wiigee

Accelerometerbased
3D Gesture Recognition

– Manual –

Benjamin Poppinga
benjamin@wiigee.org

Version 1.0 **DRAFT**
November 5, 2009

Contents

1 Introduction

2 Architecture

3 Requirements

The core of wiigee, wiigee-lib, is written device independent and therefore does not have any additional requirements beside of Java[1] in a version greater or equal to 1.6. wiigee-lib only needs acceleration data and discrete events to control the training and recognition process. Both, acceleration and discrete events like e.g. button presses, can be either simulated or generated online by any device. In general every device capable of sending these information is supported by wiigee-lib. A developer only has to make sure that the device can be connected and the corresponding data can be obtained. As the name 'wiigee' already suggests, wiigee-lib has been developed for Nintendos Wii Remote, which is usually named Wiimote.

3.1 Wiimote: `wiigee-plugin-wiimote`

Nintendos Wiimote usually comes with the popular gaming console Wii, but can also be obtained separately for about $35 (valid on July, 31th 2009). As required by wiigee-lib, the Wiimote serves acceleration data with a frequency of $100Hz$ and discrete events in form of button presses. The Wiimote serves this data using Bluetooth. Of course Nintendo did not release any information about how to connect to the Wiimote. Every information about the communication protocol has been reverse engineered[2]. The wiigee-plugin-wiimote implements a part of this reverse engineered information to obtain acceleration and button data from the Wiimote.

3.1.1 Bluetooth Device

As already mentioned the Wiimote uses Bluetooth. Therefore your device where the wiigee-lib is running on needs a Bluetooth dongle or an internal Bluetooth chip. To reach a good data throughput, especially with the frequent acceleration events, consider to get a Bluetooth 2.0 dongle with enhanced data rate (EDR).

3.1.2 Bluetooth Stack

The most important thing for wiigee is the used Bluetooth stack. This is the software, which actually controls your Bluetooth device. Common stacks on Windows are WIDCOMM[3], BlueSoleil[4] and the default Windows Bluetooth stack. Linux computers usually uses the BlueZ[5] Bluetooth stack. Apple's OS X already comes with an integrated Bluetooth stack.

[1] Sun's Java Website: `http://java.sun.com/`, last access November 5, 2009
[2] Wiibrew: `http://www.wiibrew.org/`, last access November 5, 2009
[3] Broadcomm Website: `http://www.broadcom.com/`, last access November 5, 2009
[4] Bluesoleil Website: `http://www.bluesoleil.com/`, last access November 5, 2009
[5] BlueZ Website: `http://www.bluez.org/`, last access November 5, 2009

3.1.3 JSR-82 Implementation

The JSR-82 implementation actually lets Java know what Bluetooth is and for example how to create connections. Originally specified by a consortium of multiple device manufacturers, there now exists multiple different JSR-82 implementations, like e.g. BlueCove[6] or Avetana[7]. Since the Wiimote uses a specific Bluetooth subprotocol it is necessary that the JSR-82 implementation supports L2CAP for the used Bluetooth stack.

3.1.4 Recommendations

Ok, among us there are not too many working configurations. If you still have the possibility to choose your Bluetooth device consider to buy one which comes with the WIDCOMM Bluetooth stack. This would help you to get wiigee-plugin-wiimote working on Windows. On Linux or OS X systems you could buy any compatible stack. By experience I would recommend the BlueCove JSR-82 implementation, which worked very well for me already on WIDCOMM, BlueZ and the OS X stack.

3.2 Android Phone: `wiigee-plugin-android`

While the wiimote is a separate device and wiigee's logic runs on a computer, wiigee can also run on a platform equipped with an accelerometer itself. Google's Android is such a Java-enabled platform which wiigee out-of-the-box supports. Any other mobile phones with Java and Accelerationsensor can be adopted easily. Since the Android Developers changed something within the API the wiigee-plugin-android is only working on Android Devices with a firmware greater or equal to Android 1.5.

[6] BlueCove Website: `http://www.bluecove.org/`, last access November 5, 2009
[7] Avetana Website: `http://www.avetana-gmbh.de/avetana-gmbh/jsr82.xml`, last access November 5, 2009

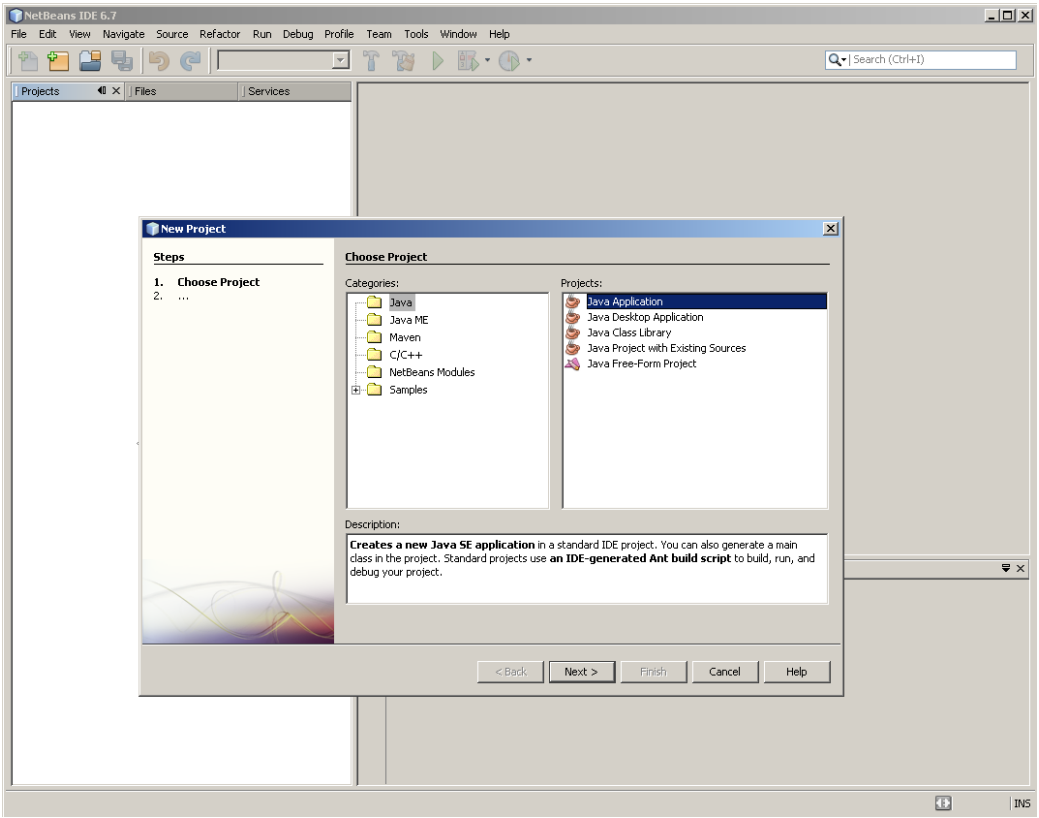
4 Integrating wiigee into a Java Project

Integrating wiigee into your Java project is easy. After wiigee's JAR file and at least one plugin has been integrated into the classpath, a `GestureListener` has to be setup being responsible for received gestures within your application. The following lines describe, how wiigee can be integrated using the well known Netbeans IDE[1]. Obviously first of all a copy of wiigee is needed, which contains of *wiigee-lib-1.5.6.jar* and in case of the Wiimote complemented by *wiigee-plugin-wiimote-1.5.6.jar*. As already mentioned, a JSR-82 implementation is needed: BlueCove for example *bluecove-2.1.0.jar*.

4.1 Netbeans 6.7

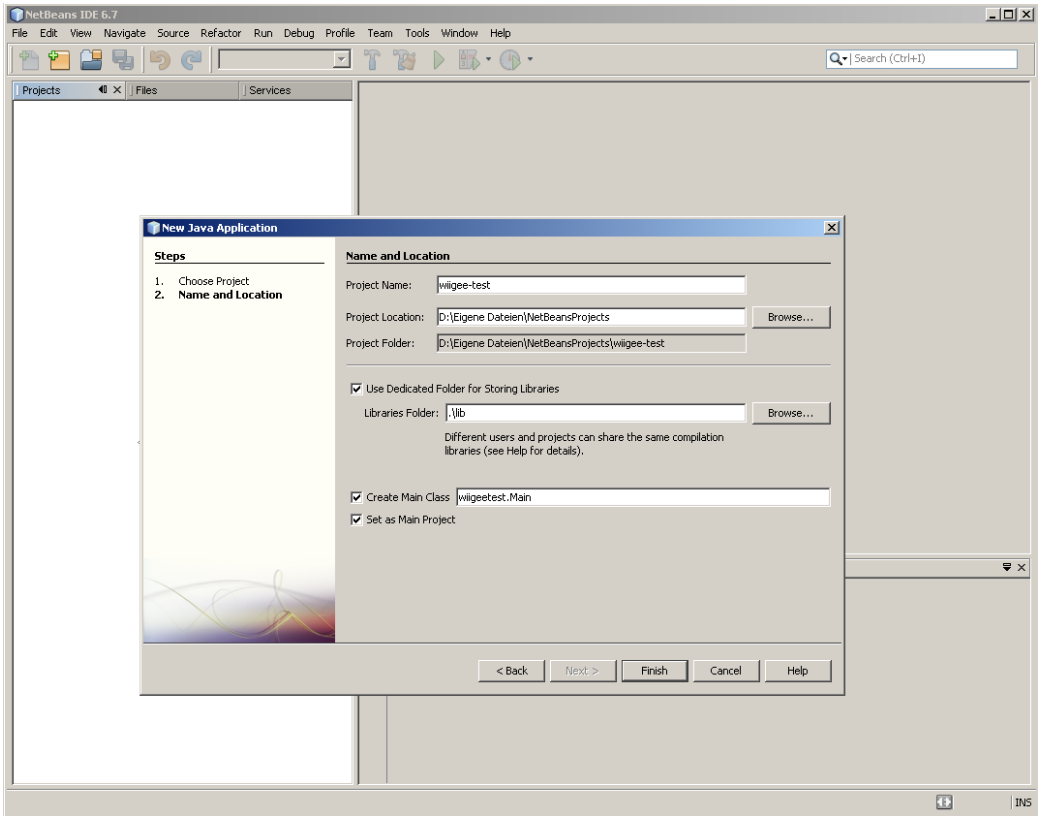
4.1.1 Create a new Java Project

Choose **File** and select **New Project**. Then select that a **Java Application** should be created and click **Next**.



[1] Netbeans Website: `http://www.netbeans.org/`, last access November 5, 2009

Choose a **Project Name** like "'wiigee-test"', select **Use Dedicated Folder for Storing Libraries** and click **Finish**. Remember the folder where you create your new project.



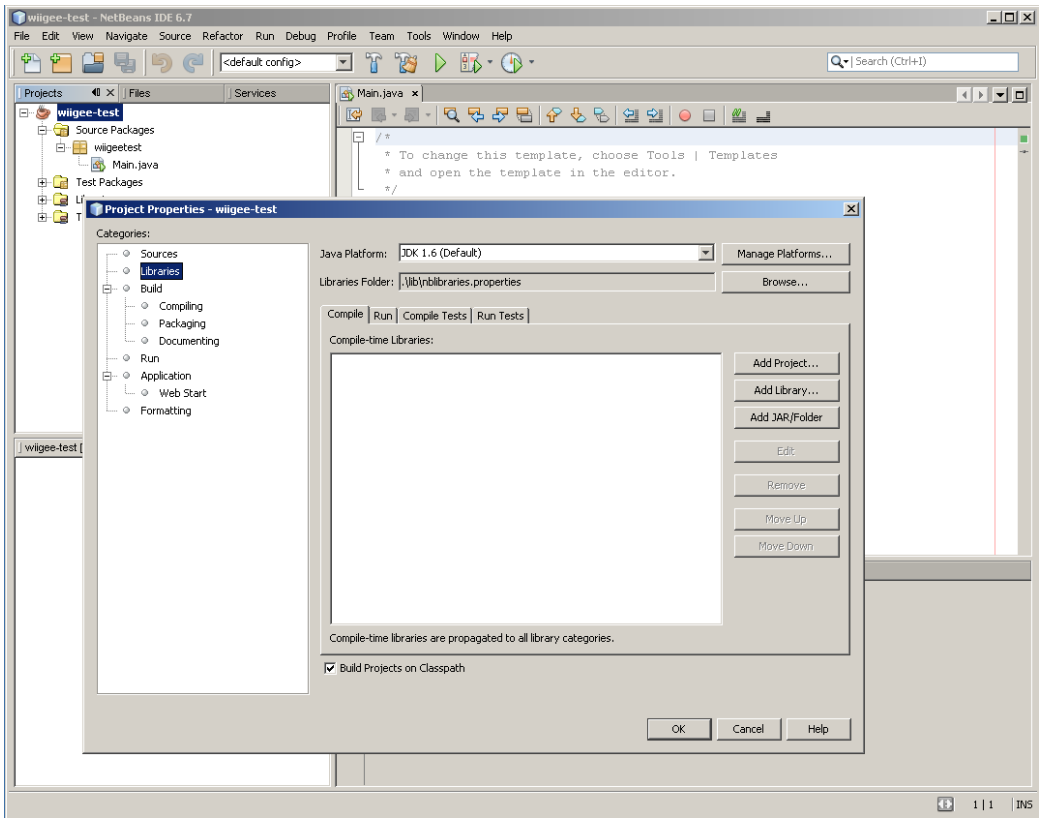
4.1.2 Integrate wiigee

Right click on your project name on the left and select **Properties**. Then select **Libraries** and click **Add JAR/Folder**, since multiple JAR files should be added.

In the popping up file selector, **change to the path** where you've downloaded wiigee, a wiigee plugin and your JSR 82 implementation. **Select** the JAR file and select **Copy to Libraries Folder**. This allows you to delete the downloaded files afterwards, since a copy of them now exists within your project folder. Finally click **Open**. Repeat this procedure for every JAR file or select multiple libraries at once. wiigee is now known to your project and can be used.

4.1.3 Implement GestureListener

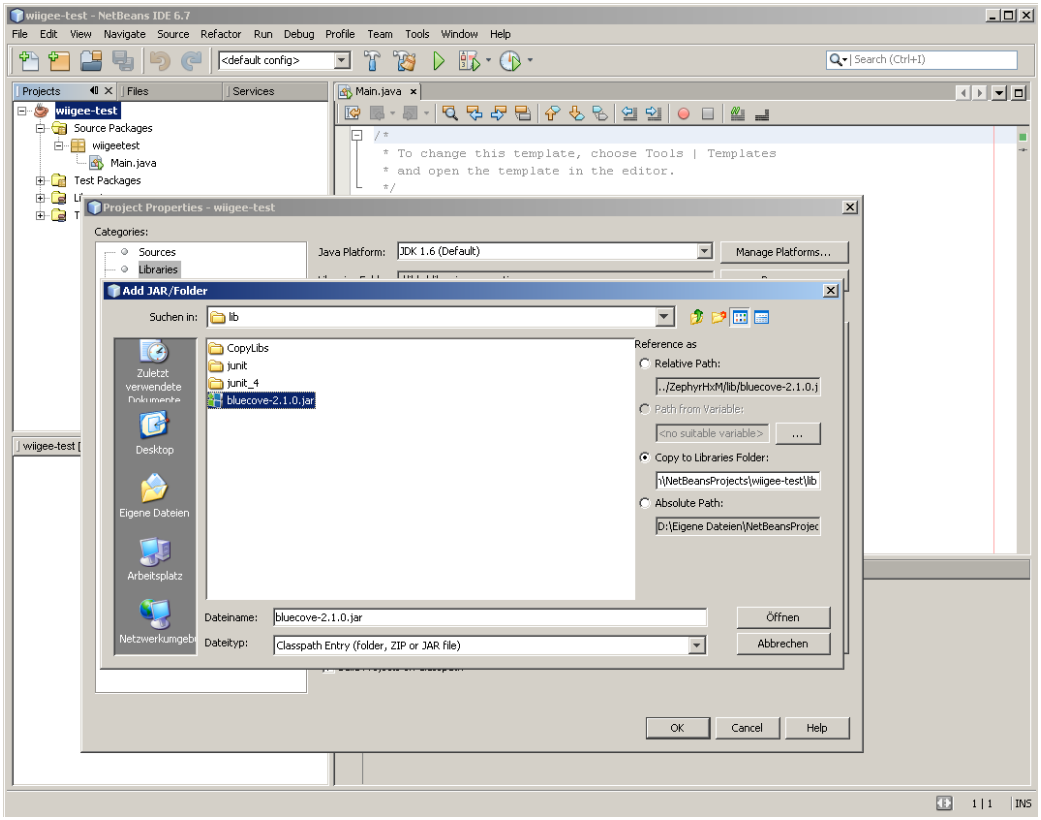
To get information about recognized gestures, the `GestureListener` has to be implemented by at least one class. To do so, create a new Java class, for example named `GestureRetriever`. To implement the `GestureListener` add it to the class descriptor as shown in Figure 4.1.3. Finally implement the defined method `gestureReceived` within this class. There you can do the actual gesture handling. This is the method



which would be called in case a gesture has been recognized. As example the determined Gesture ID is simply printed out for each gesture.

4.2 Eclipse 3.X

TODO


wiigee-test - NetBeans IDE 6.7
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects
Files
Services
wiigee-test
Source Packages
wiigeetest
Main.java
Test Packages
Main.java
Project Properties - wiigee-test
Categories:
Sources
Libraries
Java Platform:
JDK 1.6 (Default)
Manage Platforms...
Add JAR/Folder
Suchen in:
lib
CopyLibs
junit
junit_4
bluecove-2.1.0.jar
Reference as
Relative Path:
../ZephyrHxM/lib/bluecove-2.1.0.j
Path from Variable:
<no suitable variable>
Copy to Libraries Folder:
:\NetBeansProjects\wiigee-test\lib
Absolute Path:
D:\Eigene Dateien\NetBeansProjec
Desktop
Eigene Dateien
Arbeitsplatz
Dateiname:
bluecove-2.1.0.jar
Öffnen
Dateityp:
Classpath Entry (folder, ZIP or JAR file)
Abbrechen
OK
Cancel
Help

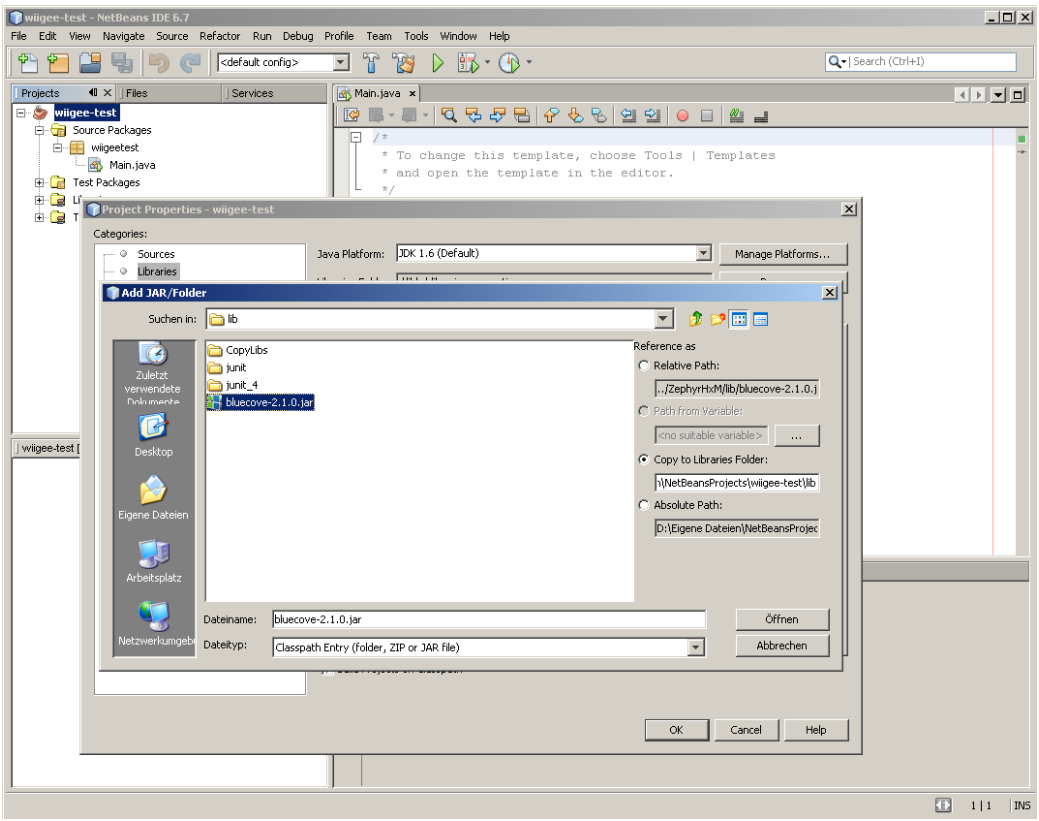


Figure 4.1: TODO

5 Using wiigee

After wiigee is set up and an application is implemented according to the previous chapter, the library can be trained with a set of gestures. The training has to be done after wiigee has started. There exist two methods for training gestures: performing the gestures while the application is running or loading pretrained gestures during startup of the application from a file.

5.1 Training of Gestures

To train a gesture during the runtime of the application press and hold the training button, which in case of the wiimote is **A**. While holding the button perform the indented gesture. When finished release the button and repeat this training procedure for at least five times to increase the recognition reliability. Finally press the gesture conclusion button to conclude this type of a gesture. In case of the Wiimote this button is **Home**. The whole procedure needs to be repeated for every gesture type which should be trained.

5.2 Recognition of Gestures

To recognize a gesture some gestures already has to be trained. Start the recognition process by pressing down and holding the recognition button. In case of the Wiimote the recognition button would be the button **B**. If a gesture has been recognized, every `GestureListener` would be called immediatly.

5.3 Save a Gestureset

It is possible to save a set of trained gestures by calling the *saveGestures* of the Device instance after the gestures has been trained. The resulting files are a WGS (wiigee gesture set) file and a TXT file for each gesture type belonging to this gesture set.

5.4 Load a Gestureset

Instead of training gestures dynamically everytime the application starts, already trained gestures can be saved to a file and trained again without any user interaction. To load a set of gestures call the *loadGestures* method a Device instance with the according path to a WGS file, which has been saved by wiigee.

6 Settings

6.0.1 Button Events

6.0.2 Acceleration

6.0.3 Rotation

6.0.4 Infrared

7 Example: Gesture Controlled Photo Slideshow

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum.

List of Figures

List of Tables