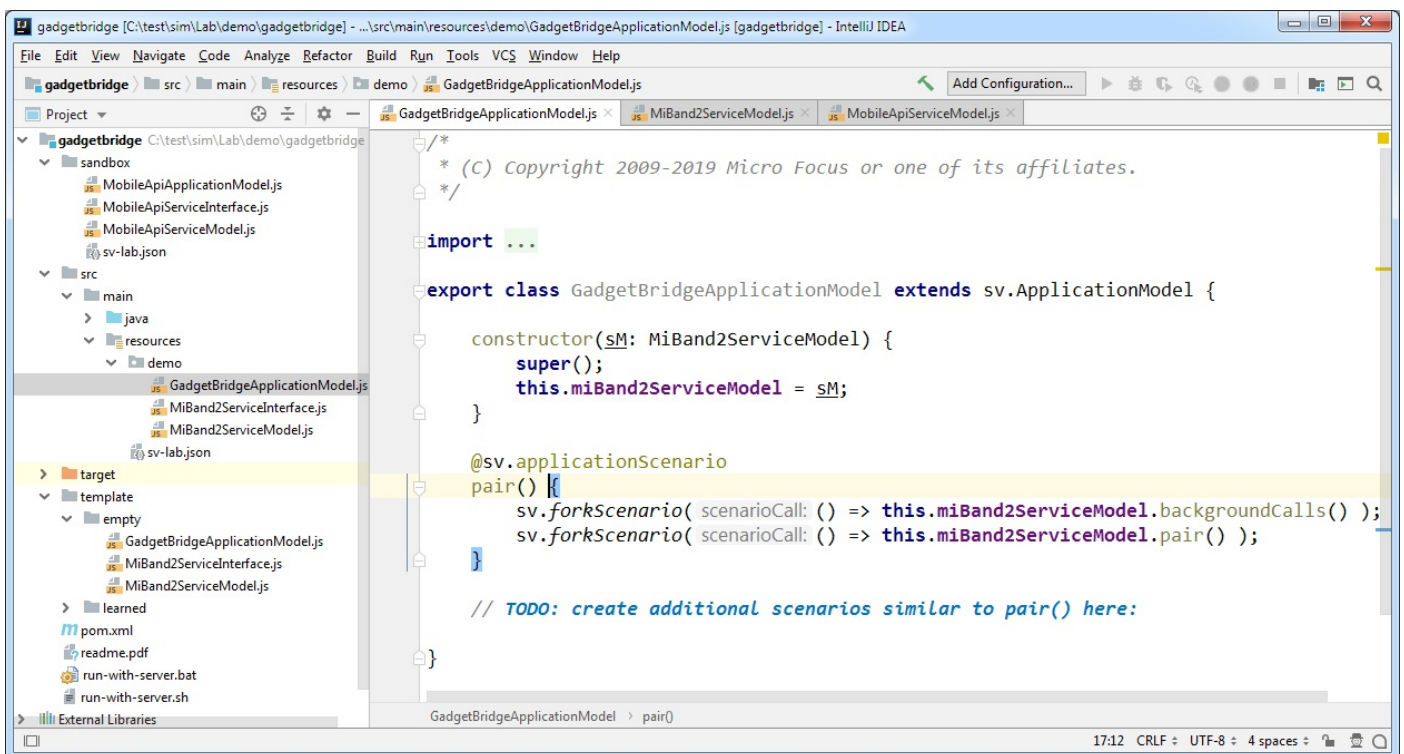
// TODO: copy the learned backgroundCalls() scenario here:
@sv.scenario
backgroundCalls() {
  this.service.ble_subscribeDeviceIntents()
    .withRequest( req: {});
  this.service.ble_unsubscribeDeviceIntents()
    .withRequest( req: {});
  this.service.ble_cancelDiscovery()
    .withRequest( req: {});
}

// TODO: create the pair() scenario here copying contents of the Learned capture()
// @sv.scenario
// pair() {
//   ...
// }
```

Now create the `pair()` scenario for the flow of Bluetooth bracelet connecting with the GadgetBridge application by copying and renaming the `capture_mobileapi...` scenario to the *target* service model:

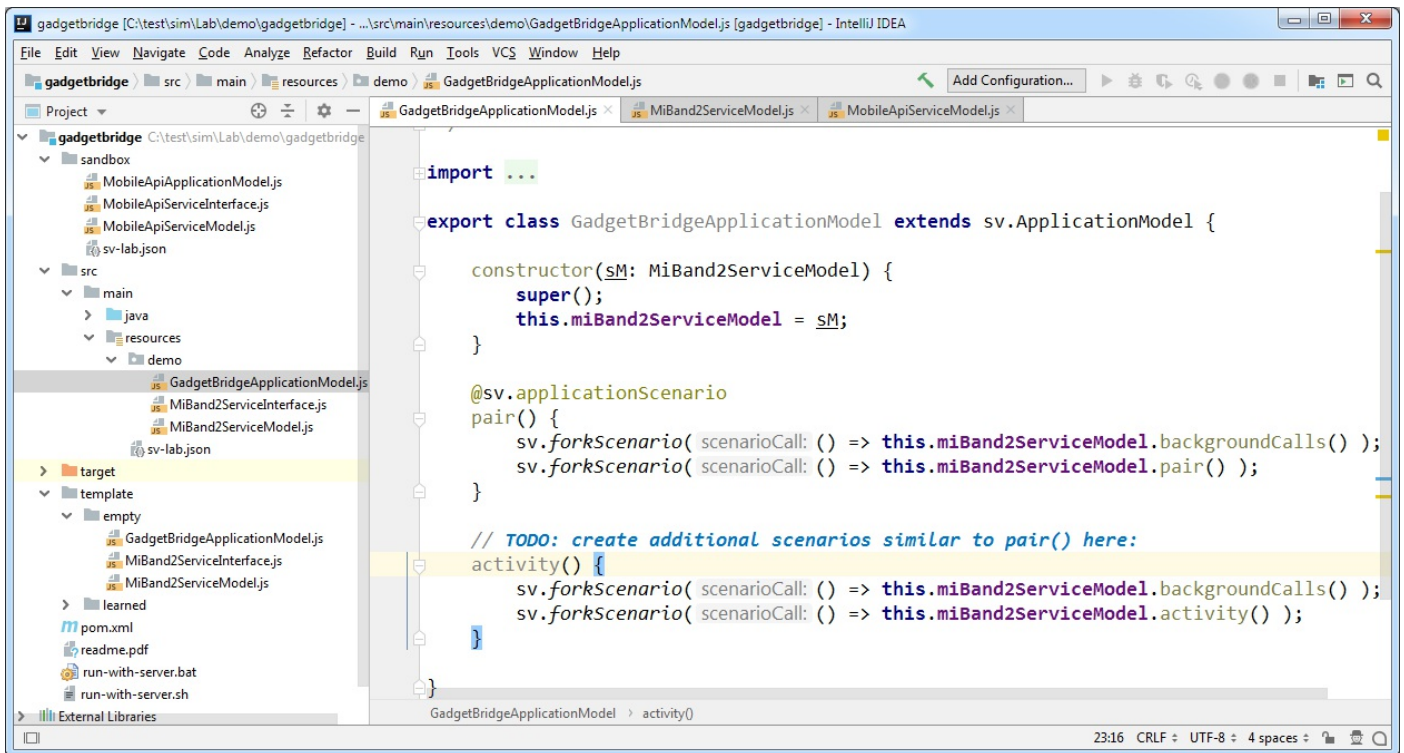


The `backgroundCalls()` and `pair()` service scenarios need to be launched in parallel. This is achieved using the `sv.forkScenario()` commands in the `src/main/resources/demo/GadgetBridgeApplicationModel.js` and the code for the `pair` scenario is already present in the template:



Now you can test simulation of the `pair` scenario you've just learned following the [previous demo](#).

Repeat the process for learning and implementing the `activity` and possibly other scenarios, creating a new flow in service model and running simultaneously with the background scenario in a corresponding application scenario like this:



To simulate the newly created scenario, follow Simulation step in previous chapter.

Source Code

```

.
|   pom.xml ..... project file for your IDE
+--src
|   |--main
|   |   |--java
|   |   |   |--demo
|   |   |   |   MiBand2.java ..... simulation runner
|   |   |--resources
|   |   |   |   sv-lab.json ..... SV Lab configuration
|   |   |   |   |--demo
|   |   |   |   |   GadgetBridgeApplicationModel.js .. 'pair', 'connect', 'syncData'
|   |   |   |   |   MiBand2ServiceInterface.js      and 'activity' scenarios
|   |   |   |   |   MiBand2ServiceModel.js
+--sandbox ..... target directory for learning
|--template
|   |--empty
|   |   GadgetBridgeApplicationModel.js ..... empty models ready to fill
|   |   MiBand2ServiceInterface.js           with learned data
|   |   MiBand2ServiceModel.js
|   |--learned
|   |   GadgetBridgeApplicationModel.js ..... backup copy of the
|   |   MiBand2ServiceInterface.js           ready-to-run scenarios
|   |   MiBand2ServiceModel.js

```