

# **Universal open platform and reference specification for Ambient Assisted Living**



## **Personal Health Devices (x073) and universAAL middleware integration**

**<< *How to for uAAL developers* >>**

**Local device discovery and  
integration group (LDDI)**



## Release history

[illegible]

# Table of contents

Release history.....	3
Overview.....	6
Execution environment.....	8
Hardware requirements.....	8
Software requirements.....	10
Shoulders to the wheel.....	16
Working without uAAL middleware.....	17
Working with uAAL middleware.....	21
Troubleshooting.....	26
Appendix A.....	28
Setting up uAAL development environment on GNU/Linux .....	28
Troubleshooting.....	36



# Overview

The aim of this document is to make available a technical guide to enable universAAL (uAAL) developers community in general and LDDI experts group in particular to start developing Ambient Assisted Living (AAL) services with Continua Health Alliance devices (also called agents or sources).

This technological solution involves two pieces of code:

- **libHDPnative.so** is a dynamic library fully developed in C code that enables communication between BlueZ Bluetooth stack (C) and uAAL services (Java) through Java Native Interface (JNI) technology and inter-process communication (IPC) DBUS system. Is an isolated file with ".so" extension (similar to DLL files in Microsoft Windows environments).
- **x73implementation.jar** is the Continua Health Alliance standard (ISO/IEEE 11073) implementation developed in pure Java and without any external dependencies. It is composed by several packages grouped by performance: coder, decoder, DIM, state machine, etc.

Additionally other files or packages will be distributed:

- **javadoc.rar** file with a short description of methods, attributes, classes and relationships between them. Quite simply for Java developers.
- A sample project (**BloodPressureMonitor\_x073\_Sample**) showing how a Continua manager can receive valid and real time measurements through a HDP wireless channel. This example works outside OSGi platforms (no uAAL middleware is needed) so it should just be consider as a proof of concept.
- A sample maven project (**WeighingScalePublisher\_x073\_Sample**) showing how a Continua manager can receive weighting scales events and publish it inside uAAL context bus is provided. Both sample applications can be followed as a template in order to develop additional managers with more advance features.

Just to put in context the future reader, this guide (text and images) have been elaborated with the following test environment (not relevant at this point):

- Sony Vaio laptop with Ubuntu 11.10 OS (kernel version 3.0.0-19).
- GNU/Linux bluetooth stack blueZ<sup>1</sup> 4.99 version.
- Anycom 2.0 USB-200 bluetooth dongle.
- Real Continua health devices (just for testing): UC-321PBT-C A&D weigh scale<sup>2</sup> and UA-767PBT-C A &D upper arm blood pressure monitor<sup>3</sup>.

---

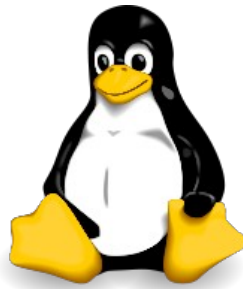
<sup>1</sup> <http://www.bluez.org>

<sup>2</sup> [http://www.aandd.jp/products/medical/bluetooth/uc\\_321pbt\\_c.html](http://www.aandd.jp/products/medical/bluetooth/uc_321pbt_c.html)

<sup>3</sup> [http://www.aandd.jp/products/medical/bluetooth/ua\\_767pbt\\_c.html](http://www.aandd.jp/products/medical/bluetooth/ua_767pbt_c.html)

# Execution environment

As has previously been mentioned, this technological solution relies over the official bluetooth stack (BlueZ) available in GNU/Linux operating systems, so from this point you should continue working with any free and open source OS distribution. Due to its high population and success, we decide work with the **version 11.10 of Ubuntu Desktop OS (32 bits)** directly downloaded from Ubuntu web page<sup>4</sup> (Ubuntu 12.04 LTS is out now).



First step is ensure that your computer has the minimum technological requirements necessary to run and start developing new AAL services with the aforementioned libraries:

## Hardware requirements

As Continua health devices employs bluetooth as transmission system basis, in order to send and receive x073 data frames between agents and managers it is obvious that your PC should support that wireless technology. **Developed source code is independent of any bluetooth dongle, so it doesn't matter which model or manufacturer you choose to test the libraries.**

In our case, we choose a bluetooth dongle USB-200 model (Bluetooth 2.0 + Enhanced Data Rate (EDR)) manufactured in 2006 by ANYCOM just because it was available for us as free resource in TSB company (please remember: any technical or strategic reason justifies this dongle choice).



---

4 <http://www.ubuntu.com/download/desktop>

In order to verify that your bluetooth dongle works fine in GNU/Linux platforms you can use (for instance) `sdptool` (interface for performing service queries on bluetooth devices around us) command as follows:

```
$ sdptool browse
Inquiring ...
Browsing 4C:ED:DE:0B:0D:98 ...
Service Name: Audio Source
Service Provider: TOSHIBA CORPORATION
Service RecHandle: 0x1008e
Service Class ID List:
  "Audio Source" (0x110a)
Protocol Descriptor List:
  "L2CAP" (0x0100)
    PSM: 25
  "AVDTP" (0x0019)
    uint16: 0x102
Language Base Attr List:
  code_ISO639: 0x656e
  encoding:    0x6a
  base_offset: 0x100
Profile Descriptor List:
  "Advanced Audio" (0x110d)
    Version: 0x0102 ...
```

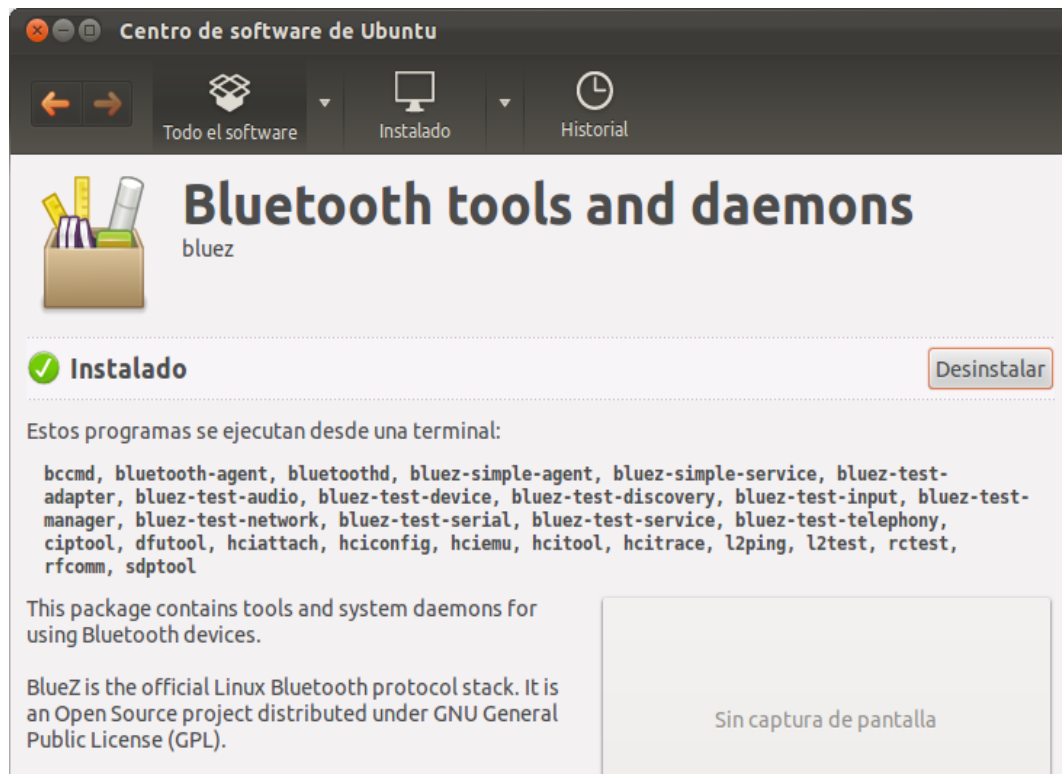
**Table 1: Ubuntu console (uAAL developer input/system output)**

```
angel@angel-laptop:~$ sdptool browse
Inquiring ...
Browsing 4C:ED:DE:0B:0D:98 ...
Service Name: Audio Source
Service Provider: TOSHIBA CORPORATION
Service RecHandle: 0x1008e
Service Class ID List:
  "Audio Source" (0x110a)
Protocol Descriptor List:
  "L2CAP" (0x0100)
    PSM: 25
  "AVDTP" (0x0019)
    uint16: 0x102
Language Base Attr List:
  code_ISO639: 0x656e
  encoding:    0x6a
  base_offset: 0x100
Profile Descriptor List:
  "Advanced Audio" (0x110d)
    Version: 0x0102
```

**Figure 1: Available bluetooth services in our coverage range (checking bluetooth dongle performance)**



Important note: `sdptool` command is part of BlueZ (bluetooth tools and daemons) package so if that package is not available in our PC, we can't use it. Just ensure that `bluez` is ready to be launched or install it from any software manager (Synaptic, Ubuntu software manager centre, etc) otherwise.



**Figure 2: Ubuntu software manager with BlueZ package installed**

## Software requirements

**This technical guide assumes that you have some expertise developing uAAL services as well as setting up the required environment and related plug-ins** (if not, please carefully read the universAAL Developer Handbook<sup>5</sup> (login required) and come back later), so you will probably have within your reach some generic Java developer tools, namely Maven, SVN client, Pax Runner and Eclipse IDE. Sorry for bothering you again in that case:

- The Java JDK available by default in GNU/Linux systems is a FLOSS (free and open source) alternative named as OpenJDK. This libraries has been developed with one of the latest versions of the **official Oracle JDK** available for download so it is highly recommended work with the same JDK than us. After completely remove the OpenJDK from our system (if necessary) we can download and install the right version of the Java Development Kit. Next table shows you how install Oracle-jdk7 for 32 bit architecture (Important note: 64 bit Oracle JDK version don't work on 32

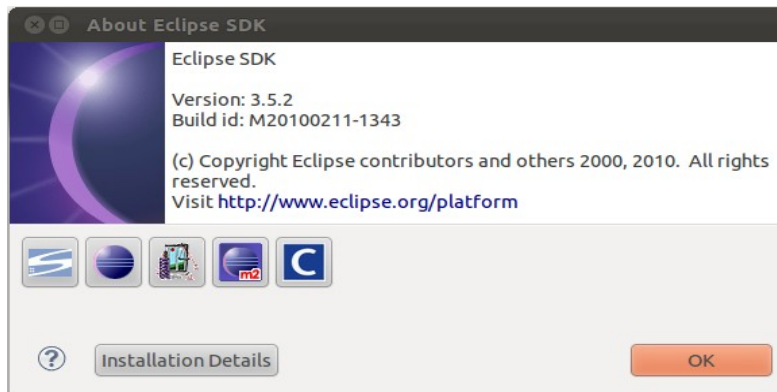
<sup>5</sup> <http://forge.universaal.org/gf/>

bit OS, ensure you are working with the appropriate binaries package).

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-jdk7-installer
$ java -version
java version "1.7.0_04"
Java(TM) SE Runtime Environment (build 1.7.0_04-b20)
Java HotSpot(TM) Server VM (build 23.0-b21, mixed mode)
```

**Table 2: Downloading and installing Oracle JDK version 7 (32 bits)**

- Eclipse IDE and related plugins suggested to improve and facilitate uAAL development working process. Please, read [Appendix A](#) at the end of this document to a better understanding of how to proceed at each case. Summarizing, you will download **Eclipse IDE for GNU/Linux OS version 3.5.2** (as recommended in the Developer Handbook of universAAL) **and install some additional software tools.**



**Figure 3: Eclipse IDE installed**

- As has previously been mentioned, BlueZ is the core of this technological solution and contains tools and system daemons for using Bluetooth devices. BlueZ is the official Linux Bluetooth protocol stack, which has full support of Health Device Profile (HDP) required to exchange data between Continua agents and managers (AAL software services running inside OSGi framework, for instance). Despite of this code is distributed by default in GNU/Linux OS, as developers we need to install next package in order to avoid future dependencies:

- **libbluetooth3** as library to use the bluez linux bluetooth stack.

Our libraries has been developed with **4.96 version** (bluez 4.96-0ubuntu4 package) of BlueZ (latest release available for download is 4.101<sup>6</sup> nowadays). Upper versions of 4.80 is mandatory due to it was the first release where HDP was integrated.

- API of GNU/Linux official bluetooth stack, BlueZ, uses the popular

<sup>6</sup> <http://www.bluez.org/download/>

message bus system created by Red Hat company D-Bus<sup>7</sup>. D-Bus is an inter-process communication (IPC) open-source system that enables a reliable way for exchanging data (messages) directly between software applications and between software applications and the operating system (where the operating system would typically include the kernel and any system daemons or processes) over the same desktop session. Although there are some implementations<sup>8</sup> (bindings) of D-Bus in several higher level development languages (Java, Python, Qt, Glib, C#, etc) we decided to base our development over the low level D-Bus implementation which API is written in C code for the following reasons:

- Low level implementation has no additional dependencies unlike higher level bindings and it is distributed by default in GNU/Linux OS also.
- Java binding (the closest to uAAL partners interests, for obvious reasons) has a very important lack compared with low level implementation (C code): file descriptor data type is not supported in the latest release available for download (D-Bus works with Unix sockets so a valid file descriptor should be reserved to any HDP channel created prior to send and receive data bytes between agents and managers).

Our libraries has been developed with **1.4.14 D-Bus version** (latest release available for download is 1.6.1 nowadays). Please keep in mind that working with low level D-Bus C API is not easy and has a very pronounced learning curve so you should only go deeper just if necessary (main page of D-Bus C API<sup>9</sup> welcomes visitors with a warm: *If you use this low-level API directly, you're signing up for some pain*). Please, ensure that any version of D-Bus is conveniently installed. Additionally, as developers we need to install some packages related with **libdbus** library in order to avoid future dependencies:

- **libglib2.0-dev** as development files for GLib libraries.
- **libdbus1.0-ci1-dev** as CLI implementation of D-Bus.
- **libdbus-glib1.0-ci1-dev** as CLI implementation of D-Bus (Glib mainloop integration).
- **libdbus-1-dev** as development headers for simple interprocess messaging system.
- **libdbus-glib-1-dev** as Glib interface.

---

<sup>7</sup> <http://www.freedesktop.org/wiki/Software/dbus>

<sup>8</sup> <http://www.freedesktop.org/wiki/Software/DBusBindings>

<sup>9</sup> <http://dbus.freedesktop.org/doc/api/html/index.html>

E	Paquete	Versión instalada	Última versión	Descripción
	libdbus-glib1.0-cil	0.5.0-3build1	0.5.0-3build1	CLI implementation of D-Bus (GLib mainloop integration)
	libdbus-glib1.0-cil-dev	0.5.0-3build1	0.5.0-3build1	CLI implementation of D-Bus (GLib mainloop integration) - development files
	libdbus1.0-cil	0.7.0-4	0.7.0-4	CLI implementation of D-Bus
	libdbus1.0-cil-dev	0.7.0-4	0.7.0-4	Implementación CLI de D-Bus - archivos de desarrollo
	libdbus-1-dev	1.4.14-1ubuntu1	1.4.14-1ubuntu1	simple interprocess messaging system (development headers)
	libdbus-glib-1-dev	0.94-4	0.94-4	sistema de mensajes interproceso simple (interfaz GLib)
	libdbus-glib-1-2	0.94-4	0.94-4	sistema simple de mensajería inter-proceso (biblioteca dinámica basada en GLib)
	dbus	1.4.14-1ubuntu1	1.4.14-1ubuntu1	simple interprocess messaging system (daemon and utilities)
	libdbus-1-3	1.4.14-1ubuntu1	1.4.14-1ubuntu1	simple interprocess messaging system (library)

**Figure 4: Synaptic package manager view with "libdbus" and "installed packages" as quick filters**

If everything goes as expected, our local folders (`usr/lib/i386-linux-gnu/dbus-1.0/include/dbus/` and `usr/include/dbus-1.0/dbus/`) should contain some new header files as shown following:

```

angel@angel-laptop: ~
angel@angel-laptop:~$ ls -l /usr/lib/i386-linux-gnu/dbus-1.0/include/dbus/
total 4
-rw-r--r-- 1 root root 2089 2011-09-02 11:30 dbus-arch-deps.h
angel@angel-laptop:~$ ls -l /usr/include/dbus-1.0/dbus/
total 204
-rw-r--r-- 1 root root 2135 2011-09-02 11:30 dbus-address.h
-rw-r--r-- 1 root root 3472 2011-09-02 11:30 dbus-bus.h
-rw-r--r-- 1 root root 25953 2012-04-10 17:02 dbus-connection.h
-rw-r--r-- 1 root root 2886 2011-09-02 11:30 dbus-errors.h
-rw-r--r-- 1 root root 23000 2011-07-07 16:52 dbus-glib-bindings.h
-rw-r--r-- 1 root root 14376 2011-07-07 16:51 dbus-glib.h
-rw-r--r-- 1 root root 2875 2011-07-07 16:51 dbus-glib-lowlevel.h
-rw-r--r-- 1 root root 8868 2011-07-07 16:51 dbus-gtype-specialized.h
-rw-r--r-- 1 root root 1384 2011-07-07 16:51 dbus-gvalue-parse-variant.h
-rw-r--r-- 1 root root 3925 2012-04-10 17:03 dbus.h
-rw-r--r-- 1 root root 4881 2011-09-02 11:30 dbus-macros.h
-rw-r--r-- 1 root root 1975 2011-09-02 11:30 dbus-memory.h
-rw-r--r-- 1 root root 13049 2011-09-02 11:30 dbus-message.h
-rw-r--r-- 1 root root 1609 2011-09-02 11:30 dbus-misc.h
-rw-r--r-- 1 root root 3124 2011-09-02 11:30 dbus-pending-call.h
-rw-r--r-- 1 root root 23178 2011-09-02 11:30 dbus-protocol.h
-rw-r--r-- 1 root root 3534 2011-07-27 13:30 dbus-python.h
-rw-r--r-- 1 root root 4794 2011-09-02 11:30 dbus-server.h
-rw-r--r-- 1 root root 5135 2011-09-02 11:30 dbus-shared.h
-rw-r--r-- 1 root root 2956 2011-09-02 11:30 dbus-signature.h
-rw-r--r-- 1 root root 9004 2011-09-02 11:30 dbus-threads.h
-rw-r--r-- 1 root root 3635 2012-04-10 17:10 dbus-types.h
angel@angel-laptop:~$

```

**Figure 5: D-Bus IPC system header files**

At this point, we should ensure that D-Bus, BlueZ and our bluetooth dongle works fine as a whole. In order to verify (again) that your bluetooth adapter is ready to work run `hciconfig` command (for example) as root.

```
angel@angel-laptop: ~  
angel@angel-laptop:~$ hciconfig  
hci0:   Type: BR/EDR   Bus: USB  
        BD Address: 00:16:38:C1:9D:EA   ACL MTU: 1017:8   SCO MTU: 64:0  
        UP RUNNING PSCAN ISCAN  
        RX bytes:1005 acl:0 sco:0 events:27 errors:0  
        TX bytes:622 acl:0 sco:0 commands:27 errors:0  
angel@angel-laptop:~$
```

**Figure 6: Checking availability of local bluetooth adapter**

Next, type the following D-Bus command to get your assigned BlueZ D-Bus address:

```
$ dbus-send --system --type=method_call --print-reply  
--dest=org.bluez "/" org.bluez.Manager.ListAdapters
```

**Table 3: List of local bluetooth adapters**

```
angel@angel-laptop: ~  
angel@angel-laptop:~$ dbus-send --system --type=method_call --print-reply --dest=org.bluez "/" org.bluez.Manager.ListAdapters  
method return sender=:1.8 -> dest=:1.73 reply_serial=2  
array [  
  object path "/org/bluez/1043/hci0"  
]  
angel@angel-laptop:~$ pidof bluetoothd  
1043  
angel@angel-laptop:~$
```

**Figure 7: Local bluetooth adapter path returned by D-Bus system**

The returned object path as string `org/bluez/1043/hci0` is our adapter's name where `1043` is the PID (Process ID) bluetooth process at this session (Important note: local bluetooth adapter PID varies after each computer boot so it shouldn't be consider as static value). As next step, we can (for instance) request for local bluetooth adapter properties as follows:

```
$ dbus-send --system --type=method_call --print-reply  
--dest=org.bluez "/org/bluez/1043/hci0"  
org.bluez.Adapter.GetProperties
```

**Table 4: Default local bluetooth adapter properties**

If everything goes well, automatically output should be (more or less) similar to:

```
method return sender=:1.8 -> dest=:1.74 reply_serial=2
array [
  dict entry(
    string "Address"
    variant          string "00:16:38:C1:9D:EA"
  )
  dict entry(
    string "Name"
    variant          string "angel-laptop-0"
  )
  dict entry(
    string "Class"
    variant          uint32 4849920
  )
  dict entry(
    string "Powered"
    variant          boolean true
  )
  dict entry(
    string "Discoverable"
    variant          boolean true
  )
  dict entry(
    string "Pairable"
    variant          boolean true
  )
  dict entry(
    string "DiscoverableTimeout"
    variant          uint32 0
  )
  dict entry(
    string "PairableTimeout"
    variant          uint32 0
  )
  dict entry(
    string "Discovering"
    variant          boolean false
  )
  dict entry(
    string "Devices"
    variant          array [
      object path "/org/bluez/1043/hci0/dev_00_09_1F_80_0A_E0"
      object path "/org/bluez/1043/hci0/dev_00_09_1F_80_04_D6"
    ]
  )
  dict entry(
    string "UUIDs"
    variant          array [
      string "00001000-0000-1000-8000-00805f9b34fb"
      string "00001001-0000-1000-8000-00805f9b34fb"
      string "00001112-0000-1000-8000-00805f9b34fb"
      string "0000111f-0000-1000-8000-00805f9b34fb"
      string "0000110a-0000-1000-8000-00805f9b34fb"
      string "0000110c-0000-1000-8000-00805f9b34fb"
      string "0000110e-0000-1000-8000-00805f9b34fb"
      string "00001103-0000-1000-8000-00805f9b34fb"
    ]
  )
]
]
```

**Table 5: Default local bluetooth adapter properties returned by D-Bus system**

# Shoulders to the wheel

Summarizing, for being able to run the libraries, you will have to need a PC with GNU/Linux operating system installed with one of the latest versions of BlueZ library and D-Bus system for managing Bluetooth connections. In our case, we had a 32-bit Ubuntu 11.10 with BlueZ 4.96 and Oracle Java JDK 7 also.

Important note: as prior step to exchange data bytes between Continua agents and managers both should have been paired first. Developed libraries doesn't have an automatic process to manage this task at this release so you should use any bluetooth manager software available for download. We decided to use the software distributed with the OS by default for simplicity. Next figure shows that application with both the weigh scale and the upper arm blood pressure monitor paired with our computer (you will probably need to consult the right instruction manual of chosen Continua device to get the secret PIN required as final step of any pairing process).



**Figure 8: Bluetooth software manager with Continua compliant paired devices**

Once the Continua agents and our PC (or any smart device with manager role) has been paired, we can continuous with the next steps as detailed below.



## **Working without uAAL middleware**

As uAAL middleware runs inside an OSGi framework and this environment is not too intuitive and easy to integrate third party libraries (at the beginning), a good practice could be start working with the libraries developed in standalone mode as first step. Please, follow next steps:

1. Update your local copy of the uAAL repository with the latest version of the libraries. The contents you must have should be:

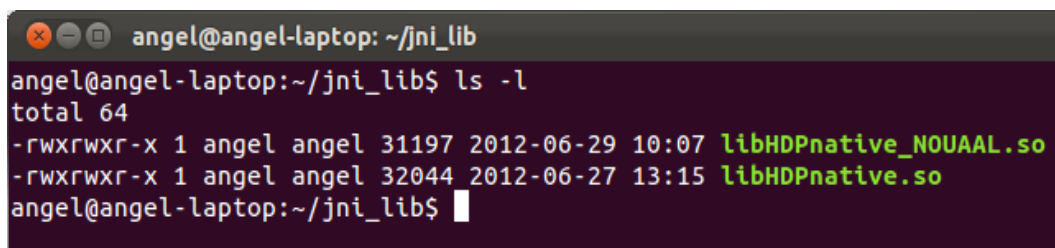
- Jar file with the implementation of ISO/IEEE 11073 standard (java pure code, distributed in several packages and without external dependencies): **x73implementation.jar**
- Native library written in C code (with ".so" extension) with JNI functions for accessing Bluetooth stack with Java through D-Bus IPC system: **libHDPnative\_NOUAAL.so**
- Javadoc documentation for helping uAAL developers creating Continua managers with Bluetooth HDP capabilities: **javadoc.rar**
- Java code (Eclipse IDE project format) with a manager as sample: **BloodPressureMonitor\_x073\_Sample**

With this three elements and the sample of code, you will be able to create your own managers, and obtain measurements from any Continua compliant medical device such as a weighing scale or a blood pressure monitor.

2. Ensure that your Continua device/devices is/are successfully paired with your computer (this step is mandatory, don't forget it).

3. Create a new folder **exactly named** `jni_lib` at your home path (Important note: this path must be exact. Otherwise, it won't work).

4. Copy the `libHDPnative_NOUAAL.so` file into the previously created directory.



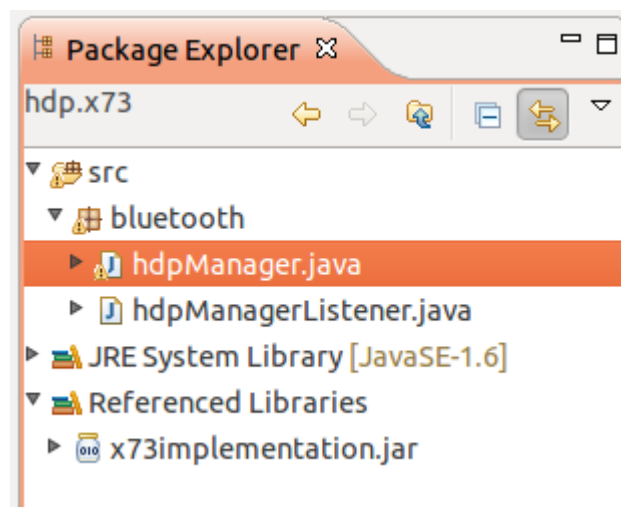
```
angel@angel-laptop: ~/jni_lib
angel@angel-laptop:~/jni_lib$ ls -l
total 64
-rwxrwxr-x 1 angel angel 31197 2012-06-29 10:07 libHDPnative_NOUAAL.so
-rwxrwxr-x 1 angel angel 32044 2012-06-27 13:15 libHDPnative.so
angel@angel-laptop:~/jni_lib$
```

**Figure 9: Native library inside our local folder**

5. Import the sample project (**BloodPressureMonitor\_x073\_Sample**) available in uAAL repository in Eclipse IDE or create a new one with the same package structure than the other one. In any case, jar package with ISO/IEEE x073 implementation must be imported into the Java project. So, your software project should contain next classes/imported packages at least:



- **hdpManager.java** is the core of the manager application. It should implement **hdpManagerListener.java** interface.
- **hdpManagerListener.java** is an interface that only defines three self-explanatory methods: **onChannelConnected**, **onChannelDisconnected** and **onDataReceived**. Please, refer the aforementioned Javadoc file to get more detailed information.
- **x73implementation.jar** is the ISO/IEEE 11073 standard implementation in Java code. It is responsible for processing and managing x073 received data frames.



**Figure 10: Package explorer of Eclipse IDE**

Important note: package name that contains **hdpManager.java** and **hdpManagerListener.java** classes must be **bluetooth**. Otherwise, native library will return an error (this file is, more or less, auto-generated by the **javah** tool included in the JDK so this value is the right expected).

6. Since the output text messages from the C functions at **libHDPnative.so** file seems not to be showing in the console of the Eclipse IDE platform, we recommend to execute the software from the GNU/Linux OS console terminal as follows:

- In **hdpManager.java** file change the **macAddressRemoteDevice** attribute with the corresponding MAC value (line 76) of your blood pressure device.
- If your project is built automatically by Eclipse IDE (Project → Build automatically option checked) go to the **bin** folder inside the sample project directory (workspace) and execute **java**

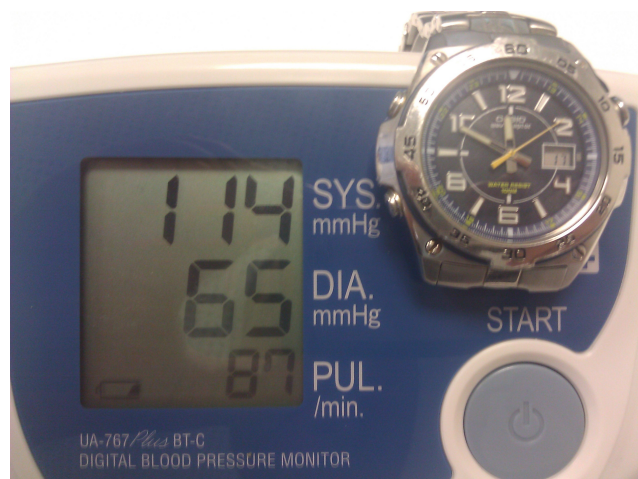
`bluetooth.hdpManager` command (where `hdpManager` is the manager main class and `bluetooth` the name of the container package). Next table shows some properties about the local bluetooth adapter, its valid PID and how an HDP channel is waiting for Continua agent frames to start the exchange data process.

```
angel@angel-laptop: ~/TSB/src/workspace_uaal/BloodPressureMonitor_x073_Sample/bin
angel@angel-laptop:~/TSB/src/workspace_uaal/BloodPressureMonitor_x073_Sample/bin$ java bluetooth.hdpManager
UNASSOCIATED
Default bluetooth adapter path: /org/bluez/1107/hci0
List of local bluetooth adapters:
Adapters: /org/bluez/1107/hci0
Local bluetooth properties:
Address: 00:16:38:C1:9D:EA
Name: angel-laptop-0
Class: 4849920
Powered: TRUE
Discoverable: TRUE
Pairable: TRUE
DiscoverableTimeout: 0
PairableTimeout: 0
Discovering: FALSE
Devices: /org/bluez/1107/hci0/dev_00_09_1F_80_0A_E0
/org/bluez/1107/hci0/dev_00_09_1F_80_04_D6
UUIDs: 00001000-0000-1000-8000-00805f9b34fb
00001001-0000-1000-8000-00805f9b34fb
00001112-0000-1000-8000-00805f9b34fb
0000111f-0000-1000-8000-00805f9b34fb
0000110a-0000-1000-8000-00805f9b34fb
0000110c-0000-1000-8000-00805f9b34fb
0000110e-0000-1000-8000-00805f9b34fb
```

**Figure 11: Blood pressure manager successfully launched**

7. Start getting measurements from the Continua devices in our area of coverage is the actual step. Next images shows an example of a blood pressure measurement:

```
LOG: Initiating processing of a received measurement from a Blood Pressure
MEASUREMENT: 114 mmHg
MEASUREMENT: 65 mmHg
MEASUREMENT: 83 mmHg
MEASUREMENT: 87 bpm
MEASUREMENT: Date of measurement: 11:49:05 - 17/07/2012
SYSTEM TIME: 11:49:07 - 17/07/2012
```



**Figure 12: Measurements done with a real Continua blood pressure monitor**

```

$ pwd
/.../workspace_uaal/BloodPressureMonitor_x073_Sample/bin
/.../workspace_uaal/BloodPressureMonitor_x073_Sample/bin$ java bluetooth.hdpManager
UNASSOCIATED
Default bluetooth adapter path: /org/bluez/1107/hci0
List of local bluetooth adapters:
Adapters: /org/bluez/1107/hci0
Local bluetooth properties:
Address: 00:16:38:C1:9D:EA
Name: angel-laptop-0
Class: 4849920
Powered: TRUE
Discoverable: TRUE
Pairable: TRUE
DiscoverableTimeout: 0
PairableTimeout: 0
Discovering: FALSE
Devices: /org/bluez/1107/hci0/dev_00_09_1F_80_0A_E0
/org/bluez/1107/hci0/dev_00_09_1F_80_04_D6
UUIDs: 00001000-0000-1000-8000-00805f9b34fb
00001001-0000-1000-8000-00805f9b34fb
00001112-0000-1000-8000-00805f9b34fb
0000111f-0000-1000-8000-00805f9b34fb
0000110a-0000-1000-8000-00805f9b34fb
0000110c-0000-1000-8000-00805f9b34fb
0000110e-0000-1000-8000-00805f9b34fb
00001103-0000-1000-8000-00805f9b34fb
Remote bluetooth adapter path: /org/bluez/1107/hci0/dev_00_09_1F_80_04_D6
Remote device properties (BEFORE):
Address: 00:09:1F:80:04:D6
Name: AND BP5091100122
Alias: AND BP5091100122
Class: 2308
Paired: TRUE
Trusted: TRUE
Blocked: FALSE
Connected: FALSE
UUIDs: 00001200-0000-1000-8000-00805f9b34fb
00001401-0000-1000-8000-00805f9b34fb
Services:
Adapter: /org/bluez/1107/hci0
Property Trusted successfully changed
Remote device properties (AFTER):
Address: 00:09:1F:80:04:D6
Name: AND BP5091100122
Alias: AND BP5091100122
Class: 2308
Paired: TRUE
Trusted: TRUE
Blocked: FALSE
Connected: FALSE
UUIDs: 00001200-0000-1000-8000-00805f9b34fb
00001401-0000-1000-8000-00805f9b34fb
Services:
Adapter: /org/bluez/1107/hci0
HDP application identifier: /org/bluez/health_app_3
HDP application (/org/bluez/health_app_3) successfully closed
HDP application identifier: /org/bluez/health_app_4
DBUS sytem status: Up
CONNECTED UNASSOCIATED
Waiting for source connections...

```

**Table 6: Continua manager waiting for input source connections (agents)**

## **Working with uAAL middleware**

Once we were able to get real time measurements from Continua devices in standalone mode as has shown before we need to migrate that technological solution inside OSGi environments due to this framework is the basis of uAAL middleware. Please, follow next steps and be patient:

1. Update your local copy of the uAAL repository with the latest version of the libraries. The contents you must have should be:

- Maven project with the implementation of ISO/IEEE 11073 standard (java pure code, distributed in several packages and without external dependencies): **ieee.x73.std**
- Native library written in C code (with ".so" extension) with JNI functions for accessing Bluetooth stack with Java through D-Bus IPC system: **libHDPnative.so**
- Javadoc documentation for helping uAAL developers creating Continua managers with Bluetooth HDP capabilities: **javadoc.rar**
- Maven project with a manager as sample: **weighingScalePublisher\_x073\_Sample**

With this three elements and the sample of code, you will be able to create your own managers, obtain measurements from any Continua compliant medical device such as a weighing scales or a blood pressure monitors and send data received to uAAL context bus.

2. Ensure that your Continua device/devices is/are successfully paired with your computer (this step is mandatory, don't forget it).

3. Create a new folder **exactly named jni\_lib** at your home path (Important note: this path must be exact. Otherwise, it won't work).

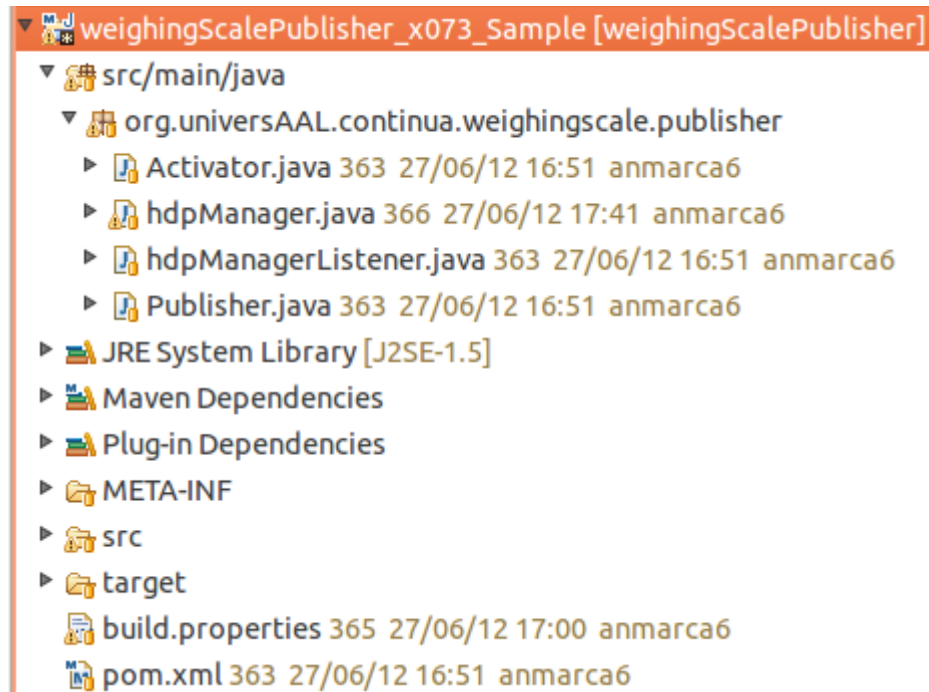
4. Copy the **libHDPnative.so** file into the previously created directory (please, review figure 9).

5. Import the sample maven project (**weighingScalePublisher\_x073\_Sample**) available in uAAL repository in Eclipse IDE or create a new one with the same package structure than the other one. So, your software project should contain next classes:

- **Activator.java** is the bundle activator (main starting point).
- **hdpManager.java** is the core of the manager application. It should implement **hdpManagerListener.java** interface.
- **hdpManagerListener.java** is an interface that only defines three self-explanatory methods: **onChannelConnected**, **onChannelDisconnected** and **onDataReceived**. Please, refer the aforementioned Javadoc file to

get more detailed information.

- `Publisher.java` is the class responsible for publishing Continua agent's events over uAAL context bus.

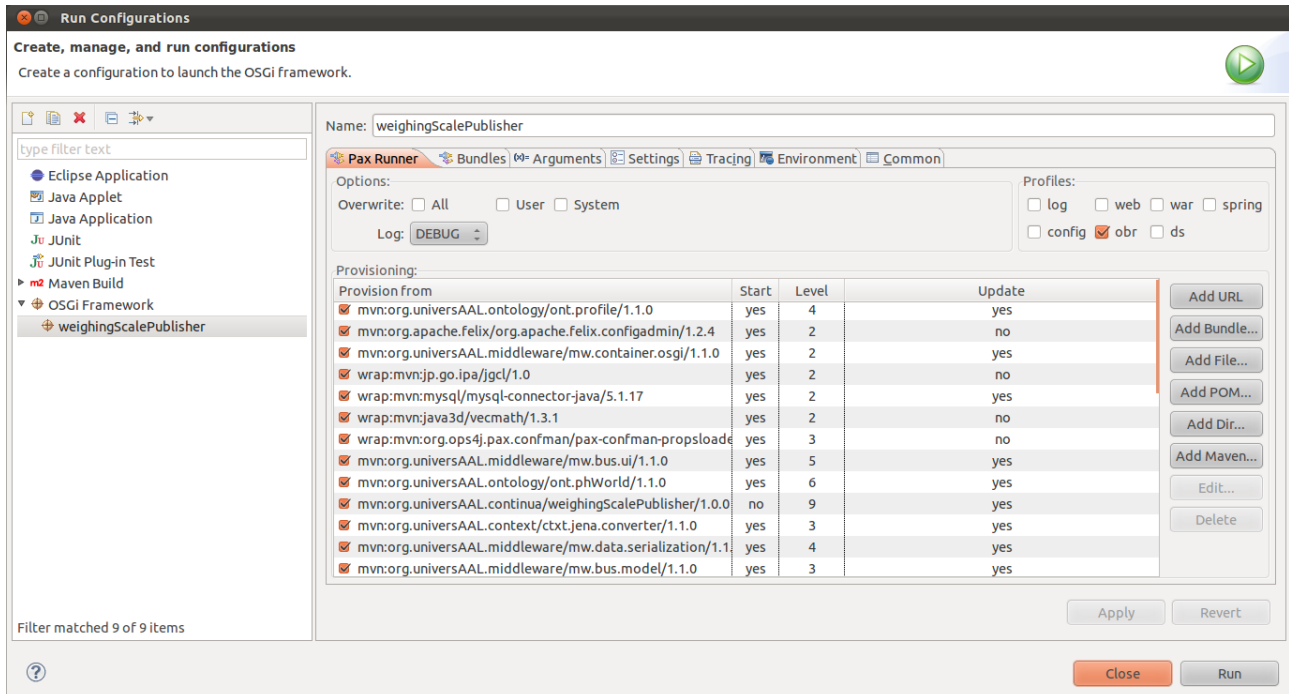


**Figure 13: Publisher sample project in detail (Eclipse IDE package explorer)**

Important note 1: (1) package name that contains related classes must be `org.universAAL.continua.weighingscale.publisher`. Otherwise, native library will return an error (this file is, more or less, auto-generated by the `javah` tool included in the JDK so this value is the right expected. **In conclusion: if this package is renamed, functions in native library should be renamed too. Otherwise, both names will not match and an error will be return as output**) (2) manage possible external dependencies with the pom file (`pom.xml`). There are some ontologies specifically created to model Continua agents as weighing scales and blood pressures. **Please review `org.universAAL.ontology.x73` package in uAAL SVN repositories** (3) Delete (if proceed) the `x073implementation.jar` JAR file previously copied inside your local path folder (`/usr/lib/jvm/java-7-oracle/jre/lib/ext`) to force maven project get the library from the right source.

6. Launch this application with an appropriate **Run Configurations** (remember: output text messages from the C functions at `libHDPnative.so` file seems not to be showing in the console of the Eclipse IDE platform, this bug will be solved ASAP by TSB team):

- In `hdpManager.java` file change the `macAddressRemoteDevice` attribute with the corresponding MAC value (line 39) of your weighing scale device.



**Figure 14: Eclipse IDE run configurations with Pax Runner tool**

Important note: VM arguments at Arguments tab in the Run Configurations window should contain the path where native library has been stored as follows: `-Djava.library.path=/home/user/jni_lib`

```
-Dosgi.noShutdown=true -Dfelix.log.level=4
-Dorg.universAAL.middleware.peer.is_coordinator=true
-Dorg.universAAL.middleware.peer.member_of=urn:org.universAAL.aal_space:test_env
-Dbundles.configuration.location=${workspace_loc}/rundir/confadmin
-Djava.library.path=/home/angel/jni_lib
```

**Table 7: VM arguments**

7. Start the appropriate bundle number and wait for input sources (Continua agents events):

```

->
-> ps
START LEVEL 20
  ID   State      Level Name
[  0] [Active]   [  0] System Bundle (2.0.1)
[  1] [Active]   [ 20] Apache Felix Bundle Repository (1.4.2)
[  2] [Installed] [  9] Weighing scale agent publisher (1.0.0)
[  3] [Active]   [  6] The Physical World Ontology (1.1.0)
[  4] [Active]   [  6] wrap_mvn_org.universAAL.continua_ieee.x73.std_1.1.0 (0)
[  5] [Active]   [  5] User Interaction Bus (1.1.0)
[  6] [Active]   [  5] Context Bus (1.1.0)
[  7] [Active]   [  5] Service Bus (1.1.0)
[  8] [Active]   [  4] The Profiling Ontology (1.1.0)
[  9] [Active]   [  4] RDF/OWL Turtle serializer (1.1.0)
[ 10] [Active]   [  4] universAAL Data Representation Model (1.1.0)
[ 11] [Active]   [  4] x73 - Ontology for x73 devices (0.1.0)
[ 12] [Active]   [  3] OPS4J Pax ConfMan - Properties Loader (0.2.2)
[ 13] [Active]   [  3] jena.model_converter (1.1.0)
[ 14] [Active]   [  3] Soda-Pop as OSGi Bundle (1.1.0)
[ 15] [Active]   [  3] OPS4J Pax Logging - Service (1.6.2)
[ 16] [Active]   [  2] Apache Felix Configuration Admin Service (1.2.4)
[ 17] [Active]   [  2] OSGi Container (1.1.0)
[ 18] [Active]   [  2] wrap_mvn_jp.go.ipa.jgcl_1.0 (0)
[ 19] [Active]   [  2] Sun Microsystems' JDBC Driver for MySQL (5.1.17)
[ 20] [Active]   [  2] wrap_mvn_java3d_vecmath_1.3.1 (0)
[ 21] [Active]   [  2] Jena OSGi Bundle (1.0.0)
[ 22] [Active]   [  2] ACL Interfaces (1.1.0)
[ 23] [Active]   [  2] OPS4J Pax Logging - API (1.6.2)
[ 24] [Active]   [  2] Container Interfaces (1.1.0)
[ 25] [Active]   [  2] wrap_mvn_org.bouncycastle.jce.jdk13_144 (0)
[ 26] [Active]   [  2] wrap_mvn_java3d_j3d-core_1.3.1 (0)
[ 27] [Active]   [  1] Apache Felix Shell Service (1.4.1)
[ 28] [Active]   [  1] Apache Felix Shell TUI (1.4.1)
-> start 2

```

**Figure 15: Available bundles in OSGi framework**

```

Channel connected
MainChannel: /org/bluez/1063/hci0/dev_00_09_1F_80_0A_E0/chan65279
HDP channel path: /org/bluez/1063/hci0/dev_00_09_1F_80_0A_E0/chan65279
HDP data channel properties:
Device: /org/bluez/1063/hci0/dev_00_09_1F_80_0A_E0
Application: /org/bluez/health_app_2
Type: Reliable
Number of file descriptor: 60
CONNECTED
Waiting for input data frames
Data received
0xE2 0x00
0x00 0x32
0x80 0x00
0x00 0x00
0x00 0x01
0x00 0x2A
0x50 0x79
0x00 0x26
0x80 0x00
0x00 0x00
0x80 0x00
0x80 0x00
0x00 0x00
0x00 0x00
0x00 0x00
0x00 0x00
0x00 0x80
0x00 0x00
0x00 0x08
0x00 0x09
0x1F 0xFF
0xFE 0x80
0x0A 0xE0
0x05 0xDC

```

**Figure 16: HDP bluetooth channel successfully created**



```

Number of bytes received (call from native code): 54
Number of bytes received (call from JAVA): 54
Answering agent data frames
LOG: New APDU Received!
LOG: AARQ Processing: Starting...
LOG: AARQ Processing: Agent capable of Protocol IEEE 11073-20601 found
LOG: AARQ Processing: Checking PhdAssociationInformation...
LOG: AARQ Processing: Protocol Version: 80 00 00 00
LOG: AARQ Processing: Encoding Rules: 80 00
LOG: AARQ Processing: Nomenclature Version: 80 00 00 00
LOG: AARQ Processing: Functional Units: 00 00 00 00
LOG: AARQ Processing: System Type: 00 80 00 00
LOG: AARQ Processing: PHD correct!
LOG: AARQ Processing: Device Configuration of the Agent: 1500
LOG: AARQ Processing: AARQ SUCCESSFUL
CONNECTED
ASSOCIATED - OPERATING
SENDING APDU: e3 00 00 2c 00 00 50 79 00 26 80 00 00 00 80 00 80 00 00

Waiting for input data frames

->
->
-> stop 2
HDP manager successfully closed

```

**Figure 17: Stopping HDP manager bundle**

```

Number of bytes received (call from native code): 62
Number of bytes received (call from JAVA): 62
Answering agent data frames
LOG: New APDU Received!
LOG: Initiating processing of a received measurement from a Weighing Scale

MEASUREMENT: 71,60 kg
MEASUREMENT: Date of measurement: 10:19:31 - 03/08/2012
-----

```

**Figure 18: Real time weighing scale measurement received**

Important note: Weighing scale ontology seems that is not able to publish events to uAAL context bus now, so it should be revised by the responsible team. Publisher class at this sample (`Publisher.java`) is a well-formed class fully functional so as soon as WS ontology will be updated, WS events will be published.



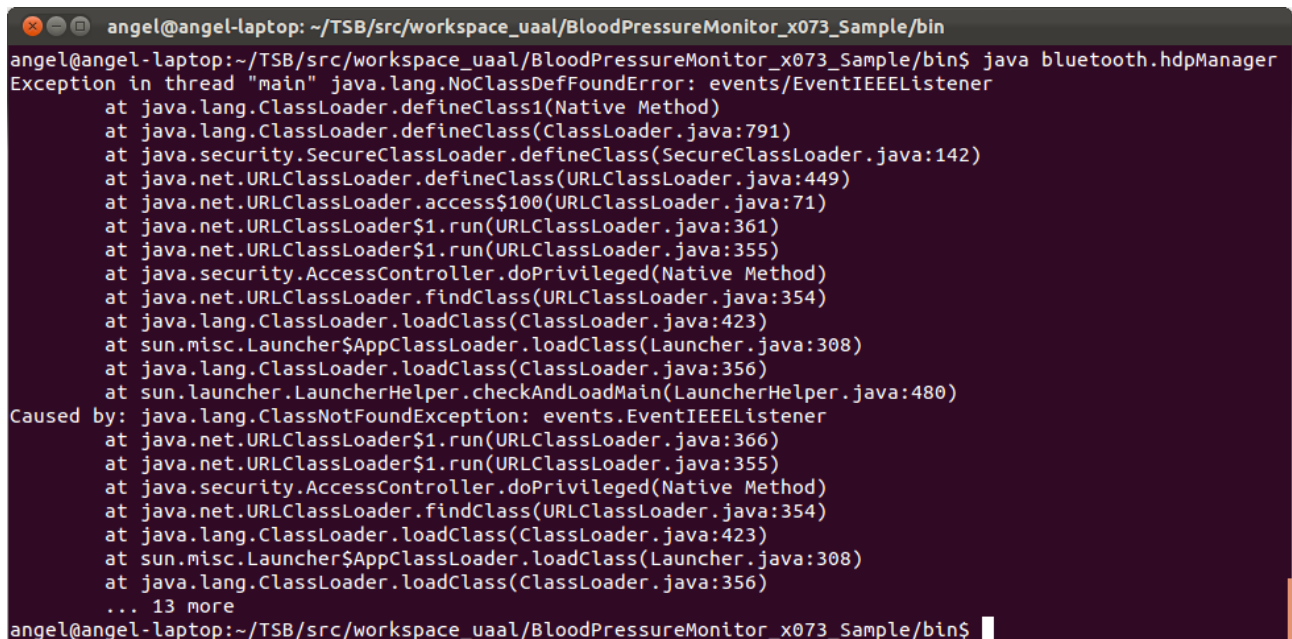
# Troubleshooting

1. Error: compiling any source class always returns a `java.lang.UnsupportedClassVersion`. Possible solution: Change your JDK to the official Oracle Java JDK or update its version.

```
$ java -version
java version "1.7.0_04"
Java(TM) SE Runtime Environment (build 1.7.0_04-b20)
Java HotSpot(TM) Server VM (build 23.0-b21, mixed mode)
```

**Table 8: UnsupportedClassVersion error**

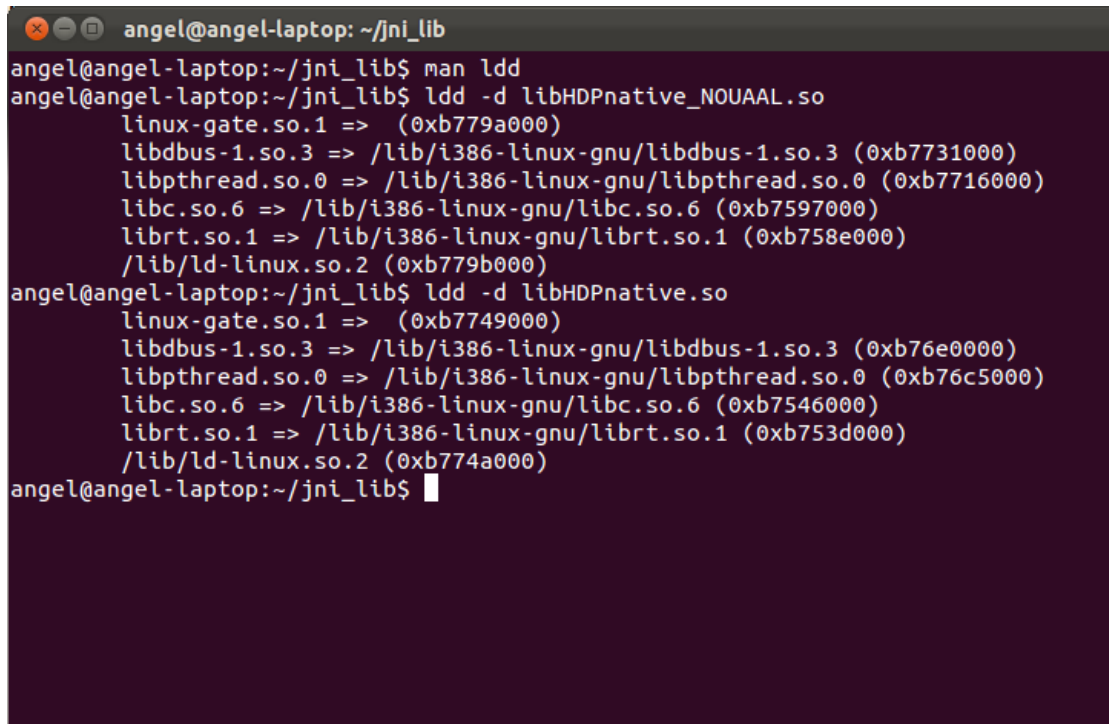
2. Error: launching manager application with `x073implementation.jar` file imported always returns a `java.lang.NoClassDefFoundError`. Possible solution: Check the classpath or directly copy the jar file inside `/usr/lib/jvm/java-7-oracle/jre/lib/ext` path folder.



```
angel@angel-laptop: ~/TSB/src/workspace_uaal/BloodPressureMonitor_x073_Sample/bin
angel@angel-laptop:~/TSB/src/workspace_uaal/BloodPressureMonitor_x073_Sample/bin$ java bluetooth.hdpManager
Exception in thread "main" java.lang.NoClassDefFoundError: events/EventIEEEListener
    at java.lang.ClassLoader.defineClass1(Native Method)
    at java.lang.ClassLoader.defineClass(ClassLoader.java:791)
    at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
    at java.net.URLClassLoader.defineClass(URLClassLoader.java:449)
    at java.net.URLClassLoader.access$100(URLClassLoader.java:71)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:361)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:423)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:356)
    at sun.launcher.LauncherHelper.checkAndLoadMain(LauncherHelper.java:480)
Caused by: java.lang.ClassNotFoundException: events.EventIEEEListener
    at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:423)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:356)
    ... 13 more
angel@angel-laptop:~/TSB/src/workspace_uaal/BloodPressureMonitor_x073_Sample/bin$
```

**Figure 19: NoClassDefFoundError error**

3. Error: launching manager application always returns an `undefined symbol` error. Possible solution: Execute `ldd -d` command (print shared library dependencies) in the OS console in the folder where native libraries (remember: `.so` files) where stored. If these libraries has successfully been linked in your system, you should get an output similar as follows:

A terminal window with a dark background and light text. The window title is 'angel@angel-laptop: ~/jni\_lib'. The user has entered two 'ldd' commands to check the linking of two shared libraries. The first command is 'ldd -d libHDPnative\_NOUAAL.so' and the second is 'ldd -d libHDPnative.so'. Both commands show a list of linked libraries and their addresses, indicating successful linking. The output for the first command lists: linux-gate.so.1, libdbus-1.so.3, libpthread.so.0, libc.so.6, librt.so.1, and /lib/ld-linux.so.2. The output for the second command lists: linux-gate.so.1, libdbus-1.so.3, libpthread.so.0, libc.so.6, librt.so.1, and /lib/ld-linux.so.2. The prompt 'angel@angel-laptop:~/jni\_lib\$' is visible at the end of the second command's output.

```
angel@angel-laptop: ~/jni_lib
angel@angel-laptop:~/jni_lib$ man ldd
angel@angel-laptop:~/jni_lib$ ldd -d libHDPnative_NOUAAL.so
        linux-gate.so.1 => (0xb779a000)
        libdbus-1.so.3 => /lib/i386-linux-gnu/libdbus-1.so.3 (0xb7731000)
        libpthread.so.0 => /lib/i386-linux-gnu/libpthread.so.0 (0xb7716000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7597000)
        librt.so.1 => /lib/i386-linux-gnu/librt.so.1 (0xb758e000)
        /lib/ld-linux.so.2 (0xb779b000)
angel@angel-laptop:~/jni_lib$ ldd -d libHDPnative.so
        linux-gate.so.1 => (0xb7749000)
        libdbus-1.so.3 => /lib/i386-linux-gnu/libdbus-1.so.3 (0xb76e0000)
        libpthread.so.0 => /lib/i386-linux-gnu/libpthread.so.0 (0xb76c5000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7546000)
        librt.so.1 => /lib/i386-linux-gnu/librt.so.1 (0xb753d000)
        /lib/ld-linux.so.2 (0xb774a000)
angel@angel-laptop:~/jni_lib$
```

**Figure 20: Native libraries successfully linked**

4. Error: launching manager application always returns a `java.lang.UnsatisfiedLinkError`. Possible solution: Check the classpath or update your `LD_LIBRARY_PATH` global variable.

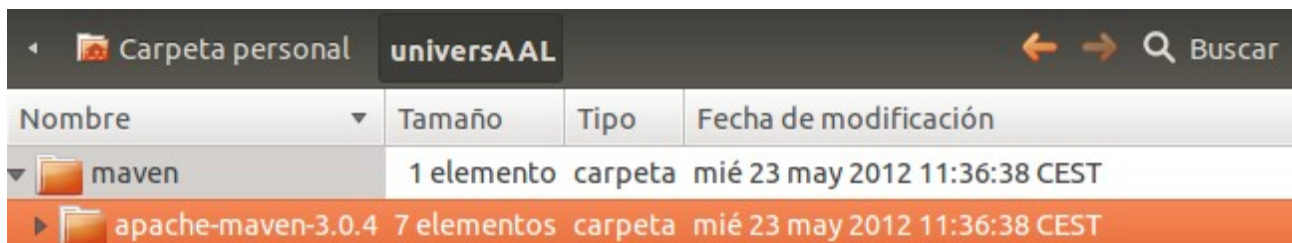
# Appendix A

## Setting up uAAL development environment on GNU/Linux

Important note: this guidelines are not official documentation supported and maintained by universAAL Consortium. This document has been elaborated (of our own necessity) in order to help any GNU/Linux user starting setting up the minimum required environment to run and develop AAL services based on uAAL middleware. Official information available in the universAAL Developer Handbook<sup>10</sup> only covers this process in Microsoft Windows environments so due to this lack we decide to create this simple guide from scratch. As any first release it isn't exempt from errors, so we apologize for any inconvenience. Please, feel free to comment or modify anything of this appendix. Any suggestion will be appreciated.

We will assume that you are using a 32-bit Linux Ubuntu 11.10 operating system.

1. Install an appropriate JDK. We recommend Oracle-jdk7.
2. Create directory tree. For example, we will use as root of universaal environment a folder in home directory.
  1. Create `~/universAAL` folder. Please note that `~` symbol refers to home directory.
3. Install maven software:
  1. Download the latest version package from the official URL<sup>11</sup>
  2. Extract the files in `~/universAAL/maven`. This will as seen in screenshot:



3. Set up the proper environment variables. For doing this permanent, we will have to edit the `.bashrc` file in home folder. In console terminal, write `gedit ~/.bashrc` (at the end of that text file, add next lines):

<sup>10</sup> <http://forge.universaal.org/gf/>

<sup>11</sup> <http://maven.apache.org/download.html>

```
export M2_HOME=/home/username/universAAL/maven/apache-maven-3.0.4
export M2=$M2_HOME/bin
export MAVEN_OPTS="-Xms256m -Xmx512m"
export PATH=$M2:$PATH
export JAVA_HOME=/usr/lib/jvm/java-7-oracle
export PATH=$JAVA_HOME/bin:$PATH
```

4. Check if it is working correctly. Enter the `mvn --version` in console terminal. If everything is okay, we have to see information about our computer and the version of maven software just installed:

```
angel@angel-laptop:~$ mvn --version
Apache Maven 3.0.4 (r1232337; 2012-01-17 09:44:56+0100)
Maven home: /home/angel/TSB/uAAL/Maven/apache-maven-3.0.4
Java version: 1.7.0_04, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-7-oracle/jre
Default locale: es_ES, platform encoding: UTF-8
OS name: "linux", version: "3.0.0-19-generic-pae", arch: "i386", family: "unix"
angel@angel-laptop:~$
```

5. Once we have maven software running in our machine, it is time to install a SVN client and to obtain the universAAL resources (source code). This guidelines will assume that you are registered in universAAL GForce web page<sup>12</sup> and you have been granted access to **Middleware**, **Ontologies** and **Support** projects.







- Install any SVN client which can be retrieved from Ubuntu Software Center or similar (for instance: RabbitVCS<sup>13</sup> or SVN Workbench<sup>14</sup>).
- Create the folders (for instance):
  - `~/universAAL/svn/middleware/`
  - `~/universAAL/svn/ontologies/`
  - `~/universAAL/svn/support/`
- Get uAAL source code from universAAL repository accordingly with SVN client previously chosen. Wait until all files are downloaded.
  - SVN source: `http://forge.universaal.org/svn/middleware`
  - SVN source: `http://forge.universaal.org/svn/ontologies`
  - SVN source: `http://forge.universaal.org/svn/support`

---

<sup>12</sup> <http://forge.universaal.org/gf/>

<sup>13</sup> <http://rabbitvcs.org/>

<sup>14</sup> <http://pysvn.tigris.org/docs/WorkBench.html>

Carpeta personal TSB uAAL src svn <b>middleware</b>			
Nombre	Tamaño	Tipo	Fecha de modificación
▶  branches	1 elemento	carpeta	lun 18 jun 2012 16:41:41 CEST
▶  oldtrunk	14 elementos	carpeta	lun 18 jun 2012 16:41:30 CEST
▶  rundir	2 elementos	carpeta	lun 18 jun 2012 16:41:30 CEST
▶  sandboxes	13 elementos	carpeta	lun 18 jun 2012 16:45:26 CEST
▶  tags	10 elementos	carpeta	lun 18 jun 2012 16:47:50 CEST
▶  trunk	4 elementos	carpeta	lun 18 jun 2012 16:41:37 CEST

5. Download Eclipse IDE for GNU/Linux OS. As recommended in the Developer Handbook of universAAL, we will get the 3.5.2 version from Eclipse repositories<sup>15</sup>.

- Extract the files on (for instance) `~/universAAL/eclipse/` (create the folder).
- As a recommendation, create a new (clean) workspace folder. For example, `~/universAAL/uAAL_workspace/`. Each universAAL project will be a sub-folder of this one.
- Start Eclipse IDE and select `~/universAAL/uAAL_workspace/` as workspace for this session (this name is not important now, since we are only adding plugins to Eclipse but is a good practice).

8. Install Eclipse required plugins:

- Install Subversion plugin<sup>16</sup> for Eclipse IDE using the normal procedure.
  - Following the recommendation from the universAAL Developer Handbook, install a svn command line client. In terminal console with root privileges, enter the command `sudo apt-get install subversion`.
- Install M2Eclipse plugin<sup>17</sup> for Eclipse IDE.
- Install Pax Runner plugin<sup>18</sup> for Eclipse IDE.

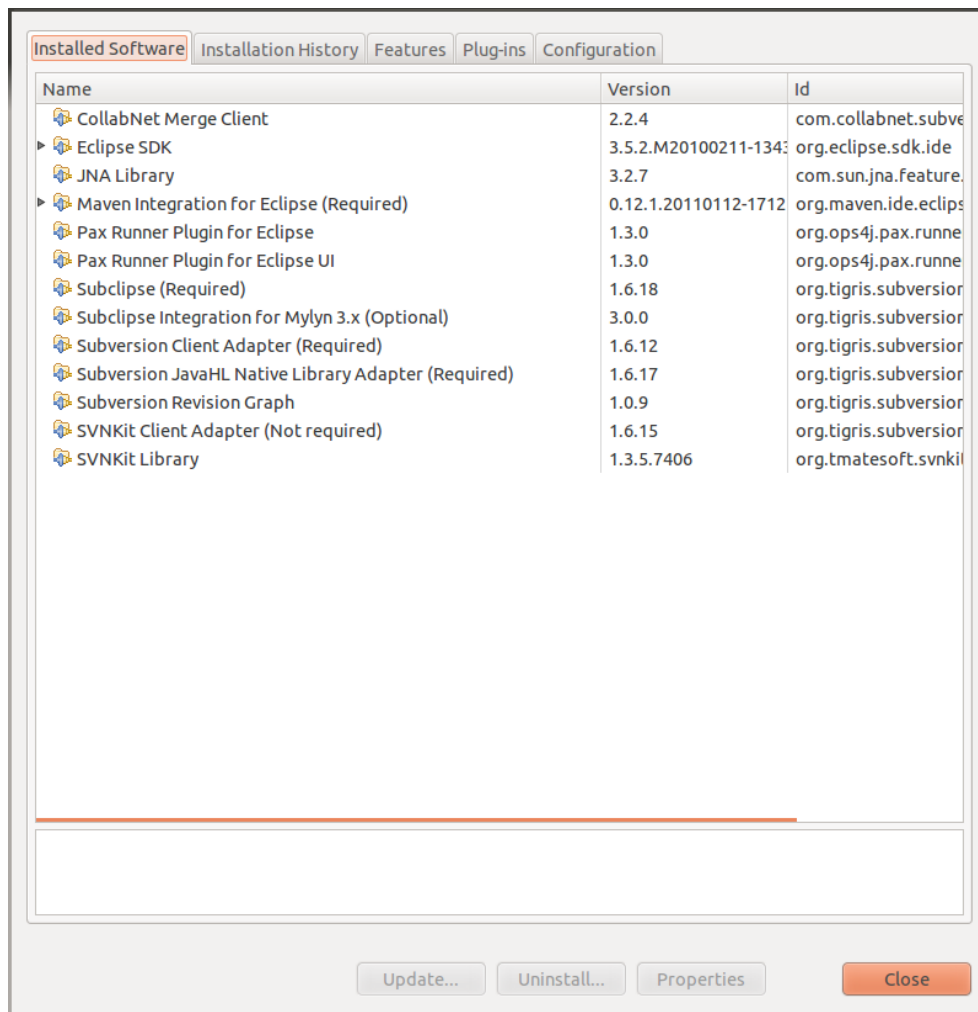
9. After installing all the additional software for Eclipse, the Installed Software window should look like this:

<sup>15</sup><http://archive.eclipse.org/eclipse/downloads/drops/R-3.5.2-201002111343/download.php?dropFile=eclipse-SDK-3.5.2-linux-gtk.tar.gz>

<sup>16</sup>[http://subclipse.tigris.org/update\\_1.6.x](http://subclipse.tigris.org/update_1.6.x)

<sup>17</sup><http://m2eclipse.sonatype.org/sites/m2e>

<sup>18</sup><http://www.ops4j.org/pax/eclipse/update/>



## 10. Configure Eclipse to look for the Java JDK.

- Open the file `eclipse.ini` in `~/universAAL/eclipse/` folder.
- Look for `-vmargs` line. Above it, write **two lines** (please keep in mind that the previous two lines have to be added as two lines, not one). Save and exit.

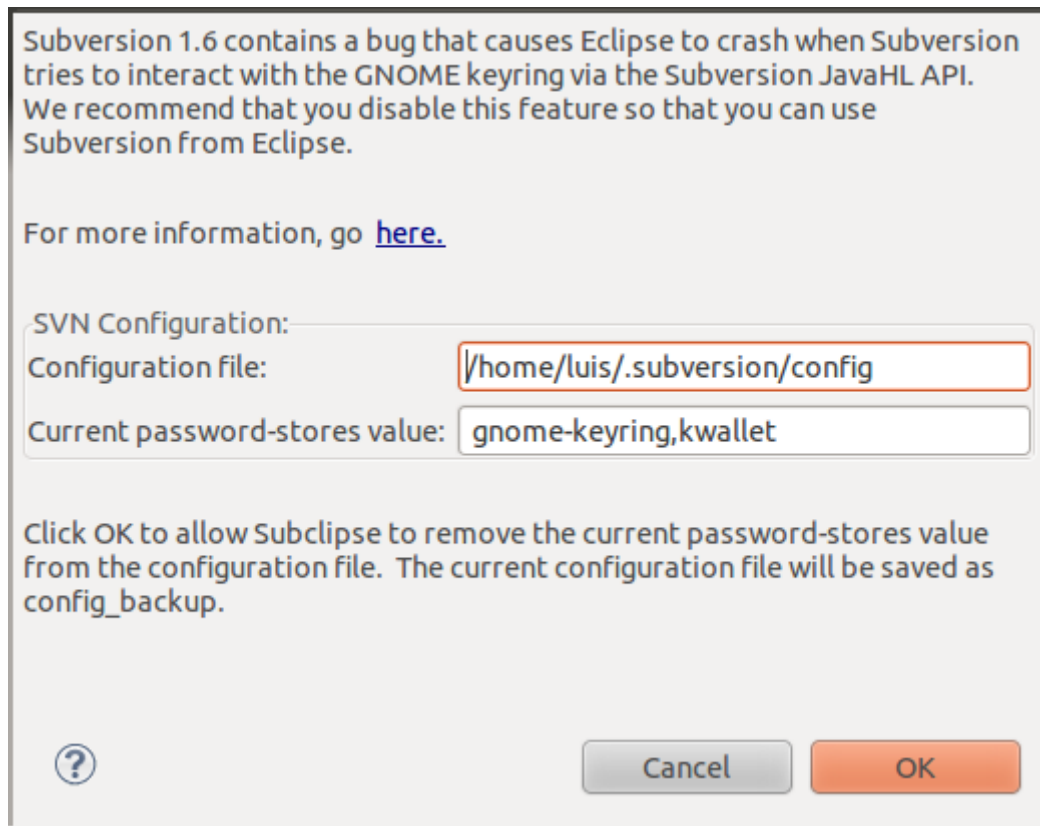
```
-vm
/usr/lib/jvm/java-7-oracle/jre/bin/java
```

## 11. Configure Maven options in the local system checking that the folder `~/ .m2` is present.

- Copy the `settings.xml` file available in **Resources** folder<sup>19</sup> (recently created at your computer) to your local `.m2` folder.

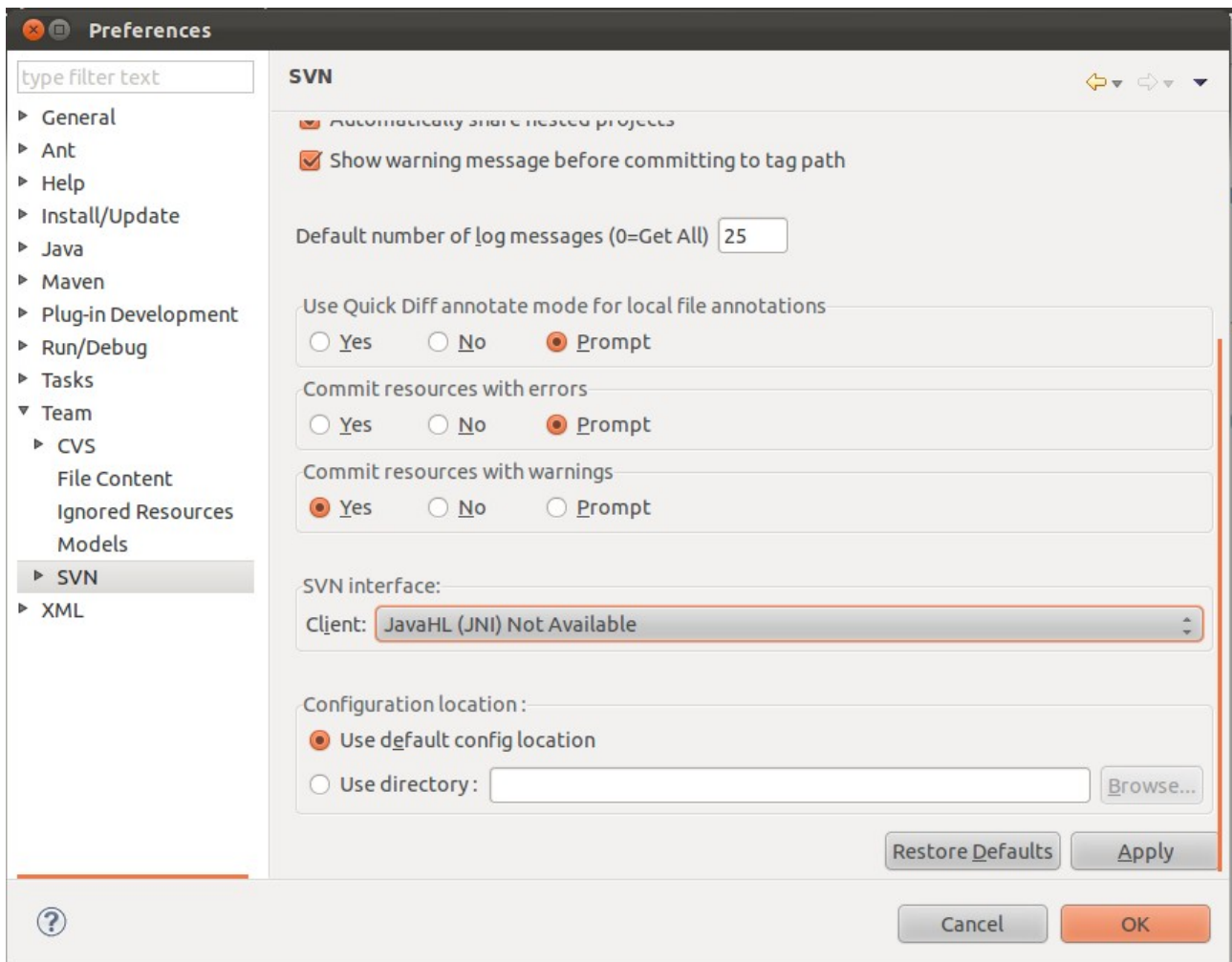
<sup>19</sup>~/universAAL/svn/support/trunk/resources/maven-repo-settings

12. Restart Eclipse. You may see a warning message saying that "**Subversion 1.6 has a bug with GNOME keyring via the Subversion JavaHL API**". Just press OK button.



13. Probably, you will have problems in order to have the JavaHL library working properly. This is used by Subversion as a high level API and serve as the JNI bridge between Java code and native libraries. You can check if the problem is present going to **Window** → **Preferences** → **Team** → **SVN** (inside Eclipse IDE). If SVN Interface menu option shows "Not available", follow the next steps. If it shows JavaHL(JNI) 1.6.x, you should go without problems to the point number 14.





- In GNU/Linux OS, the JavaHL library is in a separate package dependent on the main subversion package. To get it launch the command console and follow next steps:

```
sudo apt-get install libsvn-java  
find / -name libsvnjavahl-1.so > ~/pathJavaHL
```

- Open the file **eclipse.ini** in **~/universAAL/eclipse/** folder.
- After **-vmargs** line, add the following line:  
**-Djava.library.path=/usr/lib/jni/** (without quotes). This will create a file text with the path of the JavaHL library inside it. This information will be used in the next steps (typically, it will be **/usr/lib/jni**). Save and exit.

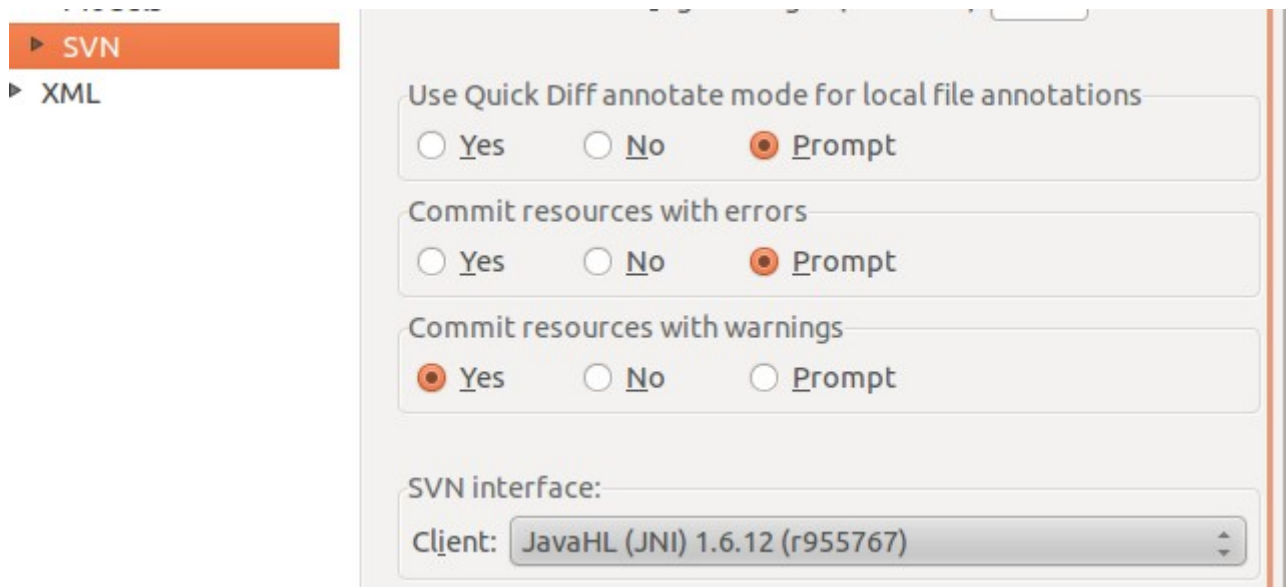


```

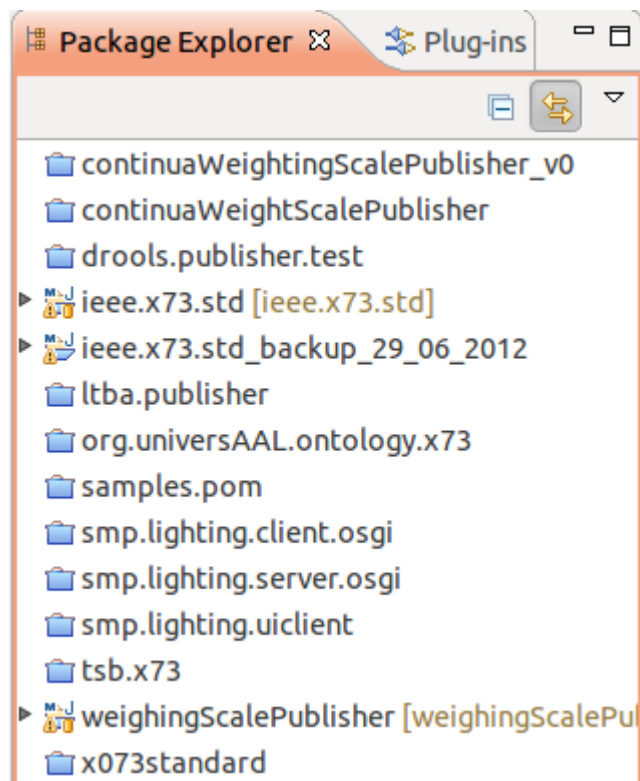
eclipse.ini
-startup
plugins/org.eclipse.equinox.launcher_1.0.201.R35x_v20090715.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.gtk.linux.x86_1.0.200.v20090520
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
-vm
/usr/lib/jvm/java-7-oracle/jre/bin/java
-vmargs
-Djava.library.path=/usr/lib/jni/
-Xms40m
-Xmx256m

```

- Open again Eclipse IDE, and check if the SVN Interface option described before has a correct value.



14. At this point, you should be able to import any uAAL project as well as run it correctly.



# Troubleshooting

1. The examples (and any universAAL bundle) will not run if the file `sodapop.key` is not in the proper folder. For solving this problem, look for any `sodapop.key` file in the `~/universAAL/svn/support/trunk/samples` local folder, and copy it into `[workspace_folder]/rundir/mw.bus.model`. If any of the folders in the later path does not exist, just create them.