shimmer
DISCOVERY IN MOTION

Shimmer Android Instrument Driver Library User Manual

Revision 1.0a

Legal Notices and Disclaimer

Table of Contents

1. Introduction

The current Android Shimmer Library version is a beta release. Bluetooth communication from the Shimmer device to the Android device is via the Serial Port Profile (SPP). SPP has been supported on Android since API level 5 (Android 2.0) was introduced, thus the driver will work with any Bluetooth enabled Android device which has version of 2.0 and above.

It should be noted that there is sometimes variability between 'Stock' firmware, manufacturer modified firmware, and custom firmware. Users are advised to investigate whether there are any known issues with their implementation of their Bluetooth stack. For example, Android 4.2 (Stock) uses a new Bluetooth Stack.

1.1. Scope of this Document

Android Developers provide extensive online documentation for all core Android components. This documentation limits itself to explaining Shimmer based functionality. While effort has been made to provide as much information that is required, the user may need to develop some further understanding through the general study of Android and Java programming respectively.

2. Pre-Requisites

The following are the pre-requisites needed to use the *Shimmer Android Instrument Driver Library*:

- JDK installed on your PC.
- Android SDK.
- Eclipse installed on your PC. (While Eclipse is not compulsory it is highly recommended).
- ADT Plugin for Eclipse.
- An Android device with Bluetooth.
- The Shimmer Android Instrument driver class files which can be downloaded from www.shimmer-research.com/download and includes the following

    a. *Shimmer.jar file*

    b. *Shimmer Android Library User Manual.pdf.*

    c. Shimmer Android Readme.txt

    Download the file Shimmer Android Driver Library (Beta 0.x).zip from here http://www.shimmer-research.com/download/softwares. Extract the files and follow the installation instructions outlined in the Getting Started section below.

- A Shimmer 2 or Shimmer 2r device programmed with either of the following firmware; BTStream or BoilerPlate. Readers should note that Boilerplate is now legacy firmware and it is highly recommended that BTStream be used when possible. The firmware can be found in the Firmware folder of the downloaded file. This is also available for download from

www.shimmer-research.com/download. These images can be loaded onto the Shimmer devices using the *Shimmer Windows Bootstrap Loader* application. See the *Shimmer User Manual* for details.

3. Getting Started

First install the appropriate Boilerplate firmware image onto your Shimmer device. Use *BoilerPlate_0_1_shimmer2.ihex* if you are using a Shimmer 2 and BTStream/*BoilerPlate_0_1_shimmer2r.ihex* if you are using a Shimmer 2r. Earlier versions of the Shimmer are not supported. These images can be loaded onto the Shimmer using the *Shimmer Windows Bootstrap Loader* application. See the *Shimmer User Manual* for details.

The next step is setting up Eclipse and the Android SDK. To do as such follow the instructions given in http://developer.android.com/sdk/installing.html.

3.1. Running an example application on your android platform

1. Open the Eclipse application and select import.

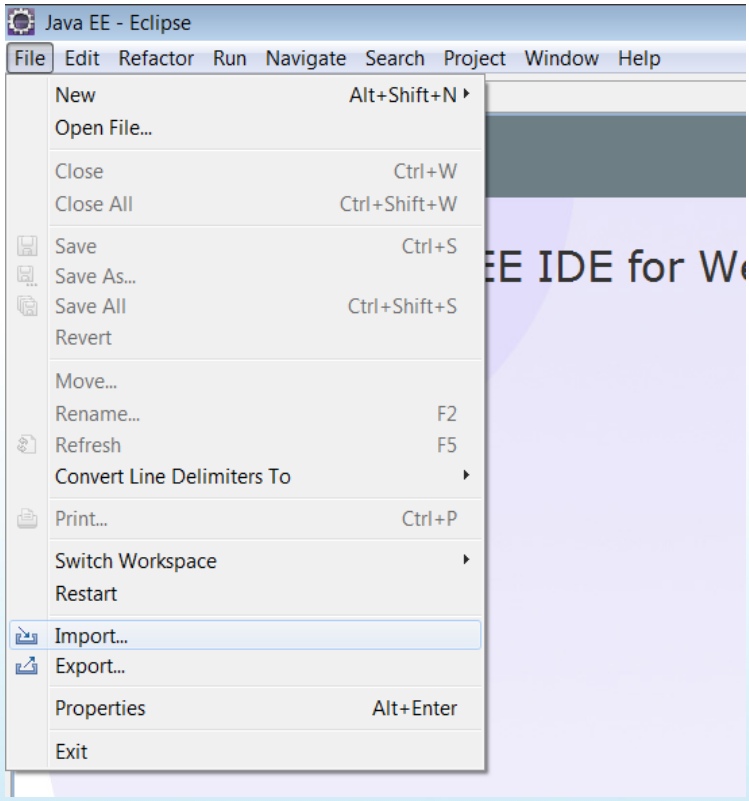


Figure 1: Import Project

2. Next Select Existing Android Code into Workspace

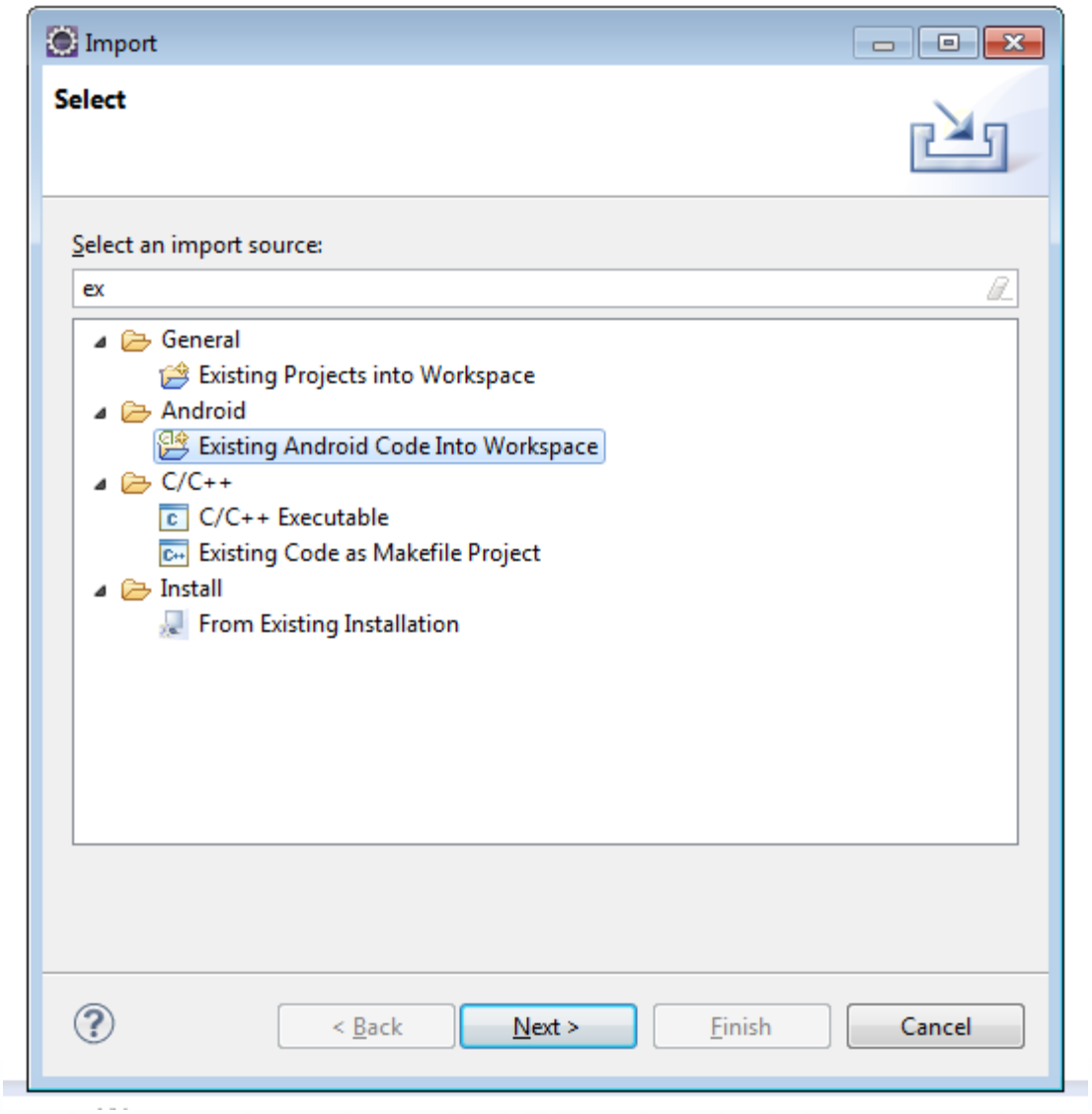


Figure 2: Select Existing Projects

3. Scroll to the example folder and select the ShimmerGraph example

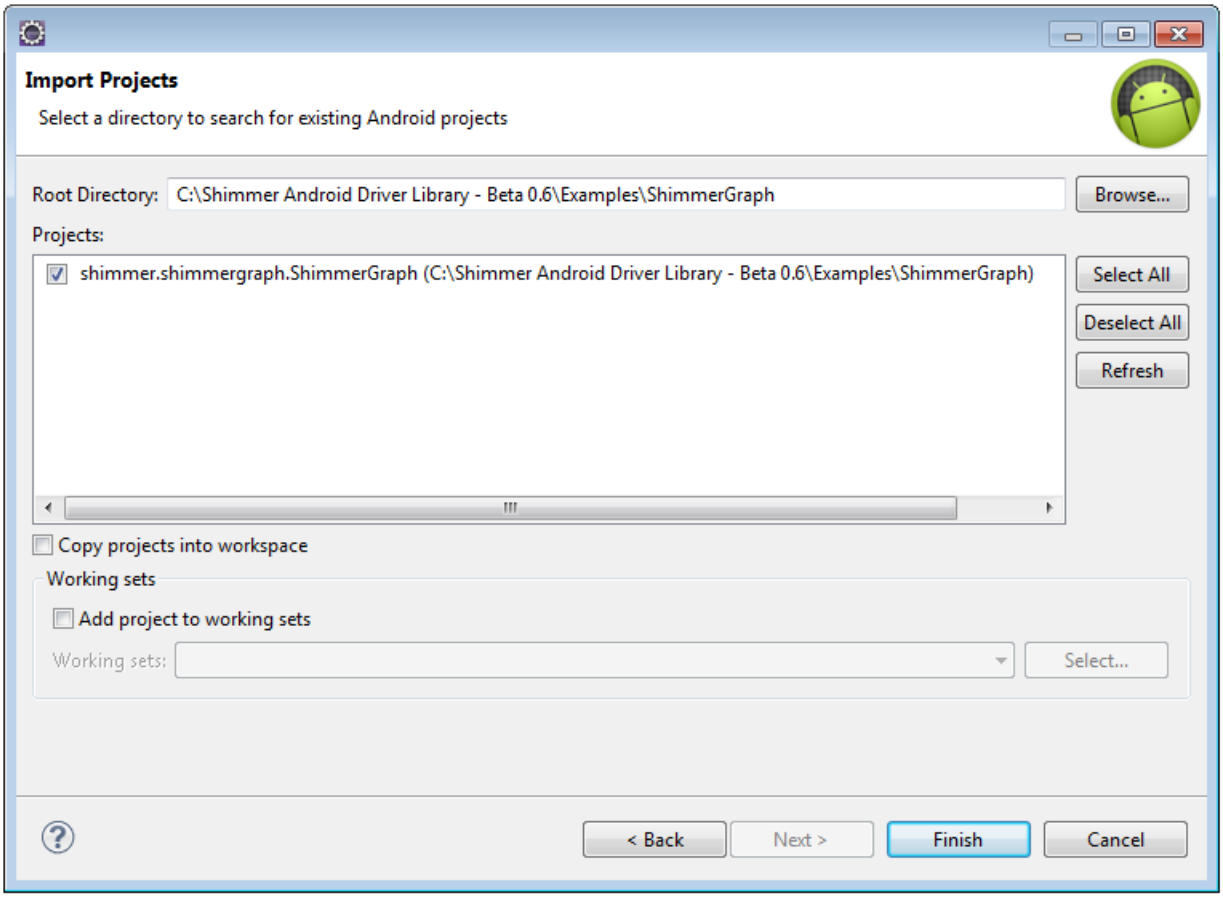


Figure 3: Select an Example

4. Click Finish and go to your workbench. You should now see the following :-

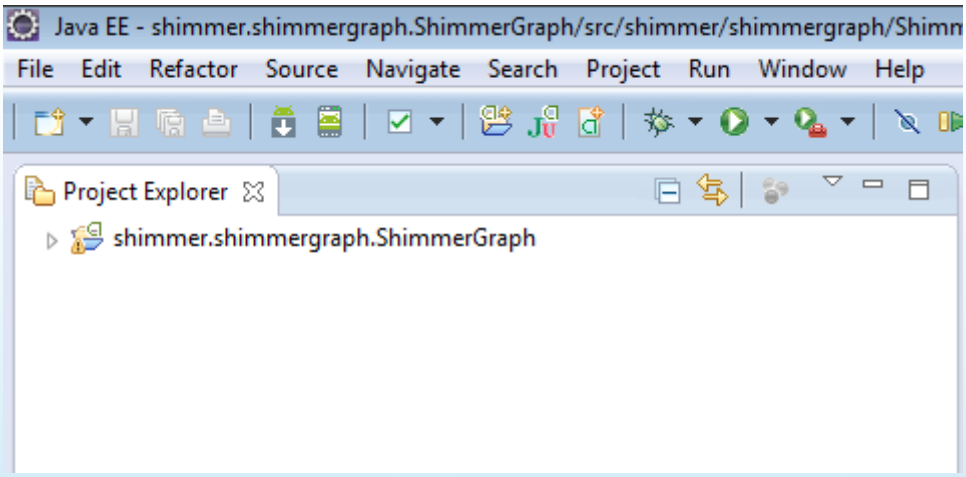


Figure 4: ShimmerGraph Project

5. Next install the application onto your target Android device, readers should note that the Android emulator does not support Bluetooth. Connect your android device to your computer via USB and enable USB debugging on your android device. This option can be found on your android device here:-

Settings-> Developer options -> USB debugging

Also ensure that the drivers for your android device have been installed on your computer.

6. Set the Deployment Target Selection Mode to manual. This can be found under Run -> Debug Configurations -> Target.

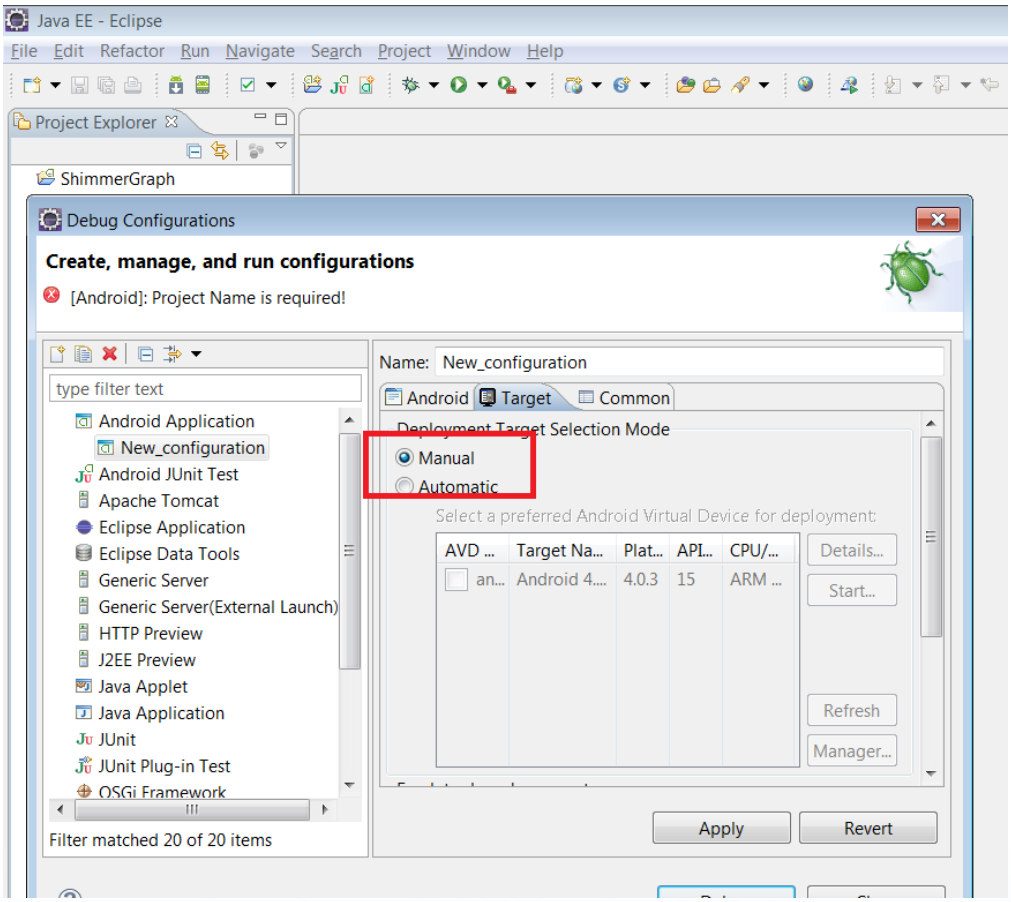


Figure 5: Select Manual Deployment Target Selection

7. Next run the application as an Android application as shown below

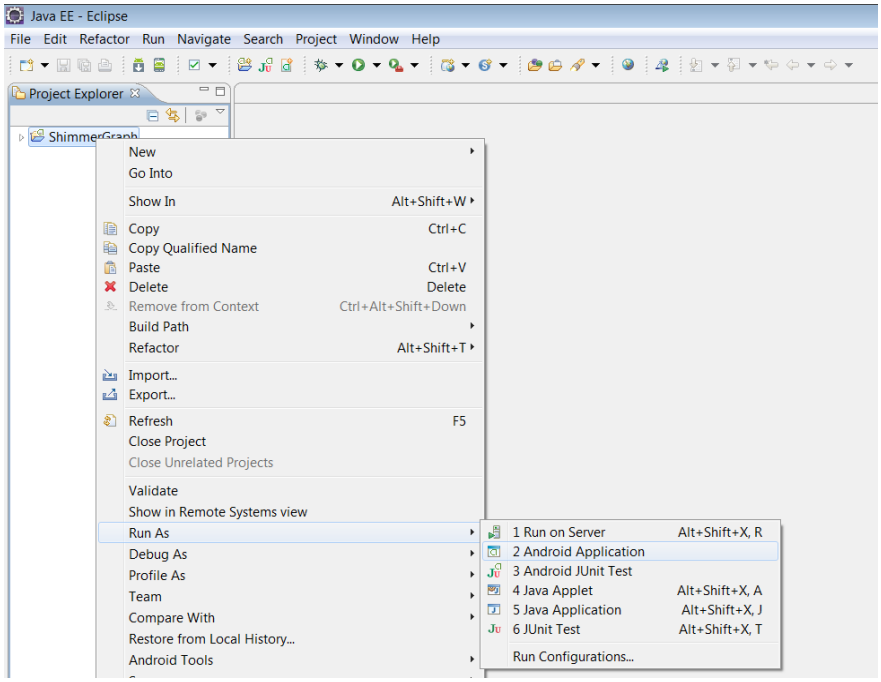


Figure 6: Run the Application

8. You should now see the ShimmerGraph App loaded onto your Android device. Next watch the related ShimmerGraph video to see an explanation of the application.

9. Next you will want to load LogCat to view messages logged by the Android device. This can be done by going to:-

   Window->Show View->Other…->Android->LogCat

   You can use this logging mechanism to debug as you start working on your implementation.

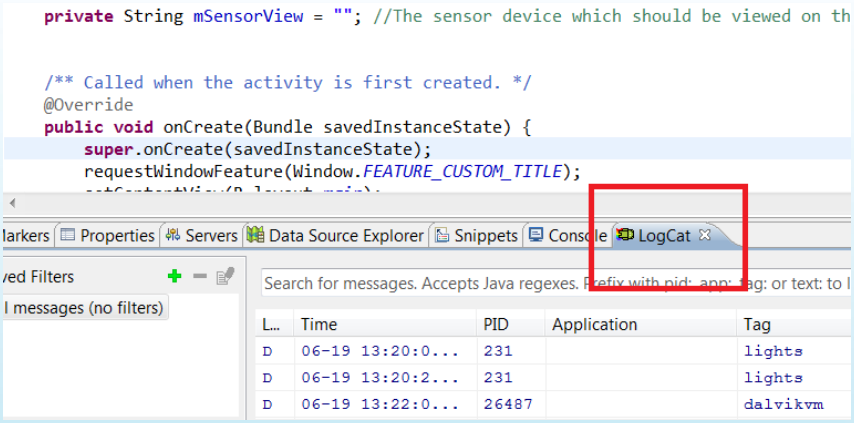


Figure 7: Load LogCat

4. Shimmer Android Instrument Driver Library

4.1. General Overview

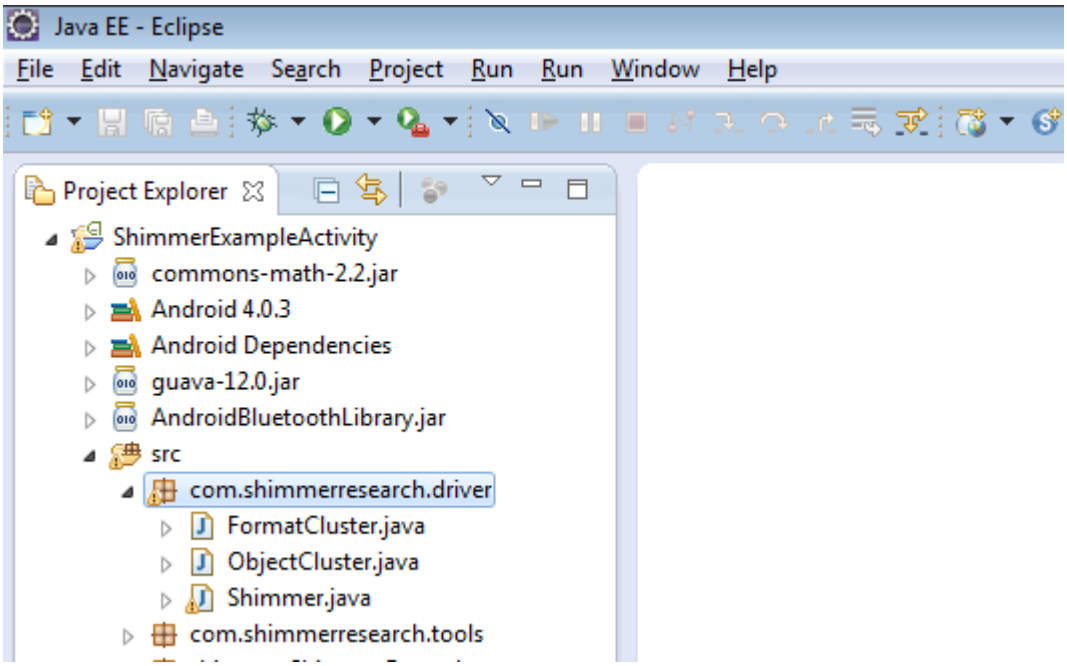


Figure 8: Set of Packages which make up the Shimmer Android Instrument Driver Library

The Shimmer Android Instrument Driver library is a set of packages provided in order to speed up the development time of Shimmer Users. The package com.shimmerresearch.driver contains three files. The first file, Shimmer class, is the main driver file, which allows the main android UI to connect to a Shimmer device. Boilerplate/BTStream is the firmware used on the Shimmer device. It provides control for the sensors and the Bluetooth radio communication. It is written in TinyOS. The Shimmer Class sits above the Android Bluetooth API and provides a number of functions which allows the main Android application to communicate with the Shimmer Device. More information concerning the Android Bluetooth API can be found here http://developer.android.com/guide/topics/wireless/bluetooth.html. The Shimmer class allows the main android application to interact with the Shimmer device (e.g. setting the sampling rate, changing the accelerometer range, enabling/disabling sensors). Multiple Shimmer devices can be connected simultaneously to an Android device using multiple instances of the Shimmer Class object.

The other two files (FormatCluster and ObjectCluster) within the ***com.shimmerresearch.driver*** package define the data structure. The package ***com.shimmerresearch.tools*** contains a Logging class which allows the user to easily log data onto an Android phone. In future we plan to add other related tools which can help speed up the development speed of applications.

4.2. The Shimmer Class

As mentioned the Shimmer class is an object oriented piece of code which allows your android application to interact with a Shimmer device.

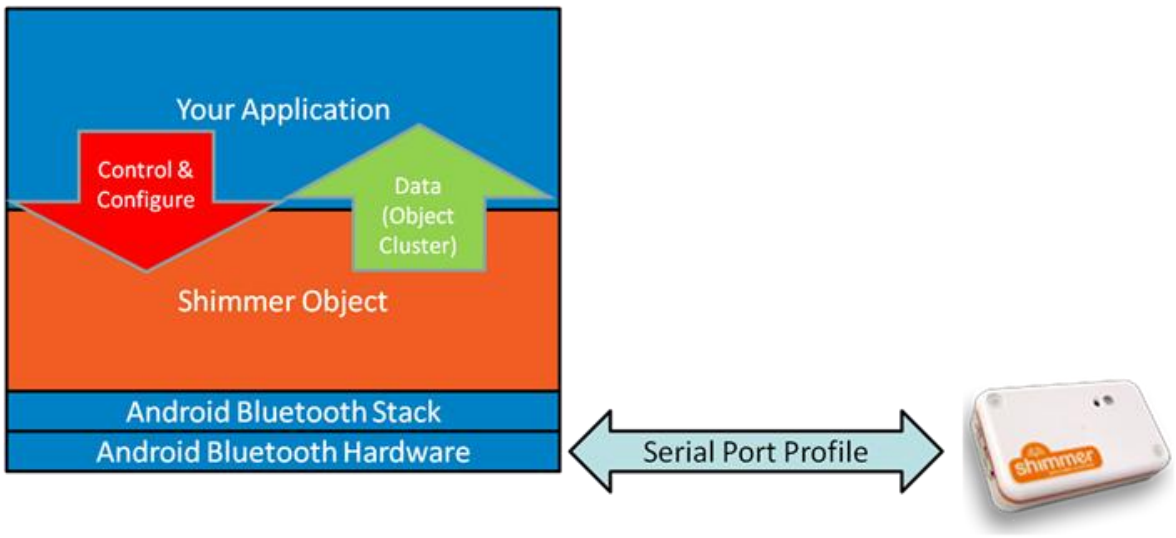


Figure 9: Shimmer Object

The Shimmer class relies on the Bluetooth stack provided by Android to connect with the Shimmer device via the Serial Port Profile (SPP). SPP emulates a serial cable link over Bluetooth wireless technology. Each Shimmer device connected to the Android Device is represented by a Shimmer object which is an instance of the Shimmer class. For example:-

```
Shimmer mShimmerDevice1 = new Shimmer(this, mHandler,"RightArm",false);
Shimmer mShimmerDevice2 = new Shimmer(this, mHandler,"LeftArm",false);
```

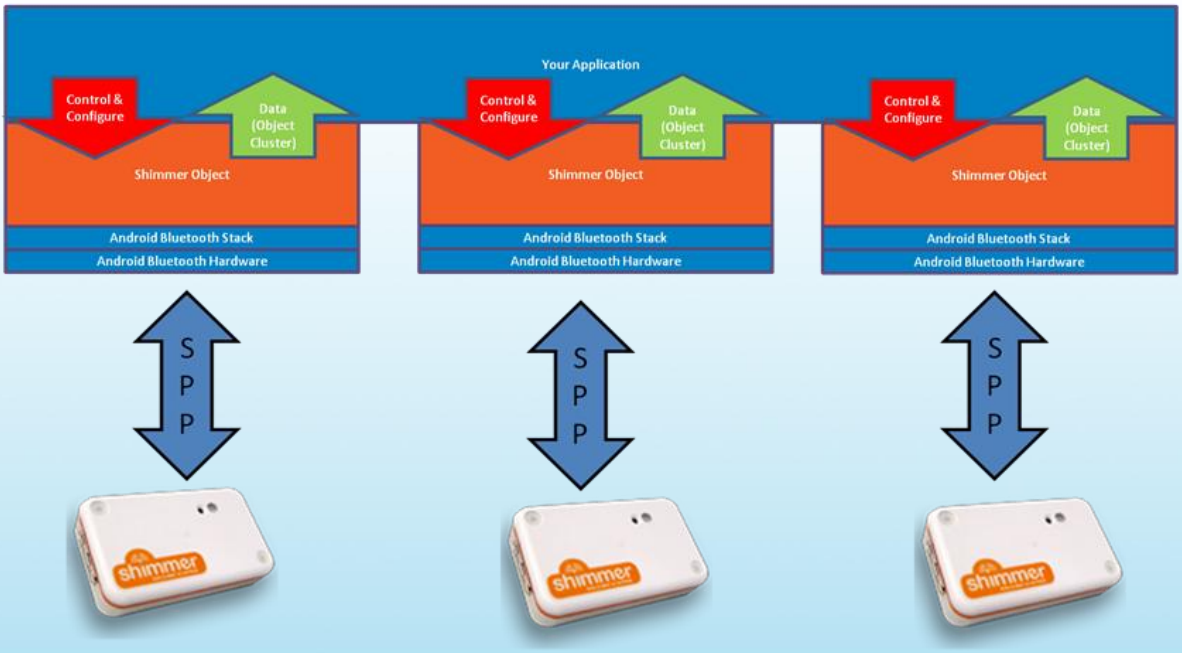


Figure 10: Multiple Shimmer Objects

There are two constructors provided for the Shimmer class, one will not change the configuration of the Shimmer device once a connection is made to the Shimmer device. The other constructor allows you to immediately set the configuration of the Shimmer device you will be connecting to. Once a connection has been established, the Shimmer device is configured according to the specifications of the constructor.

The Shimmer Class exposes a number of functions to allow the Android Device to individually control each Shimmer device. The reader should note that functions which start with either write or read are direct commands to the Shimmer device, while functions which start with get or set are commands which only access the Shimmer class object.

The Shimmer device responds with an Acknowledge packet whenever a command is received. In some cases after sending an Acknowledge packet the Shimmer device proceeds in transmitting a response packet or data packets. This occurs when a read or streaming command is transmitted.

The following are among the few primary functions provided by the class:-

- *Connect*; e.g. *mShimmerDevice1.connect(bluetoothAddress,"default");*
    - Attempts to connect the Android device to the Shimmer device. There are two separate library options provided *default* and *gerdavax*. Only in cases where the *default* library does not meet your expectations should you attempt to use the *gerdavax* library.
- *startStreaming*
    - Starts streaming of the Shimmer Device
- *stopStreaming*
    - Stops streaming of the Shimmer Device
- *Inquiry*
    - An inquiry command allows the Android device to learn the current setup of the Shimmer device. Upon the reception of an inquiry command the Shimmer device will transmit and acknowledge packet and an inquiry response packet. The response packet contains information concerning the current setup of the Shimmer device, such as the sensors which have been enabled, the data packet size, and the contents of the data packet.
- *writeEnabledSensors*
    - This command is used to enable specified sensors on the Shimmer device. After the acknowledge packet is received an inquiry command is executed to ensure the packet format is updated. When enabling multiple sensors on the Shimmer device use the following format where an Or operator is used. *E.g. mShimmerDevice1.writeEnabledSensors(Shimmer.SENSOR_ACCEL|Shimmer.SENSOR_GYRO);*

- *writeSamplingRate*
    - Transmits a command to the Shimmer unit, configuring its sampling rate.
- *writeAccelRange*
    - Transmits a command setting the range of the accelerometer on the Shimmer device

By default the Shimmer class will retrieve the calibration parameters from the Shimmer device. In the event calibration parameters are not found. The Shimmer device will use the default calibration parameters. Sensor data with units ending with * (e.g. m/(sec^2)*) means default calibration parameters were used. Please refer to the 'Shimmer 9DoF Calibration Application – User Manual' for more information on calibration.

Sensor data, commands and status messages are passed on from the Shimmer Class to the android application using the Handler Class provided by the Android library. More information concerning the Handler Class is provided here. In examples it is important to note how toast messages and data packets are sent from the Shimmer class to the main activity via the Handler.

4.3. Data Structure

The ObjectCluster and Property Cluster class describes the data structure used within this library. The data structure is used to encapsulate data from the driver. The ObjectCluster is made up of a MultiMap (PropertyCluster) where each key represents a Property (e.g. Accelerometer X) and each value the FormatCluster of that property. The FormatCluster is an object which holds the format (e.g. Calibrated), units and data of the property. An example of the data structure is shown in Figure 11.

This design was chosen to allow new properties such as Linear Acceleration and orientation to be included easily in the future along with new formats such as filtered. Through the use of MultiMaps, properties within the data structure can be found easily. More information on MultiMaps can be found here.

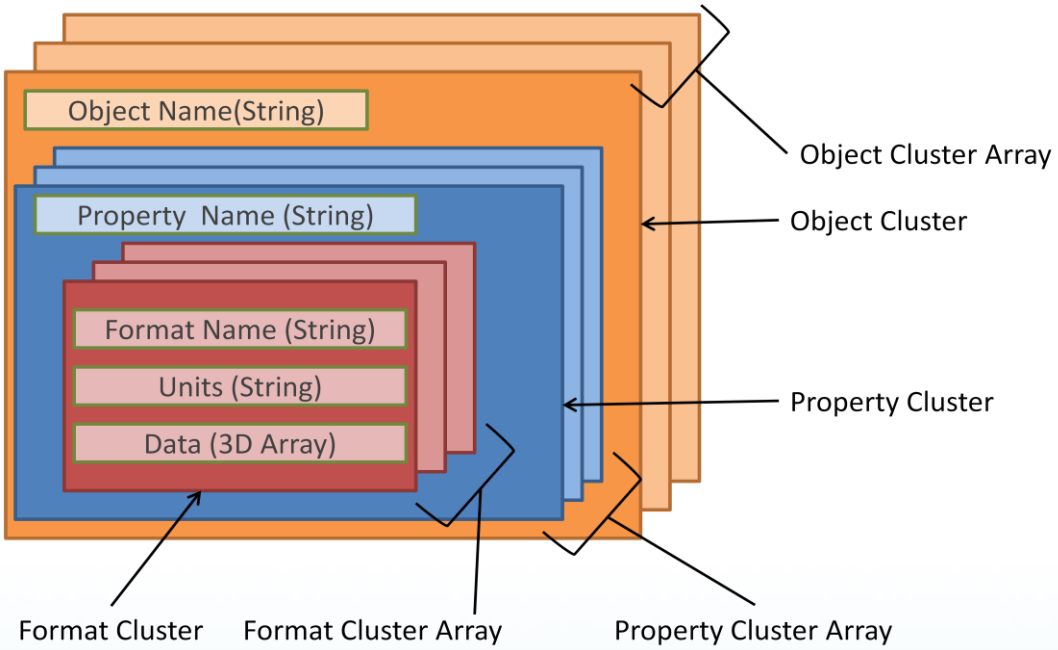


Figure 11: Data Structure

These seven lines of code show the basic operation of the Object Cluster. In the first line a new ObjectCLuster with the name RightArm is created. Lines 2-5 show how new properties are added to the object cluster. The resulting object cluster is shown in Figure 11. For better details users should refer to the *buildMsg* function within the Shimmer class.

```
1.  ObjectCluster objectCluster=new ObjectCluster("RightArm");
2.  objectCluster.mPropertyCluster.put("Accelerometer X",new FormatCluster("CAL","m/(sec^2)",0.5));
3.  objectCluster.mPropertyCluster.put("Accelerometer X",new FormatCluster("RAW","no units",50));
4.  objectCluster.mPropertyCluster.put("Accelerometer Y",new FormatCluster("CAL"," m/(sec^2)",0.5));
5.  objectCluster.mPropertyCluster.put("Accelerometer Y",new FormatCluster("RAW","no units",50))
6.  Collection<FormatCluster> accelXFormats = objectCluster.mPropertyCluster.get("Accelerometer X");
    // first retrieve all the possible formats for the current sensor device
7.  FormatCluster formatCluster =
    ((FormatCluster)objectCluster.returnFormatCluster(accelXFormats,"CAL")); // retrieve the
    calibrated data
8.  dataValue = formatCluster.mData;
```

Line 6 - 8 show how data is retrieved from a data structure. Initially the MultiMap (PropertyCluster) is searched for the property “Accelerometer X”. The result of this search is a Collection of different format clusters of Accelerometer X. More information on the Collection class can be found here http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Collection.html. Next in line 7 the returnFormatCluster function is used to return the Calibrated Data of Accelerometer X. Finally line 8 shows the data being passed to the variable dataValue.

The lists of properties currently supported by the Shimmer class are:-
Timestamp, Accelerometer X, Accelerometer Y, Accelerometer Z, Gyroscope X, Gyroscope Y, Gyroscope Z,
Magnetometer X, Magnetometer Y, Magnetometer Z, GSR, ECG RA-LL, ECG LA-LL, EMG, Strain Gauge High,
Strain Gauge Low, Heart Rate, ExpBoard A0, ExpBoard A7, VSenseReg and VSenseBatt

The lists of formats supported by the Shimmer class are:-
CAL, RAW

5. Android Design Concepts

In this section we go through some basic Android concepts which readers should take note off. Most of the information have been taken from http://developer.android.com/develop/index.html and condensed where possible to suit the needs of this user guide. For a more in depth view the reader should refer to the source.

5.1. Activity and Service

An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows.

A Service is an application component that can perform long-running operations in the background and does not provide a user interface

It is important to understand when to use a service when designing an Android application. Figure 12 shows the lifecycle of an activity. From the flowchart it can be seen that an Application process can be killed when apps with higher priority need memory.

The Android system attempts to keep application process around for as long as possible, but eventually will need to remove old processes when memory runs low. As described in Activity Lifecycle, the decision about which process to remove is intimately tied to the state of the user's interaction with it. In general, there are four states a process can be in based on the activities running in it, listed here in order of importance. The system will kill less important processes (the last ones) before it resorts to killing more important processes (the first ones).

The **foreground activity** (the activity at the top of the screen that the user is currently interacting with) is considered the *most important*. Its process will only be killed as a last resort, if it uses more memory than is available on the device. Generally at this point the device has reached a memory paging state, so this is required in order to keep the user interface responsive.

A **visible activity** (an activity that is visible to the user but not in the foreground, such as one sitting behind a foreground dialog) is considered extremely important and will not be killed unless that is required to keep the foreground activity running.

A **background activity** (an activity that is not visible to the user and has been paused) is no longer critical, so the system may safely kill its process to reclaim memory for other foreground or visible processes. If its process needs to be killed, when the user navigates back to the activity (making it visible on the screen again), its onCreate(Bundle) method will be called with the savedInstanceState it had previously supplied in onSaveInstanceState(Bundle) so that it can restart itself in the same state as the user last left it.

An **empty process** is one hosting no activities or other application components (such as Service or BroadcastReceiver classes). These are killed very quickly by the system as memory becomes low. For this reason, any background operation you do outside of an activity must be

executed in the context of an activity BroadcastReceiver or Service to ensure that the system knows it needs to keep your process around.

Sometimes an Activity may need to do a long-running operation that exists independently of the activity lifecycle itself. An example is when you are logging data from a Shimmer device. As logging can take a long time, the application should allow the user to leave the application while it is executing. To accomplish this, your Activity should start a Service in which logging takes place. This allows the system to properly prioritize your process (considering it to be more important than other non-visible applications) for the duration of the upload, independent of whether the original activity is paused, stopped, or finished. An example of how to use a service in logging context in given in the *ShimmerGraphandLogService* example.

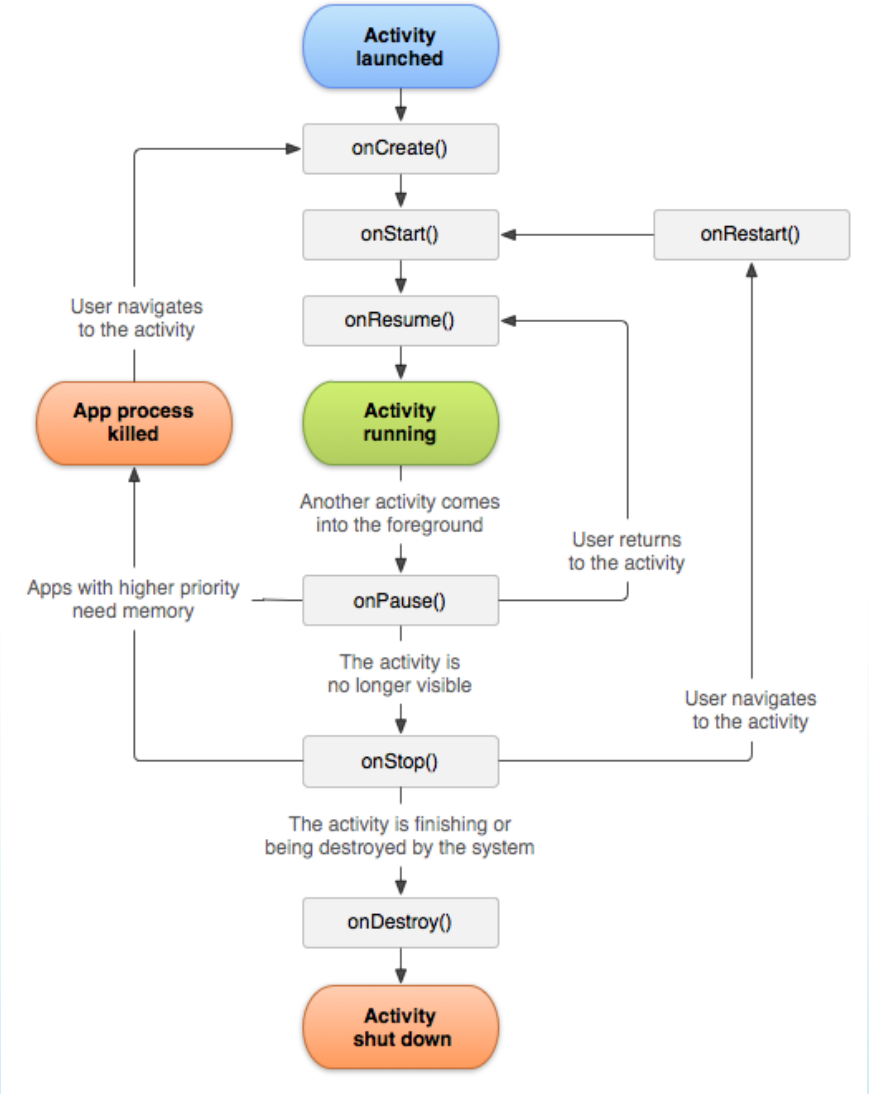


Figure 12: Activity lifecycle[1]

[1] http://developer.android.com/reference/android/app/Activity.html

5.2. The Handler

When a process is created for your application, its main thread is dedicated to running a message queue that takes care of managing the top-level application objects (activities, broadcast receivers, etc) and any windows they create. You can create your own threads, and communicate back with the main application thread through a Handler. This is done by calling the *post* or *sendMessage* methods, from your new thread.

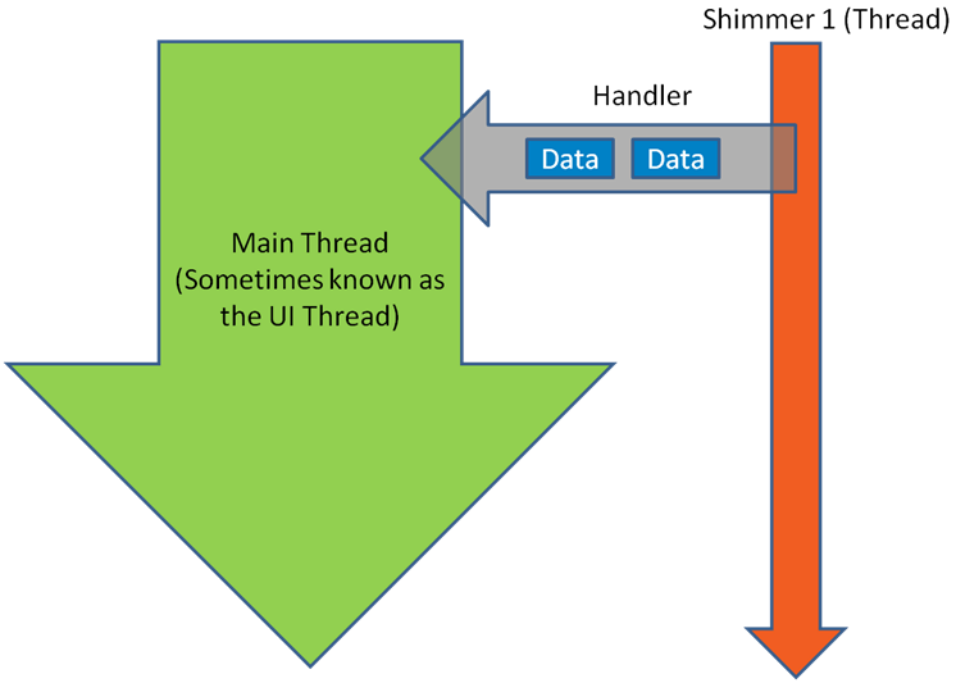


Figure 13: Handler

In Section 6.1 an example which shows some of the different types of messages passed on from the Shimmer thread to the Main UI thread is given. In Section 6.3 an example which shows how to use the Handler to connect to multiple Shimmer devices sequentially without having to lock the Main UI thread is given.

6. Examples

There are a number of examples provided with the library. In this section a run through of each example and some of the important design concepts readers should be aware of is presented.

6.1. ShimmerExample

The Shimmer example is the most basic example provided. It does not have any UI elements in it. The purpose of the example is to show how to use the Shimmer driver to connect an Android phone to a Shimmer unit. Readers should take note of the *Handler* in *ShimmerExampleActivity.* As mentioned the handler is the means of communicating back with the main application thread from your own created thread.

In the example there are three types of messages sent from the Shimmer thread to the Main UI thread via the handler. The messages are *Shimmer.MESSAGE_READ*, *Shimmer.MESSAGE_STATE_CHANGE* and *Shimmer.MESSAGE_TOAST* . *Shimmer.MESSAGE_READ is used to send sensor data*. *Shimmer.MESSAGE_STATE_CHANGE* is used to notify the activity page which state the shimmer is in. ***Note that a Shimmer is only deem connected once the Shimmer.MSG_STATE_FULLY_INITIALIZED message has been received, at which point other commands may be transmitted to the Shimmer device, such as writeenabledsensors.*** *Shimmer.MESSAGE_TOAST* is used to receive and post toast messages from the driver onto the screen. Other important things to note is how properties and formats are retrieved from the object cluster.

When using this example the user should have the LogCat open. Sensor data is printed on the LogCat via the following command:-

```
Log.d("CalibratedData","AccelX: " + formatCluster.mData + " "+ formatCluster.mUnits);
```

Figure 17 shows an example output of the LogCat.

6.2. ShimmerLogExample

This example builds on the ShimerExample and shows how to log data. Using the *Logging* class provided in *com.shimmerresearch.tools*. In particular should note the following line of code:-

```
private static String mFileName = "shimmerlogexample";
static Logging log = new Logging(mFileName,"\t"); //insert file name
log.logData(objectCluster);
```

which logs the data in a tab delimited (*"\t"*) format. The file is saved into the *externalstoragedirectory*. See figure below for example of log file, being viewed from an Android device using a file explorer program.

Figure 14: Screenshot of ShimmerLogExample

Readers should note that this example is just to get users started, and that it is better to have log data through a service, than from within an activity. An example of how to do this can be found in the example *ShimmerGraphandLogService*.

6.3. MultiShimmerExample

This example shows how to connect and stream data from two different shimmer devices. In the example the second connection is only executed after the first has been fully initialized. This is demonstrated in the figure. First a command to connect to a shimmer device is executed in the main thread. This creates the Shimmer 1 thread. When the device is connected to a message (see *Shimmer.MSG_STATE_FULLY_INITIALIZED*) is sent to the Main UI thread which then executes the command to connect to the second Shimmer device. By doing so you can avoid locking up the main UI thread when waiting for the connection to Shimmer1 to finish.

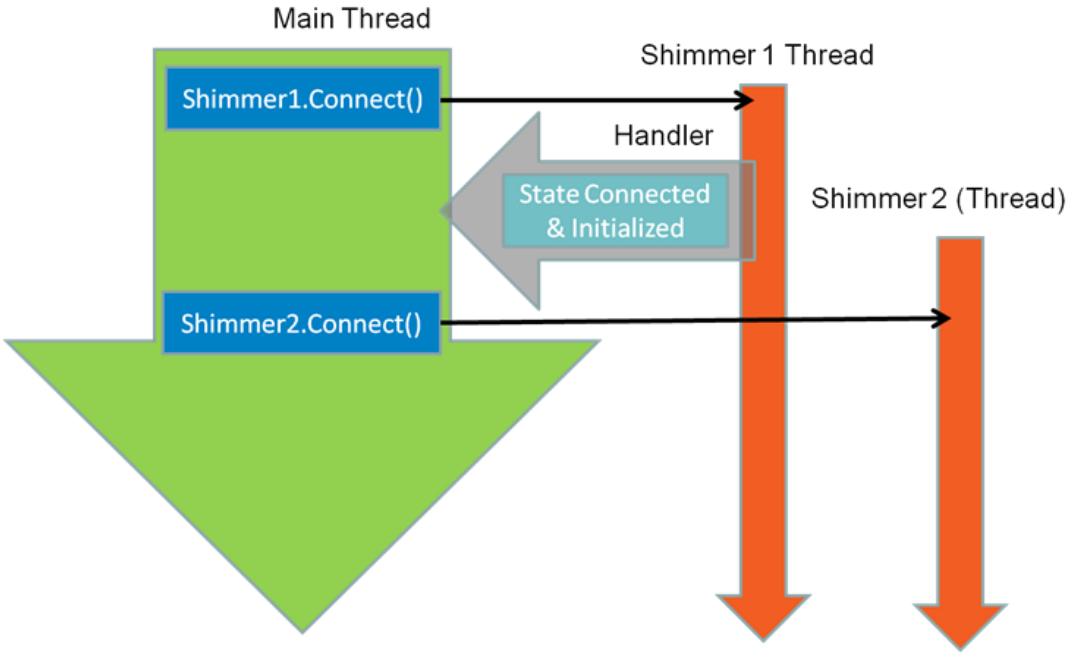


Figure 15: Handler, connecting two Shimmer devices sequentially

One reads in the Shimmer unit's Configuration, and the other sets the Shimmer unit's configuration.

6.4. ShimmerGraph

Is a simple UI example which allows you to connect and stream data from a Shimmer device. Users looking to build on this example should consider using the ShimmerGraphandLogService instead.

6.5. MultiShimmerGraph

This builds on the ShimmerGraph example and extends it for use with multiple Shimmer devices. Those developing applications with multiple shimmer devices may want to consider using a service like approach similar to the example ShimmerGraphandLogService. The *MultiShimmerPlay* example within the unsupported folder would also be a good example to look at.

6.6. Broadcast Service and Receiver Example

An example service which can broadcast sensor data from one android application to another android application is provided. The example application on how to receive this broadcast is provided. The broadcast example is provided in the folder ShimmerBroadcastServiceExample. Example application of how to receive the broadcast is provided in the folder ShimmerBroadcastReceiverExample. Brief instructions on how to use both examples follows:-

1. First import ShimmerBroadcastandService onto Eclipse
2. Run the example using Eclipse. A screen shot of the app is shown below

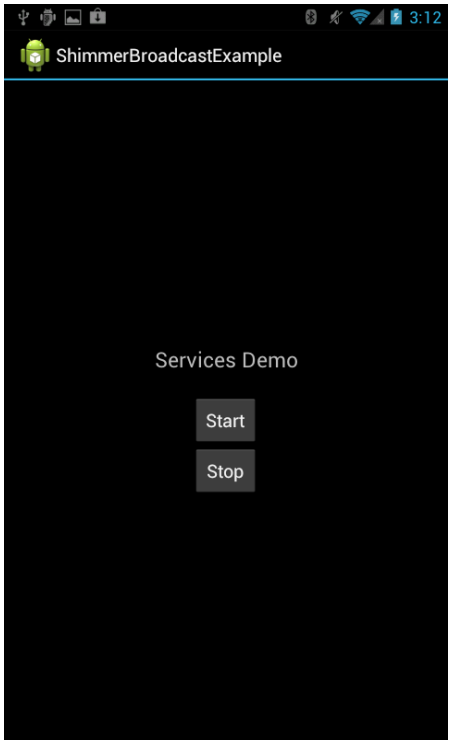
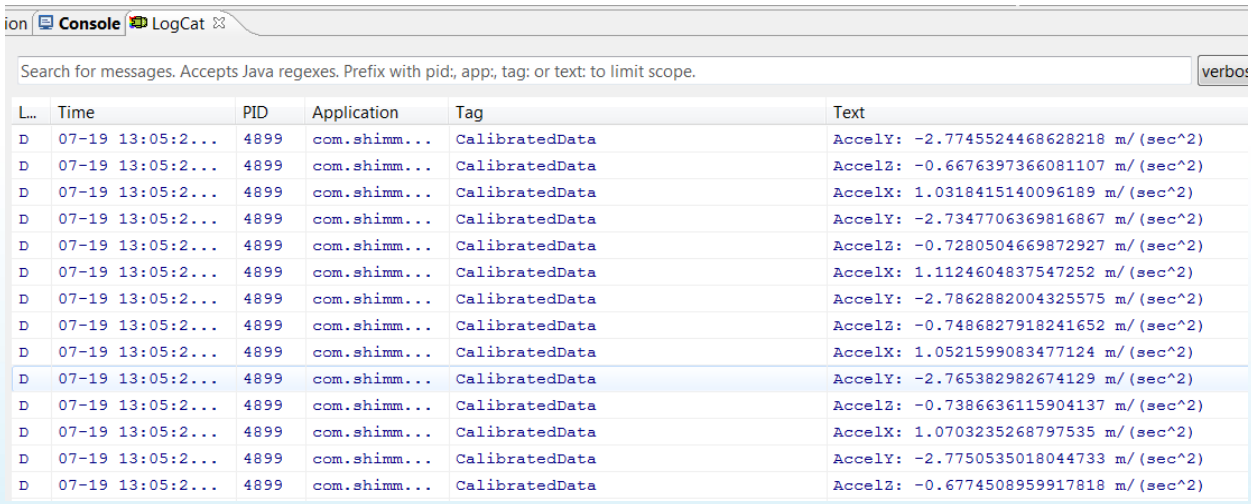


Figure 16: Screenshot of ShimmerBroadcastServiceExample

3. Once the app has been loaded, press start. **Error! Reference source not found.** shows the logcat once the Shimmer device has started streaming.



L...	Time	PID	Application	Tag	Text
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelY: -2.7745524468628218 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelZ: -0.6676397366081107 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelX: 1.0318415140096189 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelY: -2.7347706369816867 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelZ: -0.7280504669872927 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelX: 1.1124604837547252 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelY: -2.7862882004325575 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelZ: -0.7486827918241652 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelX: 1.0521599083477124 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelY: -2.765382982674129 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelZ: -0.7386636115904137 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelX: 1.0703235268797535 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelY: -2.7750535018044733 m/(sec^2)
D	07-19 13:05:2...	4899	com.shimm...	CalibratedData	AccelZ: -0.6774508959917818 m/(sec^2)

Figure 17: LogCat of ShimmerBroadcastServiceExample

4. Next import ShimmerBroadcastReceiverExample and run it. Figure 18**Error! Reference source not found.** shows the logcat when this application has been loaded. An additional

Tag named Received is seen in addition to the CalibratedData. Data is being passed from the Service to the receiver application via a broadcast. More information can be found here [2]

	Application	Tag	Text
.	com.shimmerresearch.service	CalibratedData	AccelY: -2.805183096957099 m/(sec^2)
.	com.shimmerresearch.service	CalibratedData	AccelZ: -0.7091562211952324 m/(sec^2)
.	com.shimmerresearch.receiver	Receiver	-0.7091562211952324
.	com.shimmerresearch.service	CalibratedData	AccelX: 0.9815561789881918 m/(sec^2)
.	com.shimmerresearch.service	CalibratedData	AccelY: -2.7442736820586737 m/(sec^2)
.	com.shimmerresearch.service	CalibratedData	AccelZ: -0.7185392996063936 m/(sec^2)
.	com.shimmerresearch.receiver	Receiver	-0.7185392996063936
.	com.shimmerresearch.service	CalibratedData	AccelX: 1.0000198311599897 m/(sec^2)
.	com.shimmerresearch.service	CalibratedData	AccelY: -2.7439462026376575 m/(sec^2)
.	com.shimmerresearch.service	CalibratedData	AccelZ: -0.66732770533299 m/(sec^2)
.	com.shimmerresearch.receiver	Receiver	-0.66732770533299
.	com.shimmerresearch.service	CalibratedData	AccelX: 1.0018624532534512 m/(sec^2)

Figure 18: LogCat of ShimmerBroadcastReceiverExample

5. The service can be stopped by returning to the ShimmerBroadcastServiceExample app and pressing the stop button.

A more elaborate example of the use of services can be found in the folder ShimmerGraphandLogService. Within the package com.shimmerresearch.service the file ShimmerService.java contains the implementation of the service.

6.7. ShimmerGraphandLogService

The shimmer graph and log service is an example of how to convert an application like Shimmer Graph into a service. Implementation of the service can be found in the ShimmerService.java file. The ShimmerGraph example initializes the Shimmer Object within the Activity as such:-

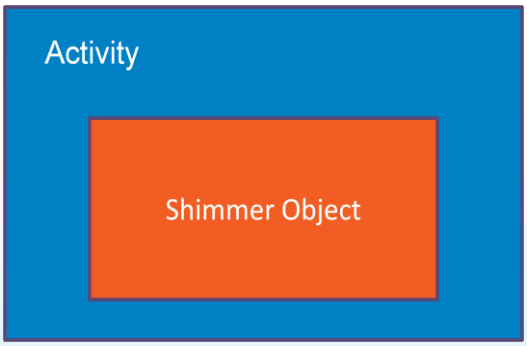


Figure 19: ShimmerGraph

As mentioned in the Section 5, due to the activity lifecycle the activity might be closed thus causing your connection to the Shimmer device to be lost. This example shows how to use a service to cope with long term operations of an Android application such as when you are logging data. Figure 20 shows a representative block diagram of the *ShimmerGraphandLogService* example.

[2] http://developer.android.com/reference/android/content/BroadcastReceiver.html

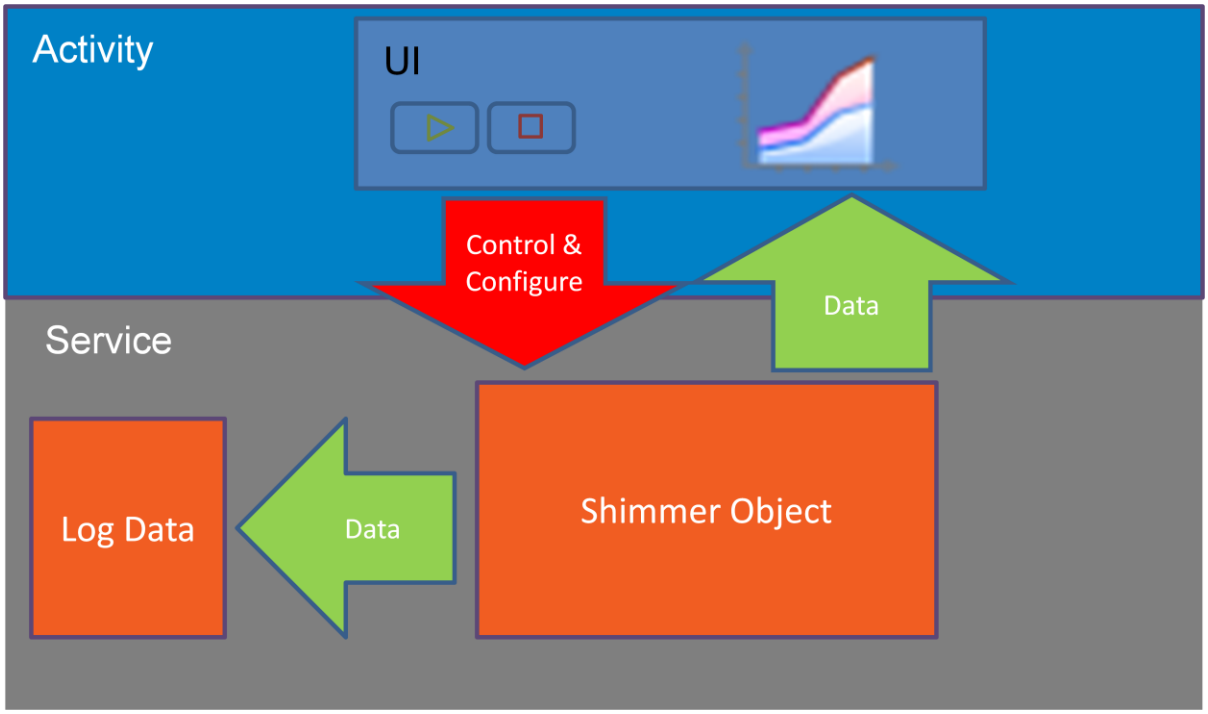


Figure 20: ShimmerGraphandLogService

Compared to Shimmer Graph, this application also has a variety of additional functionalities which are packet reception rate monitoring, battery monitoring, low battery notification and low power magnetometer. Low battery notification only works when BTStream (*firmware*) is used. As mentioned earlier, in order to use both battery monitoring and low battery notification on the Shimmer device, the battery voltage sensor has to be enabled prior to streaming. The low battery limit can be set via the Commands tab. By default the limit is set to 3.4 Volts.

There are six different sampling rates possible for the Magnetometer (0.5 Hz, 1.0 Hz, 2.0 Hz, 5.0 Hz, 10.0 Hz; 5 = 20.0 Hz; 6 = 50.0 Hz). The Magnetometer sampling rate is set to 10Hz when low power magnetometer is enabled, otherwise the sampling rate of the Magnetometer is set as close as possible to the current sampling rate of the Shimmer device.

Data is logged to the root directory of your external storage directory. The current time in milliseconds (at which the file was created) is appended to the filename you have specified. The file can be transferred to your PC via Bluetooth, USB or through the use of an application like DropBox.

An important thing to note is that while the example application only is limited to the use of a single Shimmer device, the service has been designed to be forward compatible for use with multiple Shimmer devices. Also see *MultiShimmerPlay* example located in the Unsupported Examples folder.

6.8. Shimmer3DOrientationExample

This example shows how to obtain the 3D orientation of the Shimmer device using the Accelerometer, Gyroscope and Magnetometer. The orientation algorithm used was proposed in the following paper:-

Madgwick, Sebastian OH, Andrew JL Harrison, and Ravi Vaidyanathan. "*Estimation of imu and marg orientation using a gradient descent algorithm*." Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on. IEEE, 2011.

In order to use this example users need to calibrate their Shimmer devices, using the Shimmer 9DoF Calibration Application. The Shimmer axis direction used should be the same as what is shown in Figure 21. For further information it is recommended that the user refer to the Shimmer 9DoF Calibration User Guide and Shimmer 9DoF Calibration tutorial video.

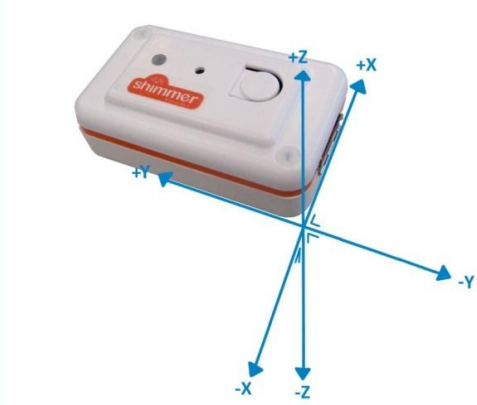


Figure 21: Shimmer Axis Direction

Once the device has started streaming position the Android device and Shimmer device as shown in the figure where both the Y Axis of the Android and Shimmer device are parallel and are pointing in the same direction. Once the cube representation of the Shimmer device has become static press the set button. Once pressed you should now see the orange side of the cube as shown in Figure 22.

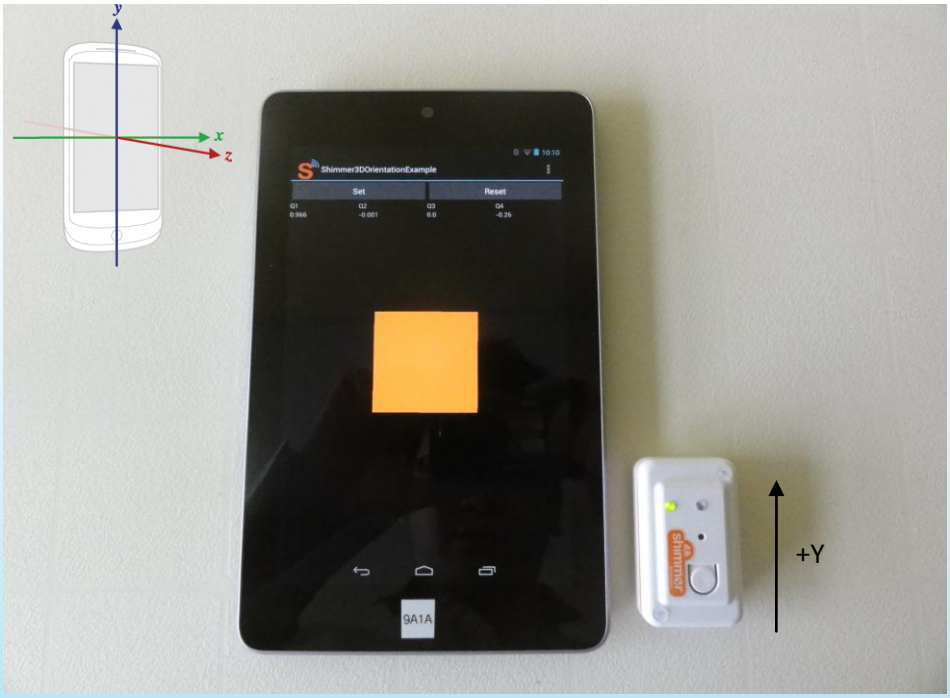


Figure 22: Front Representation of the Shimmer device

Turning the Shimmer device along the Y axis to the position shown in Figure 23 will result in the cube showing the green side.

Within this example functionality to enable Gyro On-The-Fly Calibration is provided. When enabled the offset vector of the gyroscope is recalculated whenever the Shimmer device is stationary. Implementation details of this can be found in the Shimmer.java file.

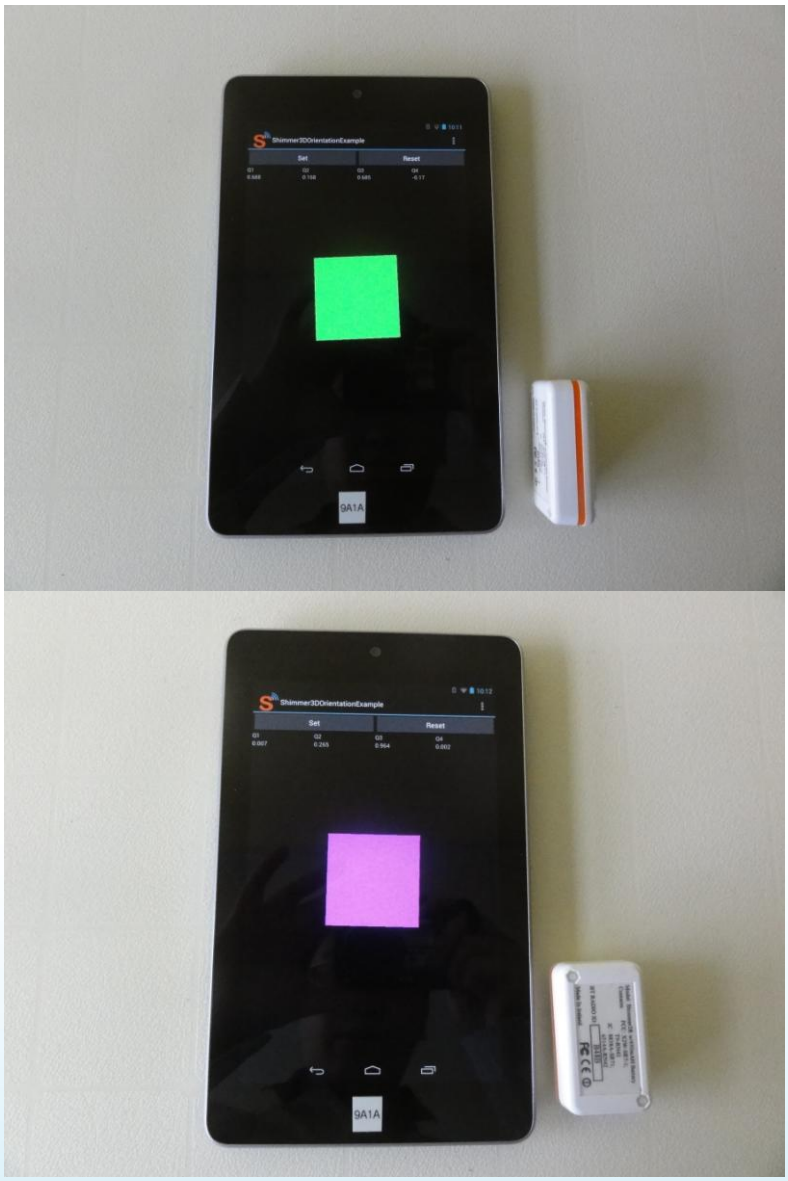


Figure 23: Side and Back 3D Representation of the Shimmer device

7. Usage Considerations

7.1. Battery Monitoring

From Android Beta 0.7 onwards battery monitoring can be done by using the *writeenabledsensors* command. Battery monitoring is done via the ADC used by ExpBoard A0 and ExpBoardaA7, thus while monitoring the battery you will be unable to use those ports. There is also a low battery warning functionality built in, which will cause the green LED of the Shimmer device to turn yellow when the voltage has dropped below a specified value. This value can be specified via the the command *setBattLimitWarning*. Users should note that battery monitoring is only supported on Shimmer devices which have the BTStream firmware installed on it. In order to use both battery monitoring and low battery notification on the Shimmer device, the battery voltage sensor has to be enabled prior to streaming. The low battery limit can be set via the Commands tab. By default the limit is set to 3.4 Volts. For an example application which uses the low battery indicator please refer to ShimmerGraphandLogService.

Users using Shimmer 2r/2/1 should note that enabling battery monitoring will cause crosstalk between the accelerometer/gyroscope sensor channels (z-axis) and one of the voltage measurements (VSenseBatt). This can influence the battery reading by 5-6% on average and up to 10% in extreme cases. This occurs because the battery resistor divider which is approximately 300kOhm leading to the ADC channel is not buffered by a voltage buffer. It is recommended where possible that the said kinematic sensors be disabled when measuring the battery voltage. When simultaneous streaming of both Accelerometer/Gyroscope and Battery Voltage is required the following are methods which might help mitigate the effects of crosstalk

- SW averaging (100 to 1000 counts) to the battery measurement as accel. signals are quasi-periodic assuming the user isn't in an elevator or vehicle
- VSenseBatt - n * Zaccel = VSenseBatt_correct where the scaling factor (n) would need to be determined experimentally
- Measuring battery voltage only when the device is in a static position

7.2. Bluetooth Connectivity

The reader should be aware that on occasion it might be difficult to make a successful connection to the Shimmer unit it due to the Bluetooth paging mechanism. This is explained below.

Paging Inquiry

- Firstly the page (and inquiry) window on the RN42 defaults to 320ms (out of a 2.56s). So the RN42 is only “listening” for a connection 12.5% of the time
- When a Bluetooth slave successfully connects or pairs with a BT master they synchronize their frequency hopping pattern. Using a FHS (frequency hopping synchronization) packet if a master has not connected to a slave over a long period of time then the frequency hopping pattern can get severely out of sync (and based on clock drift inside the RN42, this would explain why certain shimmers seem worse than others). Combine this the 12.5% duty cycle of the page window then it can result in needing multiple attempts to make a connection, but once the connecting is made the frequency hopping pattern resyncs.

8. Troubleshoot

Sometimes my device fails to connect to the Android device?

Please try the following:-

- Ensure the Shimmer device is fully charged
- Switch off your Shimmer device and the Bluetooth radio on your Android device, and then restart them.
- Unpair your Shimmer device and then pair them again

When I try to run the application on Eclipse I see the following error

```
[2012-06-15 13:28:28 - Shimmer] Installing Shimmer.apk...
[2012-06-15 13:28:30 - Shimmer] Re-installation failed due to different
application signatures.
[2012-06-15 13:28:30 - Shimmer] You must perform a full uninstall of the
application. WARNING: This will remove the application data!
[2012-06-15 13:28:30 - Shimmer] Please execute 'adb uninstall
shimmer.shimmergraph' in a shell.
[2012-06-15 13:28:30 - Shimmer] Launch canceled!
```

Launch the windows command prompt. Locate the folder platform-tools. Depending on where you have installed the SDK the directory will look something like this:-

C:\Program Files (x86)\Android\android-sdk\platform-tools>

Next execute the command which was requested, which in this case is 'adb uninstall shimmer.shimmergraph'. Alternatively uninstall the application first from the Android device.

When I try to run the application on Eclipse I see the following error

```
[2012-06-21 09:25:25 - ShimmerExample] ERROR: Application requires API version 10.
Device API version is 8 (Android 2.2).
[2012-06-21 09:25:25 - ShimmerExample] Launch canceled!
```

```
Edit AndroidManifest.xml

Change x to the appropriate value which matches your device

<uses-sdk android:minSdkVersion="x" />
```

My device fails to pair with the Android device

First ensure that you have the Boilerplate installed on your Shimmer device.

If that still fails, change the line:-

tmp = device.createRfcommSocketToServiceRecord(mSPP_UUID);

within the Shimmer class file to :-

tmp = device.createInsecureRfcommSocketToServiceRecord(mSPP_UUID);

Read http://code.google.com/p/android/issues/detail?id=15919 for explanation.

I see the error Conversion to Dalvik format failed with error x.

Ensure you do not have JAR files on your buildpath that include the same package and classes[3].

How can I retrieve the class files from the jar file

Download an extraction application such as WinRaR and extract the jar file.

[3] http://stackoverflow.com/questions/2680827/conversion-to-dalvik-format-failed-with-error-1-on-external-jar

Shimmer
The Realtime Building
Clonshaugh Technology Park
Dublin 17
Ireland
T: +353 (1)848 6112
E: info@shimmer-research.com

Shimmer North America
101 Tremont St, Suite 500
Boston
MA, 02108
USA
T: +1 857 362 7254
E: info@shimmer-research.com

www.Shimmer-Research.com
/ShimmerResearch
/ShimmerResearch
/company/Shimmer-Research
/ShimmerResearch
/ShimmerResearch