

Ecosystem infrastructure for smart and personalised inclusion   
and PROSPERITY for ALL stakeholders

Internal Report on Architecture Design and Development

ANNEX II: UCH Java Client API

|  |  |
| --- | --- |
| Project Acronym | **Prosperity4All** |
| Grant Agreement number | **FP7-610510** |

|  |  |
| --- | --- |
| Deliverable number | **Work conducted for D203.1** |
| Work package number | **WP203** |
| Work package title | **Collaborative development tools/Environments**  **T203.3 Runtime Environment** |
| Authors | **Marios Komodromos, Christos Mettouris** |
| Status | **Draft** |
| Dissemination Level | **Consortium** |
| Delivery Date | **11/07/2016** |
| Number of Pages | **13** |

Keyword List

UCH, URC, ARE, AsTeRiCS, Java

Version History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision | Date | Author | Organisation | Description |
| 1 | 04/02/2016 | Marios Komodromos | UCY | Initial authoring |
| 2 | 09/02/2016 | Christos Mettouris | UCY | Update |
| 3 | 11/07/2016 | Christos Mettouris, Marios Komodromos | UCY | Final version |

Table of Contents

[Executive Summary 1](#_Toc457489977)

[1 Introduction 2](#_Toc457489978)

[1.1 What is UCH java client library 2](#_Toc457489979)

[1.2 Why do we need UCH java client 2](#_Toc457489980)

[2 Installation 3](#_Toc457489981)

[2.1 Prerequisites 3](#_Toc457489982)

[2.2 Download 3](#_Toc457489983)

[2.3 Installation 3](#_Toc457489984)

[2.3.1 Install UCH java client to eclipse 3](#_Toc457489985)

[2.3.2 Use UCH java client in your project 3](#_Toc457489986)

[2.3.3 Install UCH server 4](#_Toc457489987)

[2.4 Test 4](#_Toc457489988)

[3 Technical 5](#_Toc457489989)

[3.1 Overview 5](#_Toc457489990)

[3.2 Details 6](#_Toc457489991)

[3.2.1 UCH Discovery 6](#_Toc457489992)

[3.2.2 UCH component descriptions (out-of-session requests) 6](#_Toc457489993)

[3.2.3 UCH component sessions 7](#_Toc457489994)

[3.3 Available tests 8](#_Toc457489995)

[4 UCH Java Client in AsTeRICS 9](#_Toc457489996)

[4.1 Discovering URC component variables 9](#_Toc457489997)

[4.2 Importing the UCH Java Client Library 9](#_Toc457489998)

[4.3 Connecting to the UCH server 10](#_Toc457489999)

[4.4 Changing UCH component value 10](#_Toc457490000)

[5 References 11](#_Toc457490001)

List of Tables

**No table of figures entries found.**

List of Figures

[Figure 1:Local network example 5](#_Toc456024250)

[Figure 2: UCH discovery example 6](#_Toc456024251)

[Figure 3: UchCommunicator object creation 6](#_Toc456024252)

[Figure 4: Retrieve information from UCH server 6](#_Toc456024253)

[Figure 5: Retrieve SocketDescription information 7](#_Toc456024254)

[Figure 6: UchSession creation example 7](#_Toc456024255)

[Figure 7: GetValuesRequest example 8](#_Toc456024256)

# Executive Summary

This document describes the usage of the UCH Java Client API developed by UCY in the context of Task 203.3 of WP203. It is the ANNEX II of the Internal Report on Architecture Design and Development.

# Introduction

## What is UCH java client library

UCH java client is a library that enables communication between the UCH server [1] and clients who want to access URC [2] components (on the UCH server).

## Why do we need UCH java client

According to the WP203.3 task, we target to integrate three different platforms: ARE [3], URC and MyUI [4]. The ARE is a platform implemented with the Java programming language, so in order to communicate with the UCH, it was necessary to use a java library that follows URC-HTTP protocol. Although the URC community provides a number of libraries to communicate with UCH, a java library was not included as one of them. Hence, under the WP203.3, a UCH java client was developed by UCY.

# Installation

## Prerequisites

1. Java SE Development Kit 7
2. Eclipse IDE

## Download

* Download the Runtime Environment content from this link: <https://github.com/mariokom/RuntimeEnvironment>
* extract the contents of the folder to a preferred location
* Important: let <**REFolderPath**> be this location (it will be needed later on)

1. Path to UCH java client: <**REFolderPath**>/Source Code/AsTeRICS/UCH Java Library
2. Path to UCH server (standalone): <**REFolderPath**>/UCH

## Installation

### Install UCH java client to eclipse

Follow these steps to install UCH java client in eclipse:

1. Download UCH java client zip file
2. Extract the contents to a desired location
3. In eclipse click File 🡪 New 🡪 Java Project
4. On the window that will appear: uncheck “use default location” checkbox🡪click the browse button 🡪 select the location where the UCH java client is saved
5. Click Finish

### Use UCH java client in your project

Follow these steps use UCH java client in your project:

1. Add UchJavaClient jar which is located in “<**REFolderPath**>/Source Code/AsTeRICS/UCH Java Library/lib”, to the project’s build path.
2. UchJavaClient jar has a dependency to the JmDNS [6] library. Add the JmDNS jar file located in “<**REFolderPath**>/Source Code/AsTeRICS/UCH Java Library/lib” to the projects build path.

### Install UCH server

Since we have the standalone version of the UCH server, no installation is needed.

## Test

Follow these steps to test UCH java client:

1. Open the terminal and navigate to UCH server standalone folder <**REFolderPath**>/UCH.
2. Run (double click) <**REFolderPath**>/UCH/startUCH.bat file.
3. Open the UCH java client project in eclipse, navigate to ‘test’ package and run (“run as application”) the test classes to confirm that the library works as expected.

# Technical

Before starting to use the UCH java client library, it is advised to read the URC-HTTP protocol documentation [5]. An individual with a basic understanding of URC-HTTP protocol should find it much easier to integrate the library in its own project.

## Overview

The UCH java client library (UCH jClient for convenience) is able to do three major things:

1. Discover UCH server’s location in the local network with a zero-configuration networking (zeroconf) implementation. With JmDNS [6], the UCH java client is searching in the local network for the UCH zeroconf service. If found, the UCH jClient is able to retrieve the IP address of the machine that hosts the server and proceed with the communication procedure.

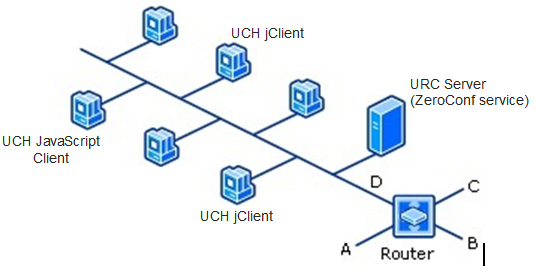


Figure 1:Local network example

1. Make out-of-session URC-HTTP requests to discover which URC components are running on the UCH server and retrieve all the available information for each one of them (static information). Using the **UCH-communicator object**, any java client can query the UCH server repository and obtain the UIlist, an xml formatted list with the running URC components descriptions. Every element of this list, contains information needed for the communication, such as the component id and its Socket Description.
2. When needed, the UCH-communicator object can create **session objects** which are following the URC-HTTP protocol as described in the URC-HTTP documentation [5]. Every session object can sent in-session requests to monitor/change the state of the URC components using a unique session id.

## Details

### UCH Discovery

To simplify the connection procedure with UCH server, UCH jClient supports an auto-discovery feature. This means that when the server and the client exist in the same local network, there is no need for manual IP configuration. A properly implemented UCH server should run a UCH zeroconf service. The client will search for this service and - if found – it will return a set of information about the hosting device. The client can then read the information and use it to connect with the server. In figure 2 you can see a snippet of code which autodiscovers the UCH server’s IP address.

Set<String> ipSet = UchDiscovery.getUchServerIPs();
String uchIP = ipSet.iterator().next();
System.out.println("This is the IP address of the UCH server: " + uchIP);

Figure 2: UCH discovery example

### UCH component descriptions (out-of-session requests)

The UCH server holds the information for every available URC component. This information is structured in XML format and is called ‘UIlist’. For every UI (URC component), you can retrieve its ID, name, SocketDescription (as a **SocketDescription** object), TargetDescription (as a **TargetDescription** object) etc. To be able to query the UCH server you must first create a **UchCommunicator object** and initialize it with an IP address and a port number (figure 3).

UchCommunicator uchCommunicator = new UchCommunicator(uchIP, 8080, false);

Figure 3: UchCommunicator object creation

After the creation, it is possible to retrieve data from the server as the example in figure 4.

String randomId = uchCommunicator.getAllUiIds().get(0);
String uiName = uchCommunicator.getUiName(randomId);
SocketDescription socketDescription = uchCommunicator.getSocketDescription(randomId);

Figure 4: Retrieve information from UCH server

The **SocketDescription** (SD) object retrieved from the UCH server (figure 4), holds information about the component SD such as the description, the creator, the date created but most importantly the variables. In order to have a better understanding, a SD variable can be the temperature variable for a thermostat component, or the light intensity variable for a lamp. The figure below (figure 5) depicts how this information can be extracted from the object.

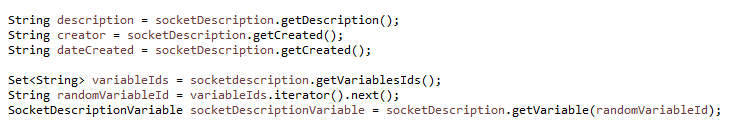


Figure 5: Retrieve SocketDescription information

### UCH component sessions

Besides getting the static description of the URC components, UCH jClient is capable of opening sessions with the actual components. These sessions allow clients to monitor or change the state of a remote URC-compliant device. To achieve something like this, you should first request from the UCH server to open a new session with a specific component and the server should return a newly generated session key. See figure 6 for the example

String randomId = uchCommunicator.getAllUiIds().get(0);
String httpUrcURI = uchCommunicator.getProtocolURI(randomId, UchCommunicator.URC_HTTP);
String sessionId = uchCommunicator.createSession_getRequest(httpUrcURI);
UchSession session = uchCommunicator.getSession(sessionId);

Figure 6: UchSession creation example

Futhermore, it is possible to call functions as described by the URC-HTTP protocol documentation [5]. The information needed for each call varies according to the type of the call. The example in figure 7, depicts a ‘readValues’ function call in a thermostat component.

GetValuesRequest getValuesRequest = new GetValuesRequest();
getValuesRequest.addGetElement(new GetElement("currentRoomTemp"));
String gvrText = session.getValues(getValuesRequest);
GetValuesResponse getValuesResponse = new GetValuesResponse(gvrText);
System.out.println(getValuesResponse);

Figure 7: GetValuesRequest example

Examples for other types of requests can be found in the test classes of the project.

## Available tests

To check the capabilities of the UCH jClient and to provide additional examples for the developers, five test classes where created (they can be found in the test package of uchJavaClient project):

* **UchDiscoveryTest** – Depicts how a client can discover information for a Uch Server that exists in the local network,
* **UiListTest** – Retrieves static information of the components from Uch Server
* **SessionTest** – Creates a session with a component and calls the available session functions
* **SessionResponsesTest** – Depicts the use of the objects responsible for parsing the session responses.
* **CompinationTest** – This is a complete example of a UCH jClient application.

# UCH Java Client in AsTeRICS

As mentioned before, UCH Java Client was initially created in order to be used by an AsTeRICS (ARE) plugin. This chapter targets to help the developer creating a new UCH component that follows AsTeRICS specifications.

**Recommended prior knowledge**:

* AsTeRICS developer manual [3], Section 3: A Quick Guide to AsTeRICS Plugin Development.
* Universal Remote Console framework [2]: Socket description.

## Discovering URC component variables

To identify the URC component variables we have to start the UCH server and visit ‘http://localhost:8080/UCH/GetCompatibleUIs. After traversing through the xml elements we can identify the component that we are interested in and visit the URL of the corresponding Socket description. In the Socket description you will find the available variables of the specific device.

## Importing the UCH Java Client Library

Firstly, we need to create an AsTeRICS component by following the instructions in the AsTeRICS developer manual (Section 3 of [3]). We will take into consideration the URC component variables from section 4.1. For example, a thermostat component (see D203.1 – section 6) could have a temperature value variable and a measurement unit variable (eg Celsius or Fahrenheit). Having in mind these two parameters and the solution we plan to build, we will construct the AsTeRICS component.

After creating the component, the developer will end up with the standard AsTeRICS component folder structure. In the components lib folder, two jar files should be added:

* “UchJavaClient-1.1.jar” jar which is located in “<**REFolderPath**>/Source Code/AsTeRICS/UCH Java Library/lib” folder.
* JmDNS jar file located in the same folder.

Additionally, these two jar files should be added in project’s build path, in the component’s ‘build.xml’ file and in component’s MANIFAST file. Having completed these steps, the library can be used as part of the AsTeRICS component.

## Connecting to the UCH server

As OSGi architecture commands, every component should contain a *start*()method which wraps the code to be executed before the component starts its life cycle. In the *start*()method, the developer should establish a connection between the AsTeRICS component and the UCH server. In order to connect to the UCH server, we must know its IP address. This address could be retrieved automatically (see Figure 2) or manually through the component parameters (see Figure 3), meaning that we will expect from the model designer to enter the UCH server IP address.

## Changing UCH component value

Usually, URC component values are altered when data are received through an *Input Port* or when an event occurs through an *Event Listener Port* (Section 7 of [3]). The developer can alter any URC component value as shown in Figure 8 below.

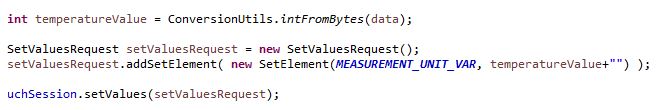


Figure 8: Changing URC component value

Note that the *‘MEASUREMENT\_UNIT\_VAR’* variable shown in Figure 8, is the name of the UCH component variable as described in URC component socket description (4.2)

# References

1. URC Consortium: Universal Control Hub 1.0 (ATR), Online: <http://www.openurc.org/TR/uch1.0-20131217/>
2. URC Consortium: Overview of the Universal Remote Console framework, Online: <http://www.openurc.org/TR/urc-overview2.0-20130218/>
3. The AsTeRICS Developer Manual, Online: [http://www.AsTeRICS.eu/download/DeveloperManual.pdf](http://www.asterics.eu/download/DeveloperManual.pdf)
4. OSGi - The Dynamic Module System for Java, Online: <http://www.osgi.org/Main/HomePage>
5. URC Consortium: URC-HTTP Protocol 2.0 (ATR), Online: <http://www.openurc.org/TR/urc-http-protocol2.0-20131217/>
6. JmDNS java library for a zeroConf implementation:

<http://jmdns.sourceforge.net/>