MT2511 Health Module Programming Guide

2017.04.19

Outline

- Overview
- Introduction
  - Main Class
  - Send Command Flow
  - Receive Data Flow
  - Data Format
- Appendix

Overview

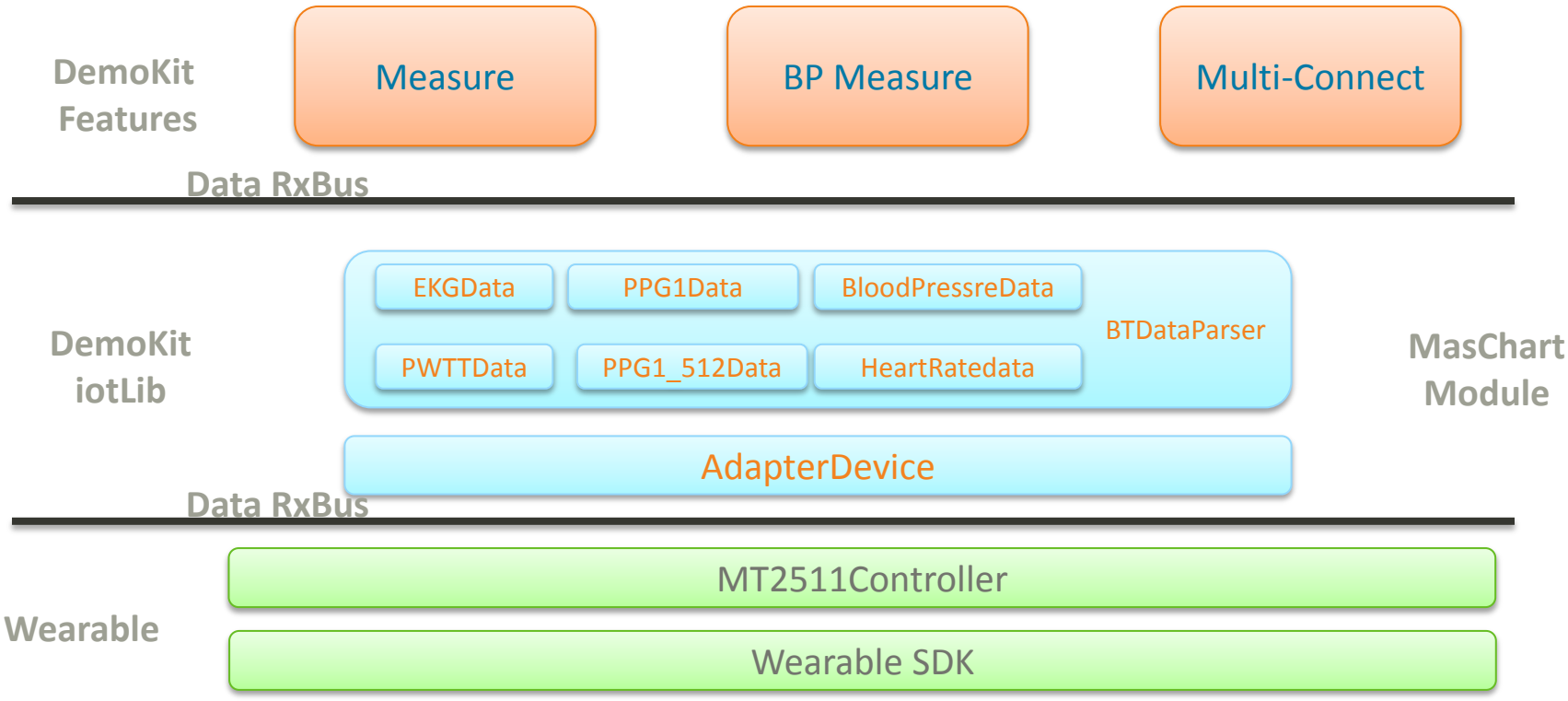
Overview

- MT2511 Health Module
  - bases on *MT2511Controller*@Smart Device, could receive feature data(Heart Rate/Blood Pressure/PWTT) and signal raw data (EKG/PPG1/PPG1-512hz) from device.

  (Note: receive signal raw data only in APK SPP mode.)

- It provides the following features:
  - Add record – Transfer personal profiles into device
  - HR Measure – measure heart rate
  - BP measure
    - General Mode – Measure personal blood pressure
    - Personal Mode – Measure bp with saved personal model, which created from calibration process.
  - Multiple connect – Add reference device for heart rate comparison

Architecture



Introduction

Package Structure

- *Package Structure*
  - DemoKit project include:
    - iot: Data Transfer & Bluetooth features

      BTData & DataParser: Data format module

    - mwcdemo: MT2511 UI features

Main Class (1/3)

- *MT2511Controller*
  - Inherit from Wearable SDK *Controller*
  - Receiver: *health_receiver.*
  - Implement *onConnectionStateChange*, *onReceive* to receive and post MT2511 *ReceiveData* by using RxBus.
  - Add/Remove *MT2511Controller* in *MainService onCreate/onDestroy.*
  - *Handshake* with device to Determine device whether or not support MT2511 health feature.
  - Observe *RequestWriteToDevice* by using *RxBus* and use *Controller send* API to *sendCommand*.

Main Class (2/3)

- *AdapterDevice*
  - Inherit from DemoKit/iot *Device*.
  - Observe *ReceiveData* from *MT2511Controller* and using *BTDataParser* to parse it, then post *BTBaseData* (HR/BP/EKG etc.) to UI module by using *RxBus*.
  - Provide *writeToDevice* API for UI module, *writeToDevice* could post *RequestWriteToDevice* to *MT2511Controller* and let *MT2511Controller* to send command.
  - *Connect/Disconnect* only post event by using *RxBus*.
  - Using *AdapterDeviceFactory* to crease *AdapterDevice*.

```
sBTDevice = new AdapterDevice(MContext.getInstance().getApplication(),
        new BTDataParser());
```

Main Class (3/3)

- *RxBus*
  - Data Bus implement with *RxJava*, could *post* java Object <T> to observer (call *toObservable* <T> class).
  - For more *RxJava* info, please refer to Appendix *RxJava*.
  - Sample Code:

    Subject Poster —— *DataParser/BTDataParser postData*

    Observer —— *MeasureFragment initView*

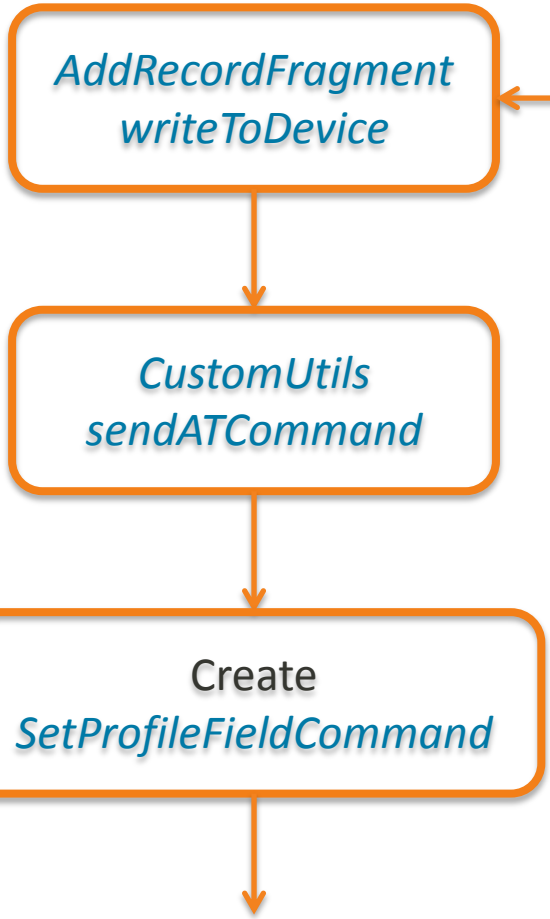
```
protected void postData(byte[] data) {
    try {
        BaseData baseData = parseData(data);
        if (isRememberData(baseData)) {
            RxBus.getInstance().postAndRemember(baseData);
        } else {
            RxBus.getInstance().post(baseData);
        }
```

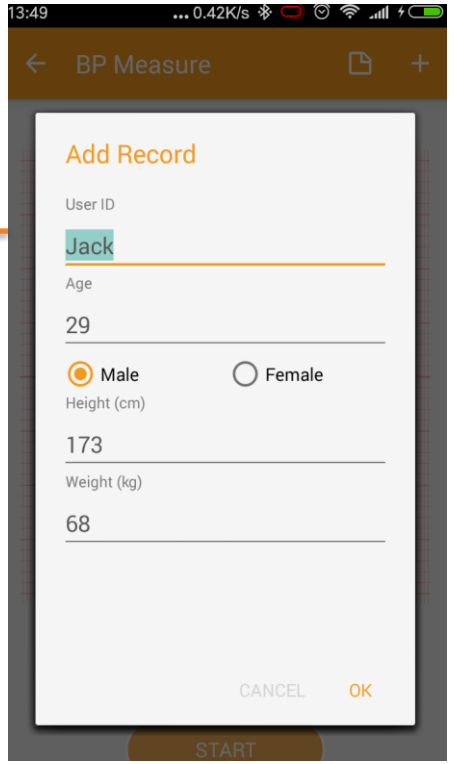
```
_subscriptions.add(RxBus.getInstance()
    .toObservable(HeartRateData.class)
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Action1<HeartRateData>() {
        @Override public void call(HeartRateData heartRateData) {
            int hr = heartRateData.get(HeartRateData.FIELD_BPM);
            String bmp_view_value = String.valueOf(hr);
            mTxtHeartRate1.setText(bmp_view_value);
        }
    }));
```

Send Command Flow (1/3)

- Send Command Flow – Send Person Profile

Add Record
(send person profile info to device)



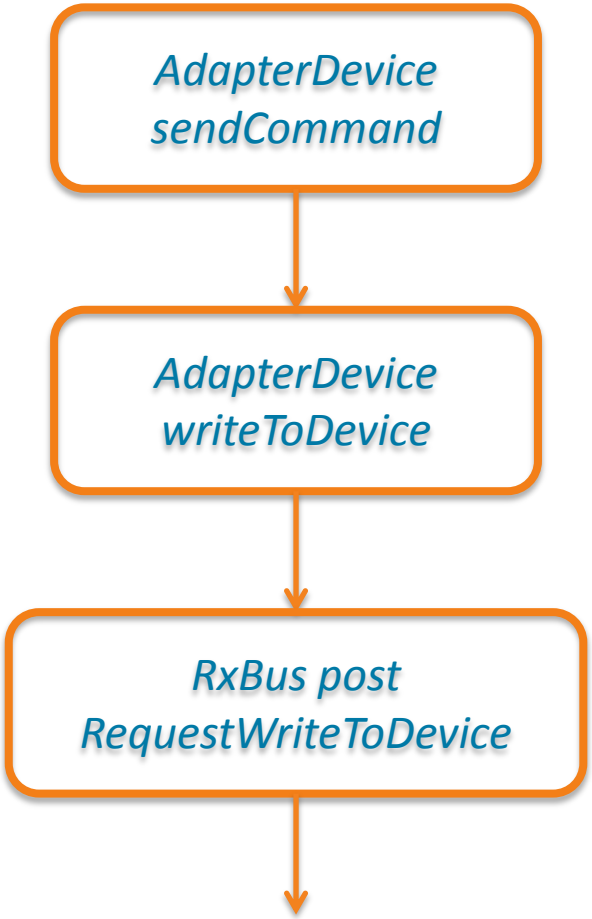


Send Command Flow (2/3)

- Send Command Flow – Send Person Profile

sendCommand
(Base Class *Device* method)

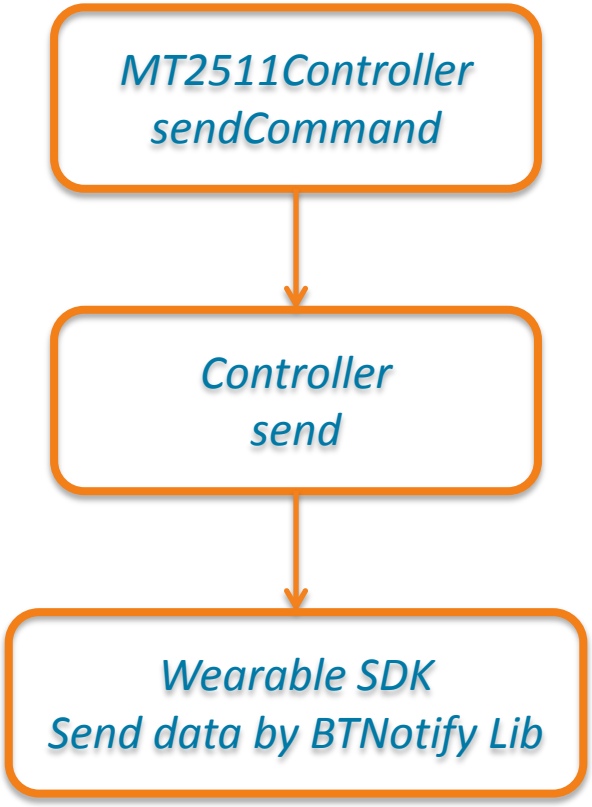
Run in
CommandOnSubscribe



Send Command Flow (3/3)

- Send Command Flow – Send Person Profile

MT2511Controller
(Observe
RequestWriteToDevice
by using *RxBus*)

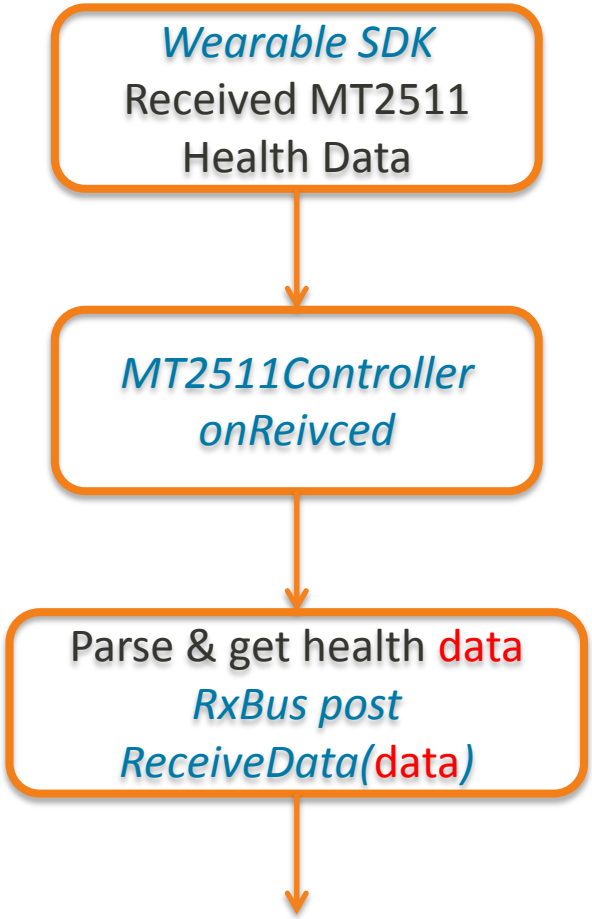


RequestWriteToDevice
(byte[] data)

health_sender health_receiver
0 0 data.length data

Receive Data Flow (1/3)

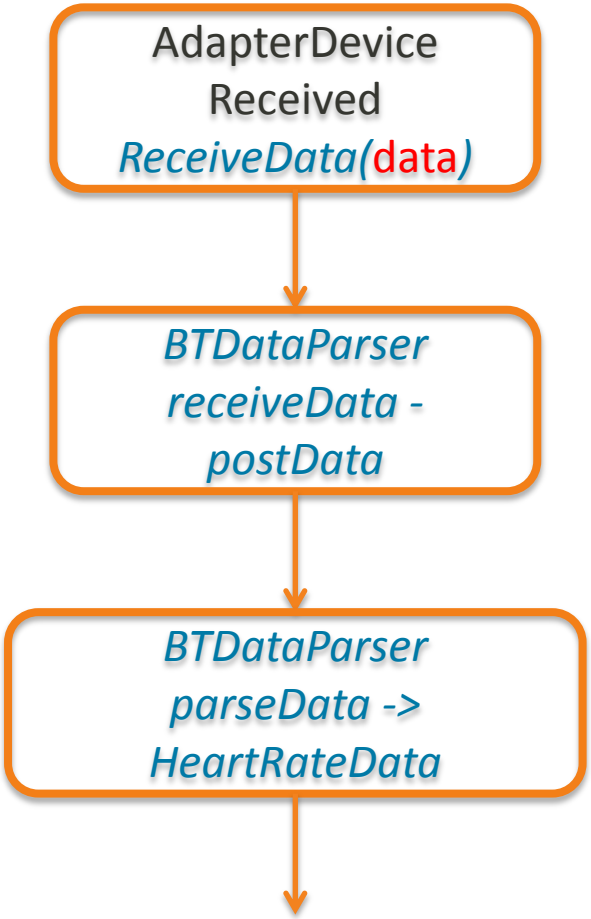
- Receive Data Flow – Receive HR bmp



Controller received data (health_send health_receiver 0 data.length data)

Receive Data Flow (2/3)

- Receive Data Flow – Receive HR bmp

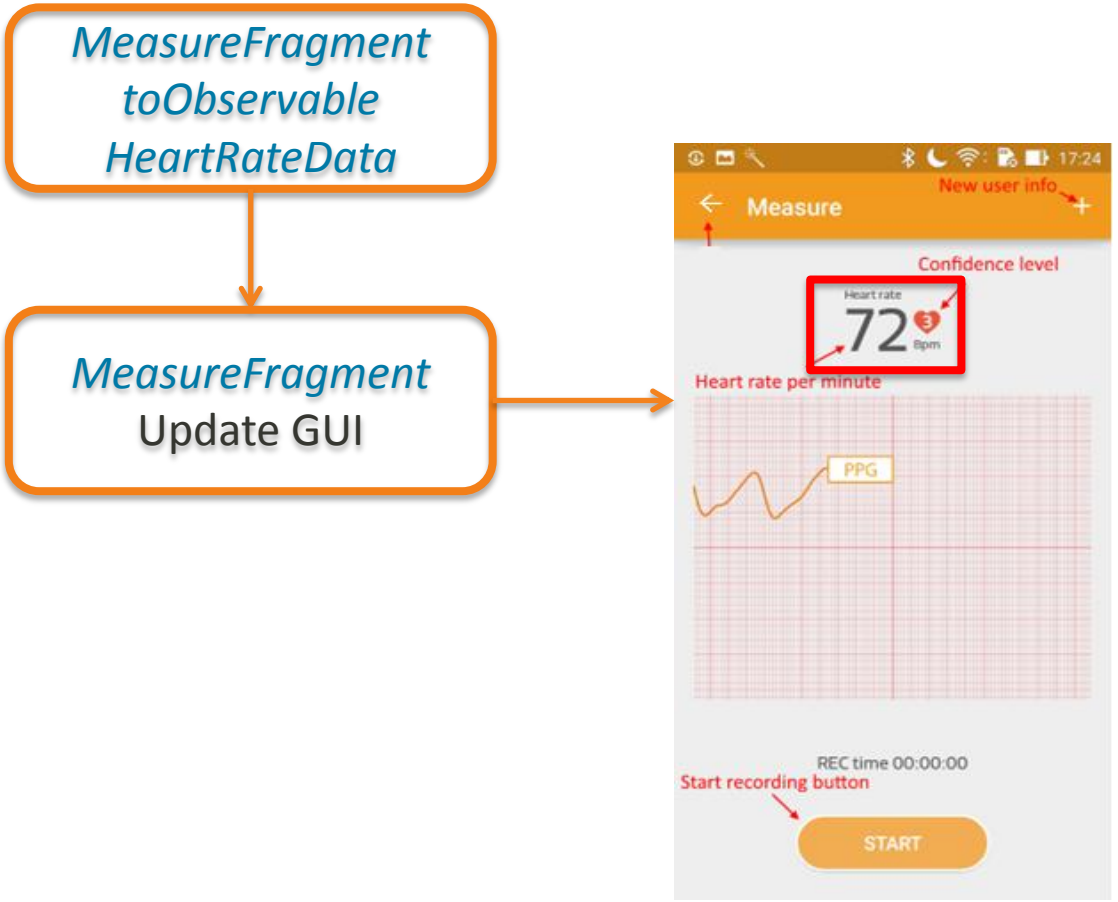


AdapterDevice
Observe *ReceiveData* by using *RxBus*. After receive *ReceiveData*, then grouped by each 64 bytes and call *Device dataAvailable*.

Finally, *RxBus* post *HeartRateData* to all *toObservable<HeartRateData>* class.

Receive Data Flow (3/3)

- Receive Data Flow – Receive HR bmp



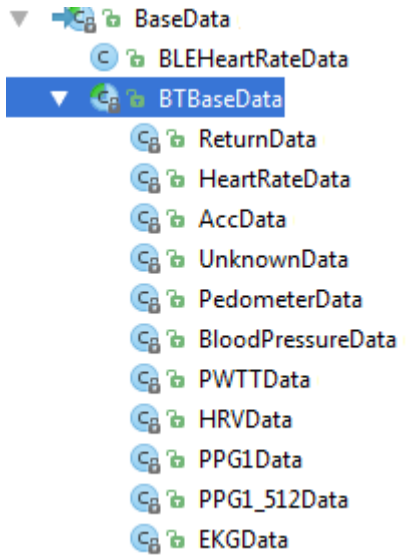
Data Format (1/8)

- Command (APK -> Device)
  - All command type extend from [BaseCommand].
  - Class Hierarchy

  - Sample Code:
    - *DemoKit\src\main\java\com\mediatek\iot\command*
    - *CustomUtils*

Data Format (2/8)

- Data (Device -> APK)
  - All data type extend from [BaseData].
  - Data Size 64 bytes = 16 * int (4 bytes)
  - Class Hierarchy



  - Sample Code:
    - *DemoKit\src\main\java\com\mediatek\iot\data*
    - *BTDataParser*
    - User can use *BTBaseData.get(int index)* method, get the filed as user need.

```
int hr = heartRateData.get(HeartRateData.FIELD_BPM);
```

Data Format (3/8)

- EKGData
  - A int array with length = 12 of EKG raw data
  - The EKG data are recorded in 512 hz.

EKG
[0] magic (54321)
[1] sensor type (sensor type is 5)
[2] sequence
[3-14] 12 ekg data
[15] reserved (12345)

```
public int[] getEkgData() {
    int[] data = new int[12];
    System.arraycopy(rawData, 3, data, 0, 12);
    return data;
}
```

  - Sample Log:

54321,5,111,7403228,7401915,7405257,7408628,7407863,7408847,7410483,7413195,7413959,7415612,7415998,7413969,12345

54321,5,112,7415451,7418458,7418712,7419345,7419505,7417862,7417064,7419372,7419947,7420994,7422060,7420012,12345

Data Format (4/8)

- PPG1Data
  - A int array with length = 12 of PPG1 raw data
  - The PPG1 data are recorded in 125 hz.

PPG1
[0] magic (54321)
[1] sensor type (sensor type is 9)
[2] sequence
[3-14] 6 ppg + 6 amb
[15] reserved (12345)

  - Sample Log:

54321,9,766,8356541,8363026,8356541,8363004,8356541,8363016,8356541,8363018,8357172,8363010,8359011,8363056,12345,

54321,9,767,8360529,8363029,8361763,8363007,8362794,8363034,8363612,8363030,8364296,8363062,8364873,8363123,12345,

54321,9,768,8365327,8363074,8365703,8363078,8365977,8363055,8366206,8363049,8366450,8363071,8366610,8363037,12345,

Data Format (5/8)

- PPG1_512Data
  - A int array with length = 12 of PPG1 raw data
  - The PPG1 data are recorded in 512 hz.

PPG1_512HZ
[0] magic (54321)
[1] sensor type (sensor type is 12)
[2] sequence
[3-14] 12 PPG Data
[15] reserved (12345)

  - Sample Log:

54321,12,1331,6018,5995,6005,6038,6052,6033,5997,6004,6044,6052,6032,5985,12345,

54321,12,1332,5991,6027,6046,6037,5996,6001,6029,6056,6041,6007,6004,6037,12345

Data Format (6/8)

- HeartRateData

HR
[0] magic (54321)
[1] sensor type (sensor type is 22)
[2] sequence
[3] bpm
[4] status
[5] timestamp
[6-15] reserved (12345)

```
public static final int FIELD_BPM       = 3;
public static final int FIELD_STATUS    = 4;
public static final int FIELD_TIMESTAMP = 5;
```

// HR Confidence Level = status & 0x000000FF

  - Sample:
  - *MeasureFragment toObservable(HeartRateData.class)*

54321,22,105,92,1228341247,-43075096,12345,12345,12345,12345,12345,12345
,12345,12345,12345,12345

Data Format (7/8)

- BloodPressureData

Blood Pressure
[0] magic (54321)
[1] sensor type (sensor type is 24)
[2] sequence
[3] sbp
[4] dbp
[5] HR bpm
[6] status
[7] timestamp
[8-15] reserved (12345)

```
public static final int FIELD_SBP      = 3;
public static final int FIELD_DBP      = 4;
public static final int FIELD_HR_BPM   = 5;
```

  - Sample:
  - *BPMeasureFragment showBPMeasureResult*

54321,24,1,109,76,62,0,0,12345,12345,12345,12345,12345,12345,12345,12345

Data Format (8/8)

- PWTTData
  - FEATURE_TYPE = 0:

    int array represent the PWTT interval

    of the blood pressure measure result
  - FEATURE_TYPE = 3:

    int array represent the personal model

    of the blood pressure calibration process

PWTT (pulse wave transit time)
[0] magic (54321)
[1] sensor type (sensor type is 8001)
[2] sequence
[3] feature type
[4] status
[5] timestamp
[6..N] feature value
(N<15)
[N+1..15] reserved (12345)

  - Sample:
  - *BPMeasureFragment .toObservable(PWTTData.class)*

54321,8001,1,0,9,0,192,162,180,204,56,153,210,53,259,12345

54321,8001,2,0,7,0,254,59,150,153,215,184,208,53,259,12345

54321,8001,3,3,6,0,109,76,62,182,324,0,208,53,259,12345

Appendix

Dependency

- **RxJava**
  - A library for composing asynchronous and event-based programs using observable sequences for the Java VM.
  - Web: Official Web in Github How-To-Use-RxJava
  - Build.gradle :

```
// RxJava - composing asynchronous and event-based programs using observable sequences
compile 'io.reactivex:rxjava:1.1.0'
compile 'io.reactivex:rxandroid:1.1.0'
compile 'com.jakewharton.rxbinding:rxbinding:0.3.0'
```

  - Sample Code
    - *RxBus/DataParser/AdaperDevice* Java File
    - *MeasureFragment initView*()

SNR Computation

- SNR is comparison result of the level of signal to the level to noise.
- SNR Class:
  DemoKit\demoKit\com\mediatek\mwcdemo\snr\SNRResult.java
- Sample Code:
  DemoKit\demoKit\com\mediatek\mwcdemo\fragments\BPMeasureFragment.java

```
// insert high pass filtered input to the lib when raw data received
double s_hpf = mECGFilterService.filter(data_ecg_512_mv);
mECGSNR.inputHPFSignal(s_hpf);
…
// when the measurement is finished, compute the low pass signal array with convolution function.
//compute the SNR result afterwards.
hpfSignalList = mECGSNR.getHPFSignal();
allLPFSingalList = mECGFilterService.conv(hpfSignalList);
mECGSNR.computeECG512SNR40(allLPFSingalList);
```