Introduction to liburbi Java for Urbi 1.x

(book compiled from 682M)

Remi Humbert

Introduction to liburbi Java for Urbi 1.x: (book compiled from 682M)

by Remi Humbert

Publication date

Table of Contents

Chapter 1. Introduction

Liburbi-java is a library designed to encapsulate an URBI connection. It handles the TCP connection with the URBI server, and the dispatching of messages it sends. The library is thread-safe and reentrant. This library is generated from the Liburbi-c++ source with swig.

The library consists of two main Java classes, `liburbi.UClient` and `liburbi.USyncClient`, and a few helpful functions.

We expect the reader to be a bit familiar with the URBI syntax.

Chapter 2. Getting started

Loading liburbijava shared library

Liburbi Java is a library generated from the liburbi C++ with the tool called swig [http://www.swig.org/]. Swig generates wrapper classes in java for all the C++ classes we want to export. It also generates JNI code to make the link between Java and the C++ library. With this in mind, you understand that in order to use the Liburbi Java, you have to somehow load the C++ JNI code in your application. We provide this C++ code in a shared library called urbijava.dll under Windows, liburbijava.so under Linux, and liburbijava.dylib under MacOs. To load the library in your java code, use the function `System.loadLibrary`. For example place this call in your Main class as showed bellow:

```
// All liburbi classes are located in the package liburbi
import liburbi.main.UClient;
import liburbi.main.liburbi;

public class Main {

  static {
    System.loadLibrary("urbijava");
  }
```

NB: The path to the library you are loading 'urbijava' must be in your path for your program to find it. On Windows make sure the 'path' environment variable contains the path to the urbijava library. On Unix make sure that your LD_LIBRARY_PATH contains the path to the urbijava library.

Connecting

To connect to an URBI server, simply create a new instance of `liburbi.main.UClient` (or `liburbi.main.USyncClient` if you want to use the synchronous functions described bellow), passing the name or the address of the server as the first parameter, and optionnaly the port as the second parameter:

```
  public static void main (String[] arg) {

    UClient client = new UClient("myrobot.ensta.fr");

    // a wrapper is also available in the liburbi.main.liburbi class:
    // UClient client = liburbi.connect("myrobot.ensta.fr");
```

The constructor will start an independant thread that will listen for incoming messages from the URBI server.

You can check if the connection was successfuly established by calling the `error` function, which returns a zero value on success, or a nonzero error code in case of failure.

```
    if (client.error() != 0)
    {
 System.err.println("Couldn't connect to the URBI server.");
```

```
         return;
            }
```

Sending URBI commands

The method `send` is the simplest way to send commands to the URBI server. It take a String as parameter:

```
    int sleeptime = 50;

    client.send("motoron;");

    for (float val=0; val<=1; val+=0.05)
      client->send("neck.val = " + val + ";wait (" + sleeptime ");");
```

Some static functions are defined in the class liburbi.liburbi: 'getComma ()', 'getSemicolon()', 'getPipe()' and 'getParallel()' and they return the characters ',', ';', '|' and '&' respectively.

Sending binary data.

To send binary data to the robot, the method `sendBin` must be used. It takes as parameters the buffer to send (we only support byte[] type for now) and its size, and optionnaly a header.

```
    client->sendBin(soundData, soundDataSize,
                    "speaker.val = BIN " + soundDataSize + " raw 2 16000 16 1;"
```

Sending a sound

Although you could use `sendBin` to play a sound on the robot, a specific and efficient method has been written for this purpose: `sendSound`.

```
client->sendSound("speaker", sound, "endsound");
```

The first parameter is the sound device to which we send the sound, the second parameter is an `liburbi.main.USound` class describing the sound to send. The third is an optionnal tag that will be used by the server to issue a "stop" system message when the sound has finished playing. The function `convert` (in liburbi.liburbi class) can be used to convert between various sound formats.

There is no limit to the size of the sound buffer, since it will be automatically cut into small chunks by the library. The data is copied by the library: the USound parameter and its associated data can be safely freed as soon as the function returns.

Chapter 3. Receiving

Tags and callbacks

Most of the messages received from the URBI server are the results of a previously sent command. The mechanism of URBI tags enables to link a message to its reply: with each command is associated a tag, and this tag is repeated in the reply message. The `liburbi.main.UClient` class handles the reception of those messages in the independant thread created by the constructor, parses them and fills a UMessage structure.

In your Java program, you can have classes extending `liburbi.main.UCallbackInterface` and redefining its method `onMessage`. You can the register these classes with the `UClient` method `setCallback` and associate them with a tag. Then each time a message with this tag is sent by the server, the redefined `onMessage` function will be called with a `liburbi.main.UMessage` class as a parameter.

The `liburbi.main.UCallbackInterface` class is defined as follow:

```
package liburbi.main;

public class UCallbackInterface {
  public UCallbackAction onMessage(UMessage msg);
}
```

The `onMessage` function take as parameter a `liburbi.main.UMessage` and return an enum: `liburbi.main.UCallbackAction` which can have two values: `liburbi.main.UCallbackAction.URBI_CONTINUE` or `liburbi.main.UCallbackAction.URBI_REMOVE` (in which case the callback is removed, and wont be called anymore, even if you receive a message with the same tag).

The `setCallback` function is defined as follow:

```
package liburbi.main;

public class UClient {

  [...]
  public long setCallback(UCallbackInterface ref, String tag);
}
```

Give as first parameter your class redefining `liburbi.main.UCallbackInterface`, and give as second argument a `String` corresponding to the tag you want to associated the callback with.

UMessage

The UMessage structure is capable of storing the informations contained in any kind of URBI message by using a "getType" function and an UValue (union of classes). These two classes are defined as follows:

```
package liburbi.main;

public enum UMessageType {
  MESSAGE_SYSTEM, /// System message
  MESSAGE_ERROR,  /// Error message
  MESSAGE_DATA;   /// Message containing data in the UValue
}

public class UMessage {

  /// Server-side timestamp.
  public int getTimestamp();

  /// Associated tag.
  public String getTag();

  /// Type of the UValue contained in the UMessage
  public UMessageType getType();

  /// UValue contained in the UMessage
  public UValue getValue();

  public String getMessage();

  /// Raw message without the binary data.
  public String getRawMessage();

  /// Client from which originated the message.
  public UAbstractClient getClient();
}
```

The type field UMessageType can be MESSAGE_SYSTEM, MESSAGE_ERROR or MESSAGE_DATA. If the type is MESSAGE_DATA, the message contains an UValue.

UValue

```
package liburbi.main;

public enum UDataType {
  DATA_DOUBLE,
  DATA_STRING,
  DATA_BINARY,
  DATA_LIST,
  DATA_OBJECT,
  DATA_VOID;
}

public class UValue {

  public UDataType getType();

  /// value if of type UDataType.DATA_STRING
  /// check that string is not null (it can be in the C++ side,
  /// remember liburbijava is build on top of liburbic++)
  /// with 'isStringNull ()'
  public String getString();
  public boolean isStringNull();

  /// value if of type UDataType.DATA_DOUBLE
  public double getDouble();
  /// value if of type UDataType.DATA_BINARY
  public UBinary getUBinary();

  /// value if of type UDataType.DATA_LIST
  public UList getUList();

  /// Used to print the UValue conveniently
  public String toString();
}
```

The UValue contains an UDataType which can take the values: DATA_DOUBLE, DATA_STRING, DATA_BINARY, DATA_LIST, DATA_OBJECT, DATA_VOID. Depending of this field, the corresponding value in the UValue will be set. If the UValue is of binary type, it contains an UBinary class defined hereafter. The UBinaryType in the UBinary structure will give additional informations on the type of data (BINARY_NONE, BINARY_UNKNOWN, BINARY_IMAGE, BINARY_SOUND), and the appropriate sound or image structure will be filled.

UBinary

```
package liburbi.main;

public enum UBinaryType {
  BINARY_NONE,
  BINARY_UNKNOWN,
  BINARY_IMAGE,
  BINARY_SOUND;
}

public class UBinary {

  public UBinaryType getType();

  /// value if of type BINARY_IMAGE
  public UImage getUImage();

  /// value if of type BINARY_SOUND
  public USound getUSound();

  /// Size of the data
  public int getSize();

  public String getMessage();
  public String getExtraHeader();
}
```

USound

```
package liburbi.main;

public enum USoundFormat {
  SOUND_RAW,
  SOUND_WAV,
  SOUND_MP3,
  SOUND_OGG,
  SOUND_UNKNOWN;
}

public enum USoundSampleFormat {
  SAMPLE_SIGNED(1),
  SAMPLE_UNSIGNED(2);
}

public class USound {

  /// total size in byte
  public long getSize();

  /// number of audio channels
  public int getChannels();

  /// rate in Hertz
  public int getRate();

  /// sample size in bit
  public int getSampleSize();

  /// format of the sound data
  /// (SOUND_RAW, SOUND_WAV, SOUND_MP3...)
  public USoundFormat getSoundFormat();

  /// sample format
  public USoundSampleFormat getSampleFormat();

  /// sound data in an array of byte
  public byte[] getDataAsByte();
}
```

UImage

```
package liburbi.main;

public enum UImageFormat {
  IMAGE_RGB(1),
  IMAGE_YCbCr(2),
  IMAGE_JPEG(3),
  IMAGE_PPM(4),
  IMAGE_UNKNOWN;
}

public class UImage {

  /// image size in byte
  public long getSize();

  /// image width
  public long getWidth();
  /// image height
  public long getHeight();

  /// IMAGE_RGB, IMAGE_YCbCr, IMAGE_JPEG...
  public UImageFormat getImageFormat();

  /// image data in an array of byte
  public byte[] getDataAsByte();
}
```

A few examples

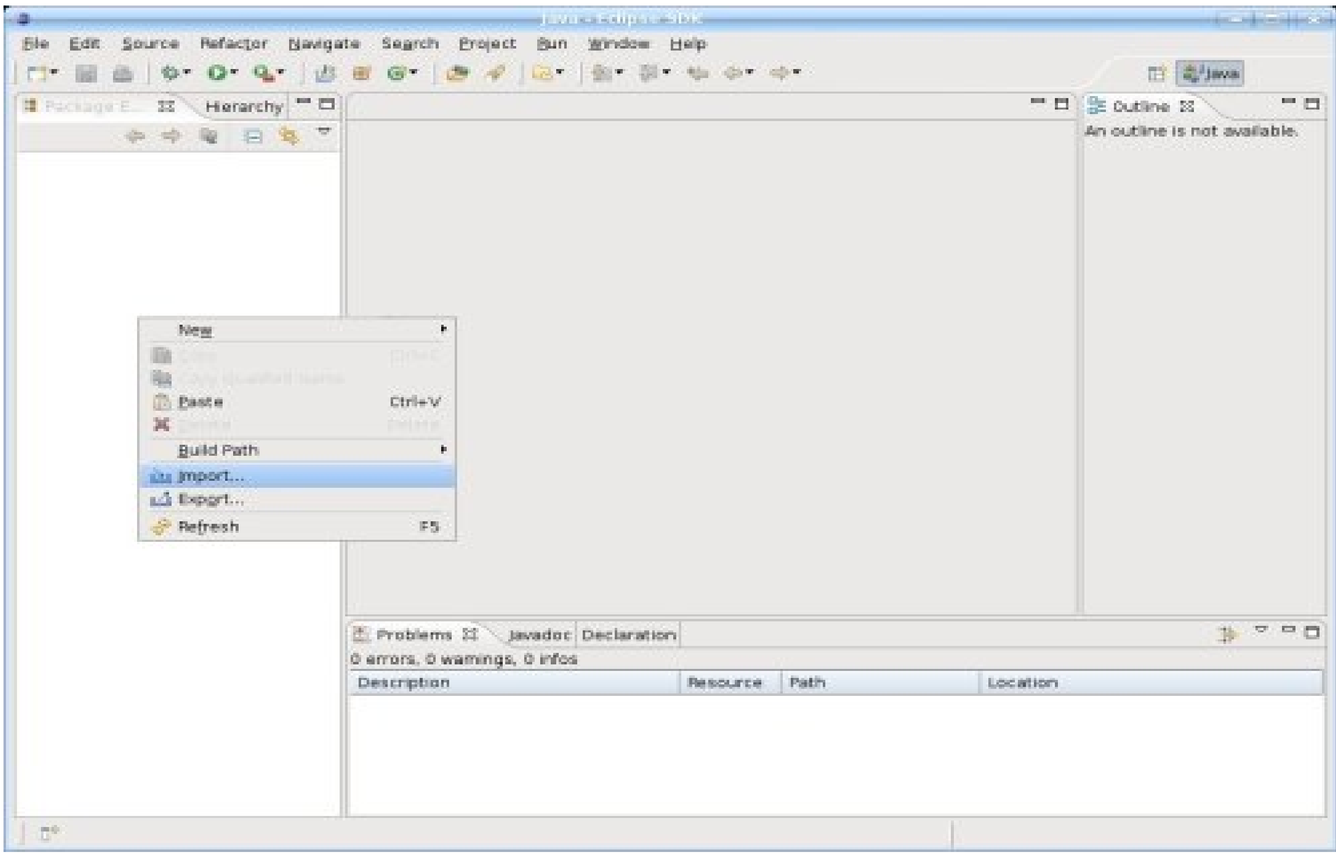
```
import liburbi.main.*;

/// Callback for images
class onImage extends UCallbackInterface {

  public UCallbackAction onMessage(UMessage msg) {
    if (!(UMessageType.MESSAGE_DATA == msg.getType ()))
      return UCallbackAction.URBI_CONTINUE;
    UValue value = msg.getValue ();

    if (!(UDataType.DATA_BINARY == value.getType ()))
      return UCallbackAction.URBI_CONTINUE;
    UBinary bin = value.getUBinary ();

    if (!(UBinaryType.BINARY_IMAGE == bin.getType ()))
      return UCallbackAction.URBI_CONTINUE;
    UImage img = bin.getUImage ();
    byte[] data = img.getDataAsByte ();

    System.out.println("Image of size " + img.getWidth () + "x"
                       + img.height + " received");

    /// myDisplay*Image are custom functions not part of liburbi-java
    /// replace by your own code
    if (img.imageFormat == UImageFormat.IMAGE_JPEG)
      myDisplayJPEGImage (data, img.getWidth (), img.getHeight ());
    else if (img.imageFormat == UImageFormat.IMAGE_YCbCr)
      myDisplayYCbCrImage (data, img.getWidth (), img.getHeight ());
    else if (img.imageFormat == UImageFormat.IMAGE_RGB)
      myDisplayRGBImage (data, img.getWidth (), img.getHeight ());

    return UCallbackAction.URBI_CONTINUE;
  }
}

/// Callback for sound
class onSound extends UCallbackInterface {

  public UCallbackAction onMessage(UMessage msg) {
    if (!(UMessageType.MESSAGE_DATA == msg.getType ()))
      return UCallbackAction.URBI_CONTINUE;
    UValue value = msg.getValue ();
    if (!(UDataType.DATA_BINARY == value.getType ()))
      return UCallbackAction.URBI_CONTINUE;
    UBinary bin = value.getUBinary ();

    if (!(UBinaryType.BINARY_SOUND == bin.getType ()))
      return UCallbackAction.URBI_CONTINUE;
    USound sound = bin.getUSound ();

    byte[] data = img.getDataAsByte ();

    System.out.println("Sound received");
    return UCallbackAction.URBI_CONTINUE;
  }
}

/// Callback for joint value
class onJoint extends UCallbackInterface {

  public UCallbackAction onMessage(UMessage msg) {
    if (!(UMessageType.MESSAGE_DATA == msg.getType ()))
      return UCallbackAction.URBI_CONTINUE;
    UValue value = msg.getValue ();

    if (!(UDataType.DATA_DOUBLE == value.getType ()))
      return UCallbackAction.URBI_CONTINUE;

    System.out.println("The joint value is " + value.getDouble ());
    return UCallbackAction.URBI_CONTINUE;
  }
}

public class test {
  static {
    /// Load the C++ liburbi library and the jni code.
    System.loadLibrary("urbijava");
  }

  public static void main(String argv[]) {

    UClient c = new UClient(argv[1]);

    if (c.error() != 0)
    {
 System.out.println("Couldn't connect to the URBI server.");
 System.exit(1);
    }

    onImage oi = new onImage ();
    onJoint oj = new onJoint ();
    onSound os = new onSound ();

    c.setCallback (oi, "img");
    c.setCallback (oj, "snd");
    c.setCallback (os, "joint");

    c.send("img: camera.val;");
    c.send("loop snd: micro.val,");
    c.send("joint: headPan.val;");
    liburbi.execute (); /// function that do a infinite sleep
  }
}
```

Chapter 4. Develop with Eclipse

Import the liburbi java project

We provide a sample Eclipse project configuration that you can import in Eclipse and use to create your project using Liburbi Java. We illustrate here how you can do this:

- Open Eclipse

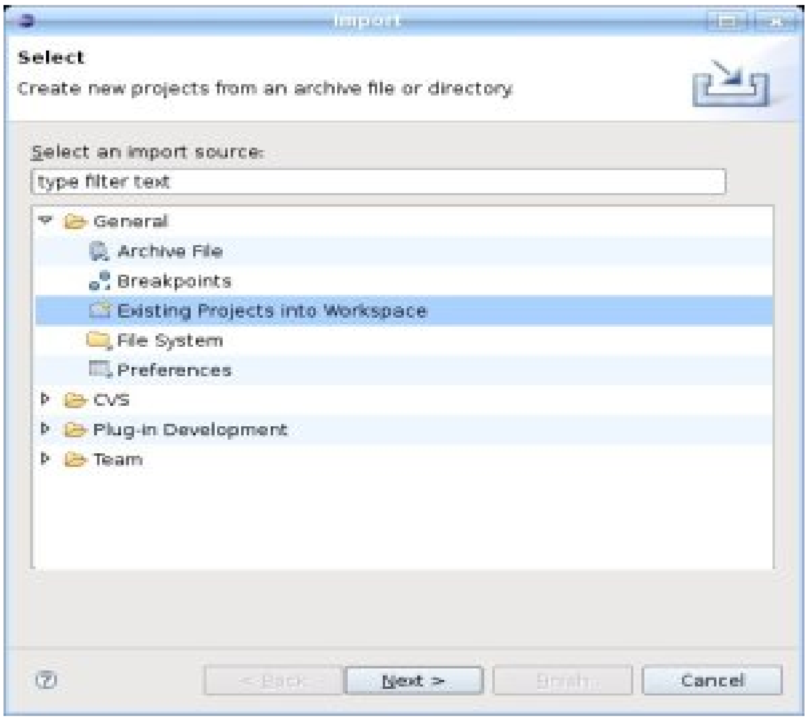


An outline is not available.

0 errors, 0 warnings, 0 infos

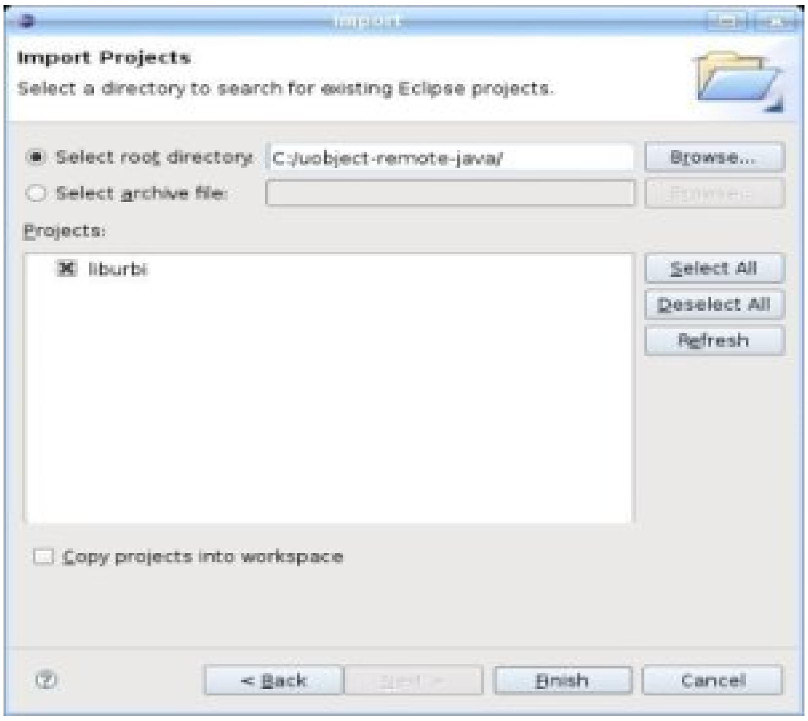
[images/eclipse-import.jpg]

- Right click in the Package Explorer panel and select 'import' (or go in File/import)



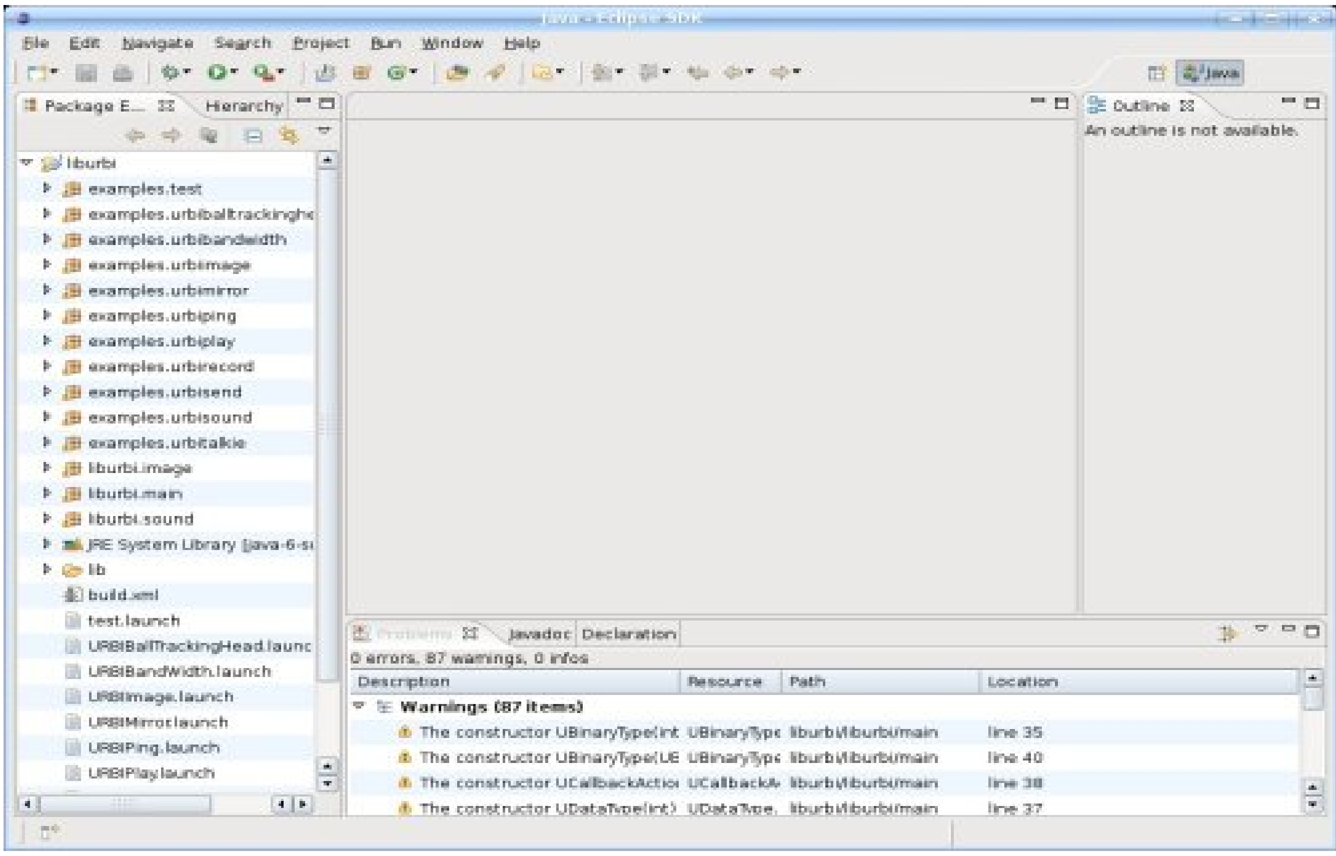
[images/select-import.jpg]

- Select 'Existing Projects into Workspace' in the opened windows
- Click 'Next'


Import
Import Projects
Select a directory to search for existing Eclipse projects.
Select root directory: C:/uobject-remote-java/
Select archive file:
Browse...
Projects:
liburbi
Select All
Deselect All
Refresh
Copy projects into workspace
< Back
Next >
Finish
Cancel

[images/select-proj.jpg]

- Enter the path of the sdk-remote java on your computer
- Eclipse should find .project file in the provided and display the 'liburbi' project
- Select the 'liburbi' project and click 'Finish'



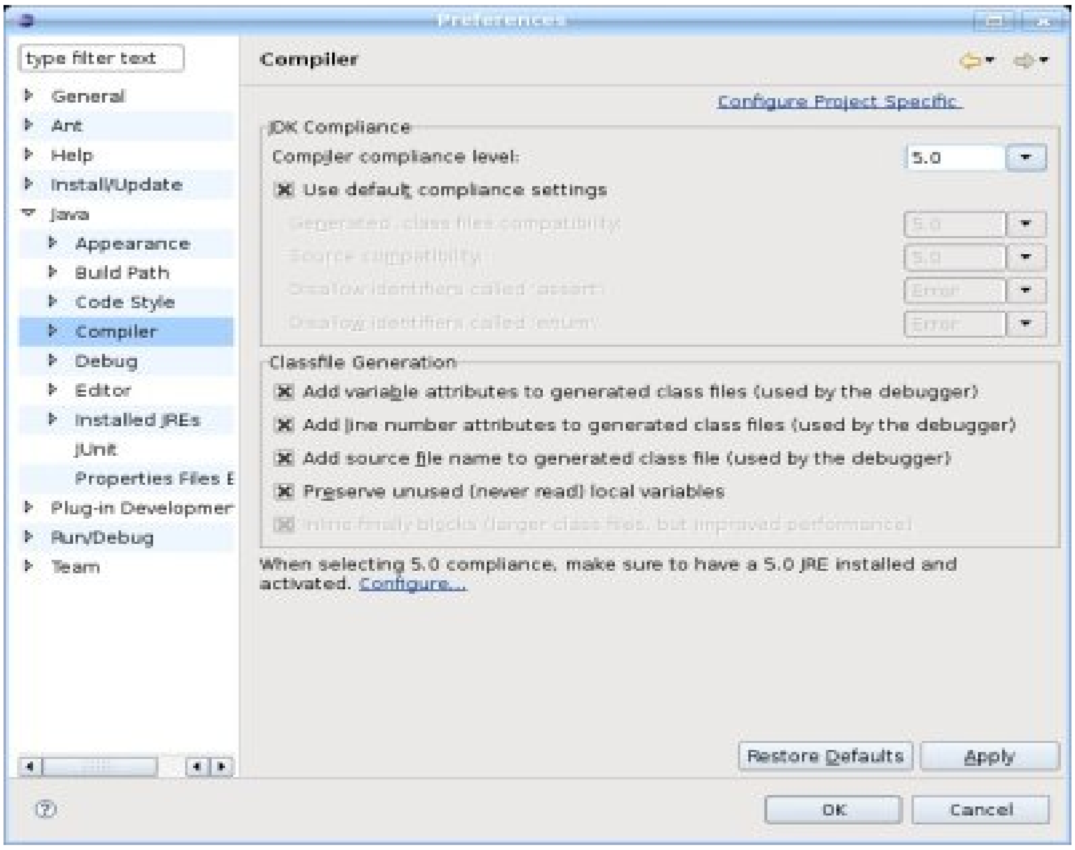
0 errors, 87 warnings, 0 infos

Description	Resource	Path	Location
Warnings (87 items)			
The constructor UBinaryType(int	UBinaryTypε	liburbi/liburbi/main	line 35
The constructor UBinaryType(UE	UBinaryTypε	liburbi/liburbi/main	line 40
The constructor UCallbackActio	UCallbackA	liburbi/liburbi/main	line 38
The constructor UDataType(int)	UDataType.	liburbi/liburbi/main	line 37

[images/project-liburbi-open.jpg]

The liburbi Java project is loaded. You can see the three packages of liburbi: liburbi.main, liburbi.image and liburbi.sound. From the source in these three packages, we made liburbijava.jar, that you can use in your applications. You can also see the sources of the different examples we provide. We put them in the packages examples.nameofexample. You can inspire yourself from these examples to make your own application. Here, we will see how to compile and run them in eclipse

NB: If Eclipse complains about errors in the source code, it can be that your compiler compliance level is two low. You have to set the compiler compliance level to Java 5 at least (Windows/Preferences/Java/Compiler).


Preferences

type filter text

- General
- Ant
- Help
- Install/Update
- Java
  - Appearance
  - Build Path
  - Code Style
  - Compiler
  - Debug
  - Editor
  - Installed JREs
  - JUnit
  - Properties Files E
- Plug-in Developmer
- Run/Debug
- Team

Compiler

Configure Project Specific

JDK Compliance

Compiler compliance level: 5.0

[✓] Use default compliance settings

Generated .class files compatibility: 5.0

Source compatibility: 5.0

Disallow identifiers called 'assert': Error

Disallow identifiers called 'enum': Error

Classfile Generation

[✓] Add variable attributes to generated class files (used by the debugger)

[✓] Add line number attributes to generated class files (used by the debugger)

[✓] Add source file name to generated class file (used by the debugger)

[✓] Preserve unused (never read) local variables

[✓] Inline finally blocks (larger class files, but improved performance)

When selecting 5.0 compliance, make sure to have a 5.0 JRE installed and activated. Configure...

Restore Defaults Apply

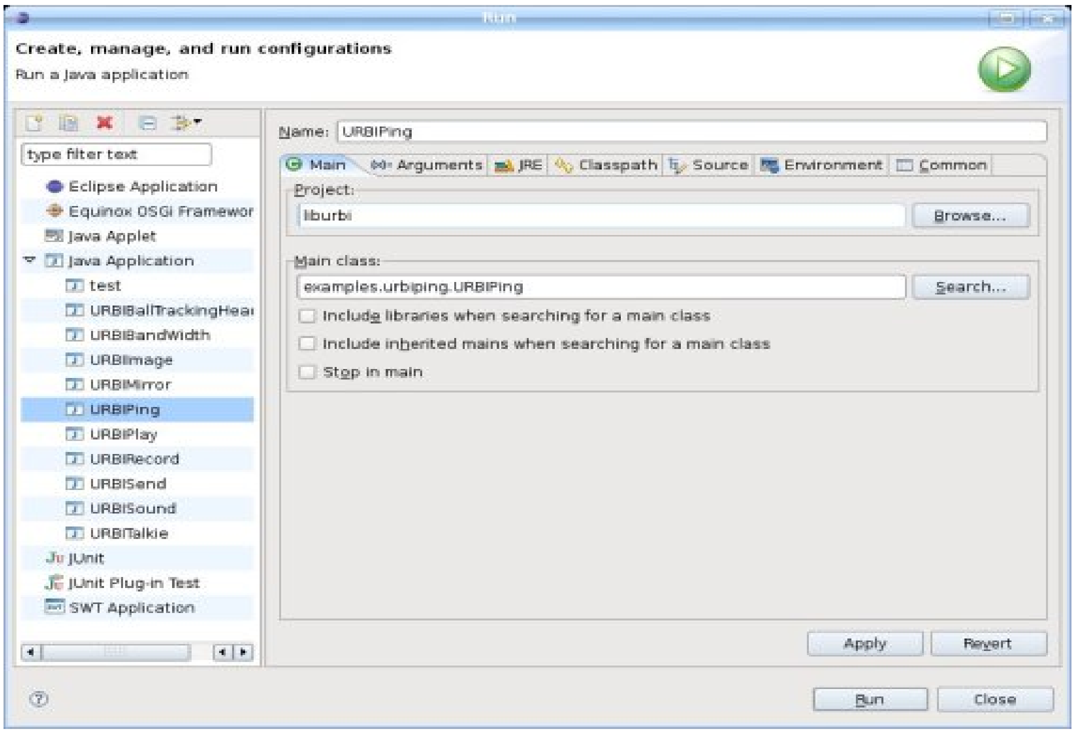
OK Cancel

[images/compiler-compliance-level.jpg]

Run the examples

We provide sample .launch files that you can load in eclipse to run the projects.

- Click on Run/Open Run Dialog (or 'Run...' in some versions of Eclipse)


Run

Create, manage, and run configurations

Run a Java application

type filter text

- Eclipse Application
- Equinox OSGi Framewor
- Java Applet
- Java Application
  - test
  - URBIBallTrackingHea
  - URBIBandWidth
  - URBIImage
  - URBIMirror
  - URBIPing
  - URBIPlay
  - URBIRecord
  - URBISend
  - URBISound
  - URBITalkie
- JUnit
- JUnit Plug-in Test
- SWT Application

Name: URBIPing

Main | Arguments | JRE | Classpath | Source | Environment | Common

Project:

liburbi Browse...

Main class:

examples.urbiping.URBIPing Search...

[ ] Include libraries when searching for a main class

[ ] Include inherited mains when searching for a main class

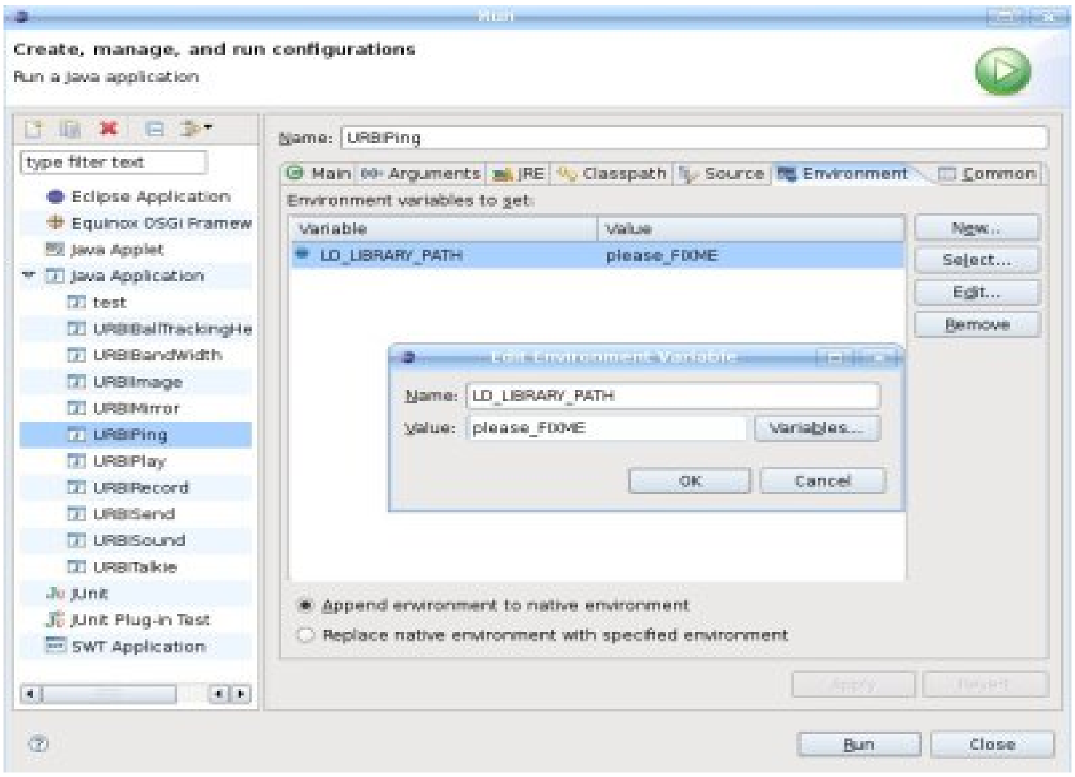
[ ] Stop in main

Apply Revert

Run Close

[images/run-urbiping.jpg]

- The launch configurations should be recognized automatically. You can choose one from the list, let's choose 'URBIPing'.
- In the right panel, select 'Environment', depending whether you are under Windows, Linux or MacOs, you will see the variable 'path', 'LD_LIBRARY_PATH' or 'DYLD_LIBRARY_PATH' defined.


Create, manage, and run configurations
Run a Java application

Name: URBIPing

Environment variables to set:

Variable	Value
LD_LIBRARY_PATH	please_FIXME

Edit Environment Variable
Name: LD_LIBRARY_PATH
Value: please_FIXME

● Append environment to native environment
○ Replace native environment with specified environment

[images/edit-ldlibpath.jpg]

- Edit the value of the variable. Set the absolute path of the liburbi java directory 'lib' (the one that contains the (lib)urbijava.{so,dll,dylib}). This is needed by the examples, in order to load the liburbijava library.
- Then click on the 'Arguments' tab, and enter the address of the urbi server you want to ping in the 'program arguments' field. For example if you want to ping the server running in Webots, set 'localhost' as address (without the quotes).
- Click 'Apply'. Click 'Run'.

Chapter 5. Programming hints

- Except if what you are doing is trivial, try not to use the sync* functions. They are less efficient than the asynchronous ones.
- The callback functions should return as fast as possible, since all callbacks are called by the same thread. If you have time-consuming operations, you should spawn an other thread and use synchronisation mechanisms such as semaphores or mutexes.

Appendix A. Copyright