Shimmer C# API

User Manual

Rev 0.6b

Legal Notices and Disclaimer

Redistribution IS permitted provided that the following conditions are met:
Redistributions must retain the copyright notice, and the following disclaimer. Redistributions in electronic form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the document.

Neither the name of Shimmer Research, or Realtime Technologies Ltd. nor the names of its contributors may be used to endorse or promote products derived from this document without specific prior written permission.

THIS DOCUMENT IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENT, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Table of Contents

1. Introduction

The *Shimmer C# API* allows for real time data to be streamed from a Shimmer to C# via Bluetooth. It communicates with the Shimmer using Bluetooth Serial Port Profile. This API provides the fundamental building blocks for interacting with a Shimmer and acts as a platform for developers to create their own applications. It works with both Windows and Linux.

1.1. Pre-Requisites

1. A Shimmer 3 device programmed with *LogAndStream* firmware or a *Shimmer2/Shimmer2r* device programmed with the latest version of *BtStream.*

   NOTE: All Shimmers are shipped pre-programmed with the *LogAndStream* firmware. If for some reason you need to reprogram your Shimmer you can do so with Consensys software in the section 'Manage Devices'.

   The Shimmer needs to be paired with the PC (over Bluetooth). For Windows this procedure is explained in the quick start section of the Shimmer user manual. For details on how to do this in Linux see appendix at the end of this document.

2. **Windows:** Microsoft.NET Framework 4.5 is required. Most Vista and Windows 7 & 8 computers will already have this installed. If this is not installed it can be downloaded from http://www.microsoft.com/net/

   Linux: Mono Framework runtime is required. There are precompiled binaries for most popular Linux versions, or, if necessary, it can be installed from source. For more information see http://www.mono-project.com

3. Ensure you are using the latest version of the API and user guide by checking http://www.shimmersensing.com/support/wireless-sensor-networks-download/

2. Getting Started

First install *LogAndStream* firmware onto the Shimmer device. This can be done in Consensys software in the section 'Manage Devices'. See the Shimmer user manual for more details.

Download the latest version of *Shimmer C# API* from the Shimmer website http://www.shimmersensing.com/support/wireless-sensor-networks-download/.

Open the project solution (ShimmerCapture.sln).

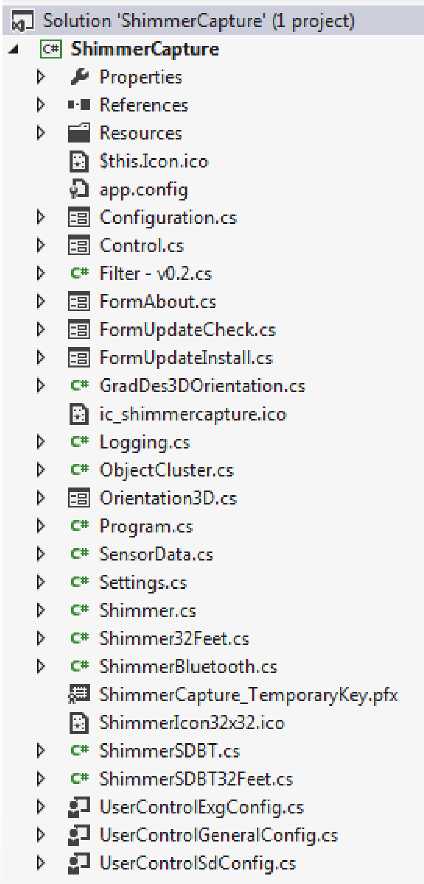


Figure 1: Project Files

The classes Shimmer.cs, ShimmerBluetooth.cs, Shimmer32Feet, ObjectCluster.cs, SensorData.cs and GradDes3DOrientation.cs make up core of the *Shimmer C# API*. The other classes make up the application '*ShimmerCapture*' which uses the API to communicate with the Shimmer.

The *Shimmer C# API* fully supports *LogAndStream* firmware.

3. Using the API

3.1. API Bluetooth Framework

The following shows the core framework of the Bluetooth API which communicates with the Shimmer device. The purpose of this design is to maintain flexibility and reusability. This design allows users to easily use different Bluetooth libraries.

Figure 2 shows an overview of the API Bluetooth Framework. The base class ShimmerBluetooth handles the interaction with *LogandStream* functionality while the extending classes implement the core Bluetooth Serial Port Profile calls. As shown that methods in italic are abstract methods (*Core Bluetooth Serial Port Profile Functions*) which must be implemented by the inheriting classes. This is demonstrated through the use of Shimmer (*which extends ShimmerBluetooth*) and implements the SerialPort calls within the abstract methods, as well as Shimmer32Feet which implements the abstract core Bluetooth Serial Port Profile calls through the use of the 32 Feet Library.[1]

[1] http://32feet.codeplex.com/

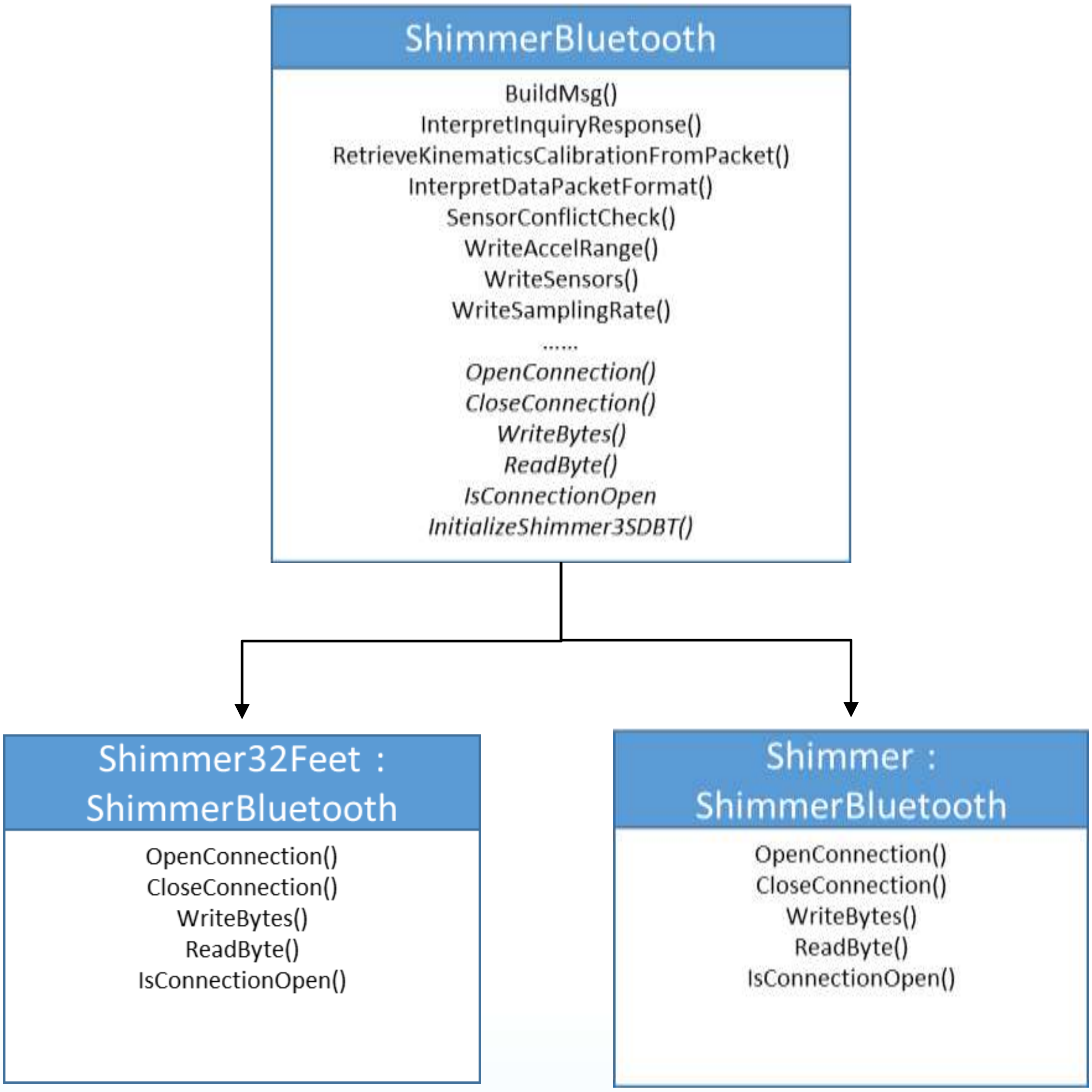


Figure 2: API Bluetooth Framework

3.2. Shimmer Class

Shimmer.cs and Shimmer32Feet.cs are the two main classes in the driver file. As the 32Feet library is not supported on all Bluetooth stacks the focus of the User guide will be on the Shimmer class. That being said it is recommended that users who are looking for advance Bluetooth functionality (e.g. connect via Bluetooth address, scan for Bluetooth devices) consider having a look at Shimmer32Feet.

Communication between the Shimmer and the API is handled directly through the serial port. Users require pairing the Shimmer devices and obtaining the serial port (com port) of the paired Shimmer device to use this class. This class provides the functionality to connect and disconnect to the Shimmer, to start and stop streaming and to set the configurations of the Shimmer.

Through a handler, messages can be sent from the API to the user interface. With these messages the API notifies the UI of a Shimmer state change, sends notification messages, sends data packets in real time when the Shimmer is in a streaming state and sends the Shimmer packet reception rate.

There are three Shimmer constructors in the API. One takes in two arguments; the device id and the COM port of the Shimmer. When using this constructor, all Shimmer configurations will be read from the Shimmer and stored in the class. The second constructor is for Shimmer3 and takes several arguments such as the sampling rate, the enabled sensors, the accelerometer range and the magnetometer range. These parameters are configured to the Shimmer once connected. The final constructor is for *Shimmer2r/Shimmer2* and is similar to the *Shimmer3* constructor but takes in fewer arguments. Again, the parameters declared in this constructor will be configured to the Shimmer once connected.

A connect thread is used to connect the API to a Shimmer. This connect thread has a timeout of 20 seconds. After this time, if no connection has been made, the API will close the serial port connection and return a message to the UI notifying it of this.

There are a range of 'Get' and 'Set' methods in this class. The 'Get' methods return a value stored in the class while the 'Set' methods change the value stored in the class. There are also a range of 'Read' and 'Write' methods. The 'Read' methods get the value stored in Shimmer memory while the 'Write' methods set the value stored in Shimmer memory. For a full list of these methods, see Table 3-3.

When enabling sensors on the Shimmer, a sensor conflict check is called. This checks whether the selected sensors can be enabled simultaneously or not. For example, if for Shimmer3, GSR and EXG1 24Bit are both selected, this check will return false and the sensors will not be enabled. In this case, whichever sensors were enabled previously will remain enabled.

3.2.1. Shimmer-API-UI Communication

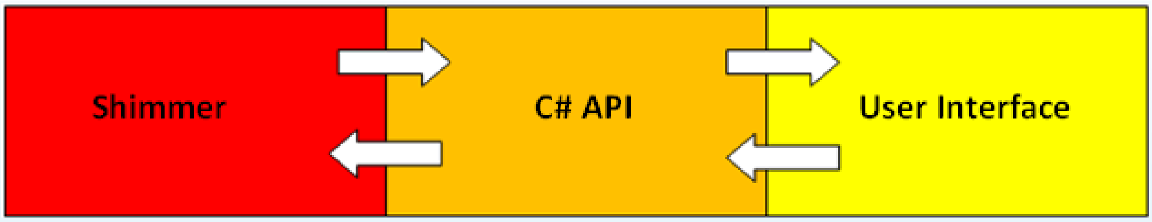


Figure 3: Shimmer-API-UI Communication Path

The Shimmer communicates with the API through a series of responses which are processed in the method 'ReadData'. This method is called via a thread. Each Shimmer response has a unique identifier which allows the API to process the response accordingly. One such response is *'MSG_IDENTIFIER_DATA_PACKET'* which indicates that the Shimmer is transmitting sensor data to the API. This response is only processed if the Shimmer is in a streaming state. In this case, a BuildMsg method is called which reads in the streaming data and creates an ObjectCluster to hold the data (see Section 3.3 for information about ObjectCluster). This is then sent to the UI by the following code snippet;

```
CustomEventArgs newEventArgs = new
        CustomEventArgs((int)ShimmerIdentifier.MSG_IDENTIFIER_DATA_PACKET,
        (object)KeepObjectCluster);
OnNewEvent(newEventArgs);
```

ShimmerIdentifier.MSG_IDENTIFIER_DATA_PACKET is an integer value which acts as an indicator to inform the UI as to what type of message is being transmitted (state change, data packet or packet reception rate). The second variable, KeepObjectCluster, is an ObjectCluster returned by '*BuildMsg*' which contains the sensor data. '*CustomEventArgs*' is a class that contains these two variables; the indicator and the ObjectCluster. This class is used by the handler to send the indicator and the necessary information to the UI.

The ObjectCluster returned by the method BuildMsg is only sent to the UI if there has been no packet loss. To check for packet loss, the ObjectCluster is stored in the variable 'KeepObjectCluster'. If the following indicator sent through the read thread is again a data packet indicator, this KeepObjectCluster will be sent to the UI and the process will be repeated. However, if the next indicator is anything other than the data packet indicator (this is the only indicator that should be sent while the Shimmer is in a streaming state), it means that there has been packet loss. In this case, the KeepObjectCluster will be discarded and will not be sent to the UI. Once the API has recovered from this packet loss, valid data will once again be sent to the UI. In this way, no invalid data should be sent to the UI.

The following is an example how to interface the callbacks with the UI :-

```
shimmer.UICallback += this.HandleEvent;
public void HandleEvent(object sender, EventArgs args)

  {

     CustomEventArgs eventArgs = (CustomEventArgs)args;

     int indicator = eventArgs.getIndicator();



     switch (indicator)

     {

        case (int)ShimmerBluetooth.ShimmerIdentifier.MSG_IDENTIFIER_STATE_CHANGE:

       ...
```

3.2.2. Signal Properties

As previously mentioned, the method 'BuildMsg' creates an ObjectCluster from the bytes received through the serial port. The list of properties contained in the ObjectCluster is illustrated in Table 3-1 List of Shimmer3 Properties (*Shimmer3*) and Table 3-2 (*Shimmer2r/Shimmer2*). The calibrated data does not have a channel identifier because calibration is performed in the API and not on the Shimmer. If the calibrated data units contain an asterisk, default calibration parameters have been used. Otherwise, calibration parameters stored on the Shimmer have been used to convert the calibrated data into the raw data.

List of supported properties	Format	Unit	Description	Inquiry Response Channel Identifier
Timestamp	RAW	No Unit	Raw timestamp data is generated from the crystal oscillator on the Shimmer which has a frequency of 32768 Hz. Signal Data Type U12 which loop around 0 when it exceeds 65536	None
Low Noise Accelerometer X	RAW	No Unit	Signal Data Type u12	0x00
Low Noise Accelerometer Y	RAW	No Unit	Signal Data Type u12	0x01
Low Noise Accelerometer Z	RAW	No Unit	Signal Data Type u12	0x02
VSenseBatt	RAW	No Unit	Signal Data Type u12	0x03
Wide Range Accelerometer X	RAW	No Unit	Signal Data Type i16	0x04
Wide Range Accelerometer Y	RAW	No Unit	Signal Data Type i16	0x05
Wide Range Accelerometer Z	RAW	No Unit	Signal Data Type i16	0x06
Magnetometer X	RAW	No Unit	Signal Data Type i16r	0x07
Magnetometer Y	RAW	No Unit	Signal Data Type i16r	0x08
Magnetometer Z	RAW	No Unit	Signal Data Type i16r	0x09
Gyroscope X	RAW	No Unit	Signal Data Type i16r	0x0A
Gyroscope Y	RAW	No Unit	Signal Data Type i16r	0x0B
Gyroscope Z	RAW	No Unit	Signal Data Type i16r	0x0C
External ADC A7	RAW	No Unit	Signal Data Type u12	0x0D
External ADC A6	RAW	No Unit	Signal Data Type u12	0x0E
External ADC A15	RAW	No Unit	Signal Data Type u12	0x0F
Internal ADC A1	RAW	No Unit	Signal Data Type u12	0x10
Internal ADC A12	RAW	No Unit	Signal Data Type u12	0x11
Internal ADC A13	RAW	No Unit	Signal Data Type u12	0x12
Internal ADC A14	RAW	No Unit	Signal Data Type u12	0x13
Pressure	RAW	No Unit	Signal Data Type u24r	0x1B
Temperature	RAW	No Unit	Signal Data Type u16r	0x1A
GSR	RAW	No Unit	Signal Data Type u16	0x1C

EXG1 Status	RAW	No unit	Signal Data Type u8	0x1D
EXG2 Status	RAW	No unit	Signal Data Type u8	0x20
ECG LL-RA	RAW	No unit	When EXG1 and EXG2 is enabled (16/24 bit) and EXG configuration is set to default ECG, see *enableDefaultECGConfiguration*	0x1E
ECG LA-RA	RAW	No unit	When EXG1 and EXG2 is enabled (16/24 bit) and EXG configuration is set to default ECG, see *enableDefaultECGConfiguration*	0x1F
EMG CH1	RAW	No unit	When EXG1 and EXG2 is enabled (16/24 bit) and EXG configuration is set to default EMG, see *enableDefaultEMGConfiguration*	0x1E
EMG CH2	RAW	No unit	When EXG1 and EXG2 is enabled (16/24 bit) and EXG configuration is set to default EMG, see *enableDefaultEMGConfiguration n*	0x1F
EXG1 CH1	RAW	No unit	When EXG1 is enabled, and default ECG/EMG configuration is not used. Signal Data Type i24r	0x1E
EXG1 CH2	RAW	No unit	When EXG1 is enabled, and default ECG/EMG configuration is not used. Signal Data Type i24r	0x1F
ECG Vx-RL	RAW	No unit	When EXG2 is enabled, and default EXG configuration is set to ECG, see *enableDefaultECGConfiguration*	0x22
EXG2 CH1	RAW	No unit	When EXG2 is enabled. Signal Data Type i24r	0x21
EXG2 CH2	RAW	No unit	When EXG2 is enabled and default ECG configuration is not used. Signal Data Type i24r	0x22
EXG1 CH1 16 Bit	RAW	No unit	When EXG1 16 bit is enabled, and default ECG/EMG configuration is not used. Signal Data Type i16r	0x23
EXG1 CH2 16Bit	RAW	No unit	When EXG1 16 bit is enabled, and default ECG/EMG configuration is not used. Signal Data Type i16r	0x24
EXG2 CH1 16 Bit	RAW	No unit	When EXG2 16 bit is enabled, and default ECG/EMG configuration is not used. Signal Data Type i16r	0x25
EXG2 CH2 16Bit	RAW	No unit	When EXG2 16 bit is enabled, and default ECG/EMG configuration is not used. Signal Data Type i16r	0x26
Bridge Amplifier High	RAW	No unit	Signal Data Type u12	0x27
Bridge Amplifier Low	RAW	No unit	Signal Data Type u12	0x28
Timestamp	CAL	mSecs		None
Low Noise	CAL	m/(sec^2)	For best performance calibrate using	None

Accelerometer X			*Shimmer 9DoF Calibration Application*	
Low Noise Accelerometer Y	CAL	m/(sec^2)	See *Shimmer 9DoF Calibration Application*	None
Low Noise Accelerometer Z	CAL	m/(sec^2)	See *Shimmer 9DoF Calibration Application*	None
Wide Range Accelerometer X	CAL	m/(sec^2)	See *Shimmer 9DoF Calibration Application*	None
Wide Range Accelerometer Y	CAL	m/(sec^2)	See *Shimmer 9DoF Calibration Application*	None
Wide Range Accelerometer Z	CAL	m/(sec^2)	See *Shimmer 9DoF Calibration Application*	None
Gyroscope X	CAL	deg/sec	See *Shimmer 9DoF Calibration Application*	None
Gyroscope Y	CAL	deg/sec	See *Shimmer 9DoF Calibration Application*	None
Gyroscope Z	CAL	deg/sec	See *Shimmer 9DoF Calibration Application*	None
Magnetometer X	CAL	local	See *Shimmer 9DoF Calibration Application*	None
Magnetometer Y	CAL	local	See *Shimmer 9DoF Calibration Application*	None
Magnetometer Z	CAL	local	See *Shimmer 9DoF Calibration Application*	None
VSenseBatt	CAL	mVolts		None
External ADC A7	CAL	mVolts		None
External ADC A6	CAL	mVolts		None
External ADC A15	CAL	mVolts		None
Internal ADC A1	CAL	mVolts		None
Internal ADC A12	CAL	mVolts		None
Internal ADC A13	CAL	mVolts	Also used for PPG sensor (GSR+ board)	None
Internal ADC A14	CAL	mVolts		None
Pressure	CAL	kPa		None
Temperature	CAL	Celsius		None
GSR	CAL	kOhms	Conductance is the inverse of this value	None
EXG1 Status	CAL	mVolts		None
EXG2 Status	CAL	mVolts		None
ECG LL-RA	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None
ECG LA-RA	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None
EMG CH1	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None
EMG CH2	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None

EXG1 CH1	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None
EXG1 CH2	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None
ECG Vx-RL	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None
EXG2 CH1	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None
EXG2 CH2	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None
EXG1 CH1 16 Bit	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None
EXG1 CH2 16Bit	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None
EXG2 CH1 16 Bit	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None
EXG2 CH2 16Bit	CAL	mVolts	Depends on EXG register configuration, see comments on RAW	None
Bridge Amplifier High	CAL	mVolts		None
Bridge Amplifier Low	CAL	mVolts		None
Quaternion 0	CAL	local	3D Orientation must be enabled using the following method enable3DOrientation()	None
Quaternion 1	CAL	local	see enable3dOrientation()	None
Quaternion 2	CAL	local	see enable3dOrientation()	None
Quaternion 3	CAL	local	see enable3dOrientation()	None
Angle Axis A	CAL	local	see enable3dOrientation()	None
Angle Axis X	CAL	local	see enable3dOrientation()	None
Angle Axis Y	CAL	local	see enable3dOrientation()	None
Angle Axis Z	CAL	local	see enable3dOrientation()	None

Table 3-1 List of Shimmer3 Properties

List of supported properties	Format	Unit	Description	Inquiry Response Channel Identifier
Timestamp	Raw	No Unit	Raw timestamp data is generated from the crystal oscillator on the Shimmer which has a frequency of 32768 Hz. Signal Data Type U12 which loop around 0 when it exceeds 65536	None
Accelerometer X	Raw	No Unit	Signal Data Type u12	0
Accelerometer Y	Raw	No Unit	Signal Data Type u12	1
Accelerometer Z	Raw	No Unit	Signal Data Type u12	2

Gyroscope X	Raw	No Unit	Signal Data Type u12	3
Gyroscope Y	Raw	No Unit	Signal Data Type u12	4
Gyroscope Z	Raw	No Unit	Signal Data Type u12	5
Magnetometer X	Raw	No Unit	Signal Data Type i16	6
Magnetometer Y	Raw	No Unit	Signal Data Type i16	7
Magnetometer Z	Raw	No Unit	Signal Data Type i16	8
ECG RA LL	Raw	No Unit	Signal Data Type u12	9
ECG LA LL	Raw	No Unit	Signal Data Type u12	0A
GSR	Raw	No Unit	Signal Data Type u16, two MSBs signify the range, 12 bits only contain data	0B
EMG	Raw	No Unit	Signal Data Type u12	0D
ExpBoard A0	Raw	No Unit	Signal Data Type u12	0E
ExpBoard A7	Raw	No Unit	Signal Data Type u12	0F
Strain Gauge High	Raw	No Unit	Signal Data Type u12	10
Strain Gauge Low	Raw	No Unit	Signal Data Type u12	11
Heart Rate	Raw	No Unit	Signal Data Type u16, this is not to be confused with ECG and is not derived from the ECG module	12
VSenseReg	Raw	No Unit	PMux must be switched on, which switches the mux input used for expboardA0 to the battery	0E
VSenseBatt	Raw	No Unit	PMux must be switched on, which switches the mux input used for expboardA7 to the battery	0F
Timestamp	CAL	mSecs		None
Accelerometer X	CAL	m/(sec^2)	For best performance calibrate using *Shimmer 9DOF Calibration Application*	None
Accelerometer Y	CAL	m/(sec^2)	See *Shimmer 9DOF Cal* App	None
Accelerometer Z	CAL	m/(sec^2)	See *Shimmer 9DOF Cal* App	None
Gyroscope X	CAL	deg/sec	See *Shimmer 9DOF Cal* App	None
Gyroscope Y	CAL	deg/sec	See *Shimmer 9DOF Cal* App	None
Gyroscope Z	CAL	deg/sec	See *Shimmer 9DOF Cal* App	None
Magnetometer X	CAL	local	See *Shimmer 9DOF Cal* App	None
Magnetometer Y	CAL	local	See *Shimmer 9DOF Cal* App	None
Magnetometer Z	CAL	local	See *Shimmer 9DOF Cal* App	None
GSR	CAL	kOhms	Conductance is the inverse of this value	None
ECG RA LL	CAL	mVolts		None

ECG LA LL	CAL	mVolts		None
EMG	CAL	mVolts		None
Strain Gauge High	CAL	mVolts		None
Strain Gauge Low	CAL	mVolts		None
Heart Rate	CAL	BPM	Does not work with ECG, requires the extension radio module, and a polar heart rate strap	None
ExpBoard A0	CAL	mVolts		None
ExpBoard A7	CAL	mVolts		None
VSenseReg	CAL	mVolts		None
VSenseBatt	CAL	mVolts		None
Quaternion 0	CAL	local	3D Orientation must be enabled using the following method enable3DOrientation()	None
Quaternion 1	CAL	local	see enable3dOrientation()	None
Quaternion 2	CAL	local	see enable3dOrientation()	None
Quaternion 3	CAL	local	see enable3dOrientation()	None
Angle Axis A	CAL	local	see enable3dOrientation()	None
Angle Axis X	CAL	local	see enable3dOrientation()	None
Angle Axis Y	CAL	local	see enable3dOrientation()	None
Angle Axis Z	CAL	local	see enable3dOrientation()	None

Table 3-2 List of Shimmer2r/Shimmer2 Properties

3.3. Object Cluster Class

This class defines the data structure being sent from Shimmer.cs to the UI when data is being streamed from the Shimmer. It contains the device ID, the Shimmer COM port, the names, units and format (raw or calibrated) of the streaming signals as well as the signal data. There are four ObjectCluster constructors. The first of these takes in two arguments; the Shimmer COM Port and the device ID. The second takes in the Shimmer COM Port, the device ID, the list of signal names, the list of signal formats and the list of units for the respective signals. The third constructor takes in the same arguments as the aforementioned with the addition of the data for each signal. The final constructor takes in an ObjectCluster as its only argument and creates a new ObjectCluster. This is used to prevent an ObjectCluster being a reference to another ObjectCluster. The method Add is used to add a signal name, format, unit and data point to their respective lists stored in the ObjectCluster while the method AddData is used to add a data point to the data list. Finally, there are a range of Get methods which can be used to extract data from the ObjectCluster, as described below.

3.3.1. Extracting data from the ObjectCluster

Within this class there are 'Get' methods to allow for the extraction of the data held in the ObjectCluster. There is a 'GetIndex' that returns the index of the specified signal name and signal format. There are two 'GetData' methods. One takes in an index as an argument and returns the corresponding data and units (in the form of SensorData - see Section 3.4). The second takes in the signal name and format and again returns the data and units corresponding to the specified inputs. Users should note that the signal names and formats used to find the index or data must be identical to those used in 'BuildMsg'. Examples below demonstrate the extraction of the raw x axis low noise accelerometer data from the object cluster;

Method 1:
double xAccel = (objectCluster.GetData("Low Noise Accelerometer X", "RAW")).GetData();
Method 2:
int index = objectCluster.GetIndex("Low Noise Accelerometer X", "RAW");
double xAccel = (objectCluster.GetData(index)).GetData();

3.4. Sensor Data Class

SensorData.cs defines a new data structure which contains two variables; unit and data. This structure is used within the ObjectCluster to extract sensor data through the sensor name and format. The example above demonstrates the use of this class. The code '*objectCluster.GetData("Low Noise Accelerometer X", "RAW")*' returns an object called 'SensorData'. To get either the data or the units from this, 'GetData()' or 'GetUnit()' must follow as demonstrated above.

3.5. Shimmer API Commands

Table 3-3 lists the API commands that can be called from another class to control the Shimmer.

Command	**Description**
Connect	Connects to the specified COM port
Disconnect	Disconnects from the specified COM port
GetAccelRange	Returns the accelerometer range stored in the class
GetAccelSamplingRate	Returns the accelerometer sampling rate stored in the class
GetComPort	Returns the COM port stored in the class
GetConfigSetupByte0	Returns the configuration byte 0 stored in the class
GetDeviceID	Returns the device id. stored in the class
GetEnabledSensors	Returns the enabled sensors stored in the class (Integer)
GetEXG1RegisterContents	Returns the ExG chip1 register contents stored in the class
GetEXG1RegisterByte	Returns the specified ExG Chip1 register byte stored in the class

GetEXG2RegisterContents	Returns the ExG chip2 register contents stored in the class
GetEXG2RegisterByte	Returns the specified ExG Chip2 register byte stored in the class
GetExpansionBoard	Returns the expansion board stored in the class
GetFirmwareInternal	Returns the internal firmware stored in the class
GetFirmwareVersion	Returns the firmware version stored in the class
GetFirmwareVersionFullName	Returns the full firmware version name stored in the class
GetBaudRate	Returns the baud rate of the Bluetooth module stored in the class
GetGSRRange	Returns the GSR range stored in the class
GetGyroRange	Returns the gyroscope range stored in the class
GetInternalExpPower	Returns whether or not the internal exp power is enabled
GetMagRange	Returns the magnetometer range stored in the class
GetMagSamplingRate	Returns the magnetometer sampling rate stored in the class
GetPMux	Returns the PMux bit stored in the class
GetPressureResolution	Returns the pressure resolution stored in the class
GetSamplingRate	Returns the sampling rate stored in the class
GetShimmerVersion	Returns the Shimmer hardware version
GetState	Returns the state of the Shimmer (Integer)
GetStateString	Returns the state of the Shimmer (String)
GetVReg	Returns the five volt regulator bit stored in the class
Inquiry	Sends an inquiry command to the Shimmer
Is3DOrientationEnabled	Returns whether or not default 3D orientation is enabled (Boolean)
IsConnected	Returns whether or not the Shimmer is connected (Boolean)
IsDefaultECGConfigurationEnabled	Returns whether or not default ECG configuration is enabled (Boolean)

IsDefaultEMGConfigurationEnabled	Returns whether or not default EMG configuration is enabled (Boolean)
IsGyroOnTheFlyCalEnabled	Returns whether or not gyroscope in use calibration is enabled (Boolean)
ReadAccelRange	Gets the accelerometer range stored on the Shimmer
ReadAccelSamplingRate	Gets the accelerometer sampling rate stored on the Shimmer
ReadBaudRate	Gets the baud rate of the Bluetooth module stored on the Shimmmer
ReadBlinkLED	Gets the LED status of the Shimmer
ReadCalibrationParameters	Gets the calibration parameters stored on the Shimmer
ReadConfigByte0	Gets the configuration byte 0 stored on the Shimmer
ReadEXGConfigurations	Gets the ExG configurations stored on the Shimmer
ReadExpansionBoard	Gets the expansion board (if any) attached to the Shimmer
ReadGSRRange	Gets the GSR range stored on the Shimmer
ReadGyroCalibration	Gets the gyroscope calibration parameters stored on the Shimmer
ReadGyroRange	Gets the gyroscope range stored on the Shimmer
ReadInternalExpPower	Gets whether or not the internal exp power of the Shimmer is enabled
ReadMagCalibration	Gets the magnetometer calibration parameters stored on the Shimmer
ReadMagRange	Gets the magnetometer range stored on the Shimmer
ReadMagSamplingRate	Gets the magnetometer sampling rate stored on the Shimmer
ReadPressureCalibrationCoefficients	Gets the pressure calibration coefficients stored on the Shimmer
ReadSamplingRate	Gets the sampling rate of the Shimmer
Set3DOrientation	Enables/disables 3D orientation
SetAccelRange	Sets the accelerometer range stored in the class

SetAccelSamplingRate	Sets the accelerometer sampling rate stored in the class
SetBaudRate	Sets the baud rate stored in the class
SetConfigSetipByte0	Sets the configuration byte 0 stored in the class
SetEXG1RegisterContents	Sets the EXG chip 1 register bytes stored in the class
SetEXG2RegisterContents	Sets the EXG chip 2 register bytes stored in the class
SetGSRRange	Sets the GSR range stored in the class
SetGyroOnTheFlyCalibration	Enables/disables gyro in use calibration
SetGyroRange	Sets the gyroscope range stored in the class
SetInternalExpPower	Enables/disables internal exp power
SetLowPowerAccel	Enables/disables low power accelerometer
SetLowPowerGyro	Enables/disables low power gyroscope
SetLowPowerMag	Enables/disables low power magnetometer
SetMagRange	Sets the magnetometer range stored in the class
SetMagSamplingRate	Sets the magnetometer sampling rate stored in the class
SetPMux	Sets the PMux bit stored in the class
SetPressureCalibration	Sets the pressure calibration parameters stored in the class
SetState	Sets the state of the Shimmer
SetVReg	Sets the five volt regulator bit stored in the class
StartStreaming	Sends start streaming command to the Shimmer
StopStreaming	Sends stop streaming command to the Shimmer
ToggleLED	Sends command to the Shimmer to toggle the red LED
Write5VReg	Writes the five volt regulator bit on the Shimmer
WriteAccelRange	Writes the accelerometer range on the Shimmer
WriteBaudRate	Writes the baud rate of the Bluetooth module on the Shimmer
WriteBufferSize	Writes the buffer size to the Shimmer

WriteEXGConfigurations	Writes the ExG configurations to the Shimmer
WriteGSRRange	Writes the GSR range on the Shimmer
WriteGyroRange	Writes the gyroscope range on the Shimmer
WriteGyroSamplingRate	Writes the gyroscope sampling rate on the Shimmer
WriteInternalExpPower	Enables/disables the internal exp power
WriteMagRange	Writes the magnetometer range on the Shimmer
WriteMagSamplingRate	Writes the magnetometer sampling rate on the Shimmer
WritePMux	Writes the PMux bit to the Shimmer
WritePressureResolution	Writes the pressure resolution to the Shimmer
WriteSamplingRate	Writes the sampling rate to the Shimmer
WriteSensors	Enables/disables the sensors on the Shimmer
WriteWRAccelSamplingRate	Writes the wide range accelerometer sampling rate to the Shimmer (Shimmer3 only)

Table 3-3 Shimmer API Commands

4. ShimmerCapture

ShimmerCapture is an example application that uses the *Shimmer C# API*. The application allows the user to connect to a single Shimmer, to configure a range of parameters and to stream data in real time to a PC. The streaming data can be displayed on a graph and written to file. It focuses on the basic functionality of the Shimmer to allow users to quickly get started with their Shimmer and acts as a stepping stone towards more advanced applications. *ShimmerCapture* works equally well on both Windows and Linux.

Users should note that for both Windows and Linux, the files 'ShimmerPPGtoHR.dll', 'ZedGraph.dll', 'OpenTK.GLControl.dll', 'OpenTK.Compatibility.dll' and 'OpenTK.dll' need to be in the same directory as the executable file.

The 'Control' class contains a HandleEvent method which demonstrates how the application interacts with the API handler. It is here that the UI is informed of a Shimmer state change and incoming data packets are processed. Everytime a message is sent through the UICallback EventHandler of the API, it is received and processed by this HandleEvent method.

In the references folder there are a number of .dll files. The ZedGraph.dll file is the graphing library used for *ShimmerCapture*, the ShimmerPPGtoHR.dll file is the PPG to heart rate algorithm and the OpenTK .dll files are the graphical library files needed for the 3D orientation cube example.

For more information, see the *ShimmerCapture* user guide, available for download at http://www.shimmersensing.com/support/wireless-sensor-networks-download/.

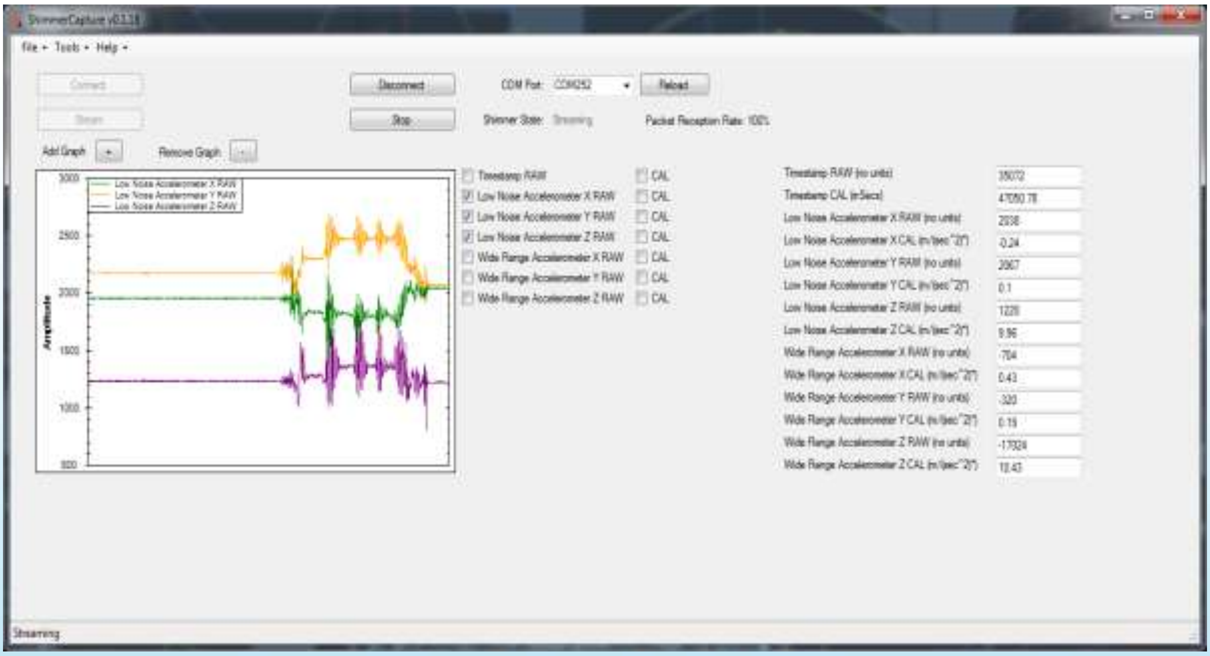


Figure 4 ShimmerCapture user interface

5. Appendix

5.1. Linux

The operation of pairing a Shimmer device in Linux to be used with *ShimmerCapture* might vary from distribution to distribution. The following procedure was tested in Ubuntu 10.04 (including ShimmerLive 10.04) , Slackware 13, and OpenSuse 11.3.

Requirements:

A working Bluetooth radio and the BlueZ Bluetooth libraries and tools need to be installed. See http://www.bluez.org for details.

Procedure:

All the commands given here should be entered from the command line (in a terminal window).

1. Ensure the Bluetooth radio is available by running the “hciconfig” command.

   E.g.:

   ```
   tiny2@ShimmerLive:~/Desktop$ hciconfig
   hci0:    Type: USB
   BD Address: 00:19:0E:0A:D6:62 ACL MTU: 1021:8 SCO MTU: 64:1
   UP RUNNING PSCAN
   RX bytes:1013 acl:0 sco:0 events:34 errors:0
   TX bytes:1347 acl:0 sco:0 commands:34 errors:0
   ```

2. Scan for the Shimmer by running “hcitool scan”.

   E.g. :

   ```
   tiny2@ShimmerLive:~/Desktop$ hcitool scan
   Scanning ...
           00:06:66:42:22:BD       RN42-22BD
           00:A0:96:28:DF:E8       FireFly-DFE8
           00:06:66:42:24:18       RN42-2418
   ```

3. To use the shimmer2r named RN42-2418 it must be bound to an rfcomm device. The “rfcomm bind <n> <MAC_ADDRESS>” command achieves this. The <n> gives the rfcomm device number, which must be different for each Shimmer paired, and the <MAC_ADDRESS> is the Shimmers MAC address, as can be seen from the “hcitool scan” output above. This command normally needs root privileges, so “sudo” is used.

   E.g.:

   ```
   tiny2@ShimmerLive:~/Desktop$ sudo rfcomm bind 0 00:06:66:42:24:18
   [sudo] password for tiny2:
   ```

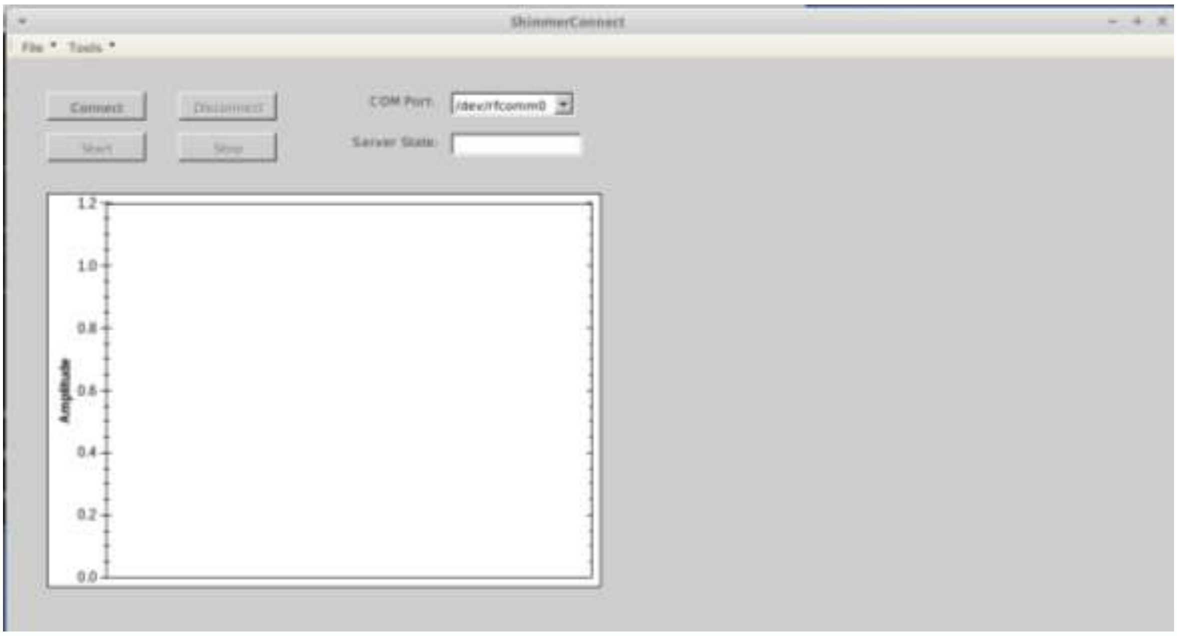
4. Running the “rfcomm” command with no arguments shows which Shimmer is bound to which rfcomm device, along with the current connection status.

   E.g.:

   ```
   tiny2@ShimmerLive:~/Desktop$ rfcomm
   rfcomm0: 00:06:66:42:24:18 channel 1 clean
   rfcomm1: 00:A0:96:28:DF:E8 channel 1 clean
   ```

5. To connect to the Shimmer in *ShimmerCapture* enter “/dev/rfcomm<n>” in the “Select COM port” field.

   E.g. to connect to Shimmer RN42-2418 which is bound to rfcomm0 as above:



5.2. Quaternion calculation algorithm

The algorithm used to calculate quaternion from the accelerometer, gyroscope and magnetometer sensors 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.

Shimmer International Offices:
Europe – Dublin, Ireland.
USA – Boston, MA.
Asia – Kuala Lumpur, Malaysia.

Web: www.ShimmerSensing.com
Email: info@ShimmerSensing.com

www.Shimmersensing.com
/ShimmerResearch
@ShimmerSensing
/company/Shimmer
/ShimmerSensing
/ShimmerResearch