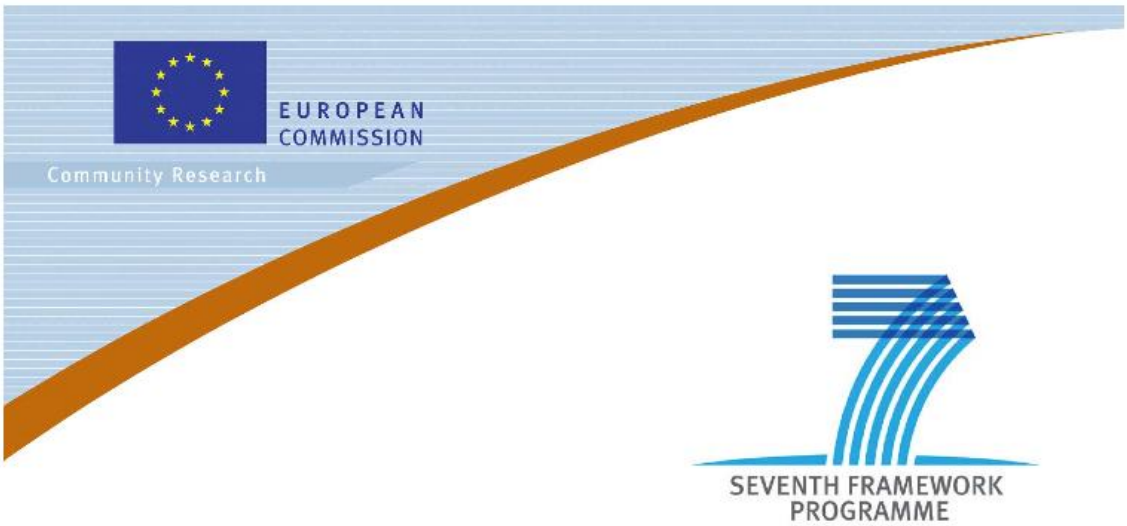


Private Public Partnership Project (PPP)

Large-scale Integrated Project (IP)

D.13.3.3: FI-WARE Installation and Administration Guide

Project acronym: FI-WARE
Project full title: Future Internet Core Platform
Contract No.: 285248
Strategic Objective: FI.ICT-2011.1.7 Technology foundation: Future Internet Core Platform
Project Document Number: ICT-2011-FI-285248-WP13-D.13.3.3
Project Document Date: 2014-04-30
Deliverable Type and Security: Public
Author: FI-WARE Consortium
Contributors: FI-WARE Consortium

1.1 Executive Summary

This document describes the installation and administration process of each Generic Enabler developed within in the "Advanced Middleware and Web-based User Interfaces" chapter. The system requirements for the installation of a Generic Enabler are outlined with respect to necessary hardware, operating system and software. Each GE has a section dedicated to the software installation and configuration process as well as a section, which describes sanity check procedures for the system administrator to verify that the GE installation was successful.

1.2 About This Document

The "FI-WARE Installation and Administration Guide" comes along with the software implementation of components, each release of the document referring to the corresponding software release (as per D.x.2), to facilitate the users/adopters in the installation (if any) and administration of components (including configuration, if any).

1.3 Intended Audience

The document targets system administrators as well as system operation teams of FI-WARE Generic Enablers from the FI-WARE project.

1.4 Chapter Context

FI-WARE will enable smarter, more customized/personalized and context-aware applications and services by the means of a set of assets able to gather, exchange, process and analyze massive data in a fast and efficient way. Nowadays, several well-known free Internet services are based on business models that exploit massive data provided by end users. This data is exploited in advertising or offered to 3rd parties so that they can build innovative applications. Twitter, Facebook, Amazon, Google and many others are examples of this.

The Advanced Middleware and Web User Interface (UI) Architecture chapter (aka. MiWi) of FI-WARE offers Generic Enablers from two different but related areas:

Advanced Middleware

The high-performance middleware that is backward compatible with traditional Web services (e.g. REST) but offers advanced features and performance and dynamically adapts to the communication partners and its environment. A novel API separates WHAT data needs to be communicated from WHERE the data come from within the native data structures of the application, and HOW the data should be transmitted to the target. Additionally, the middleware offers "Security by Design" through a declarative API, where the application defines the security requirements and policies that apply to its data, which are then automatically enforced by the middleware. The middleware uses of the security functionality offered by the Security chapter of FI-WARE.

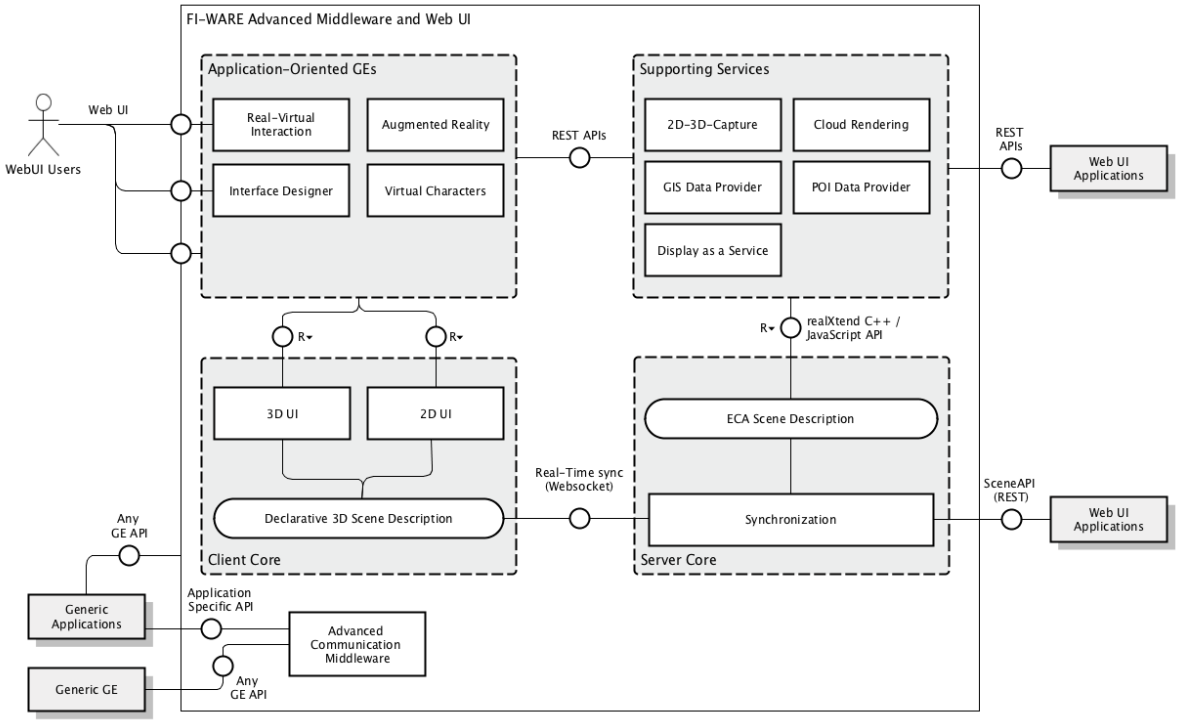
Advanced Web-based User Interface (Web UI)

In order to become widely visible and adopted by end users, the FI-WARE Future Internet platform must not only offer server functionality but must also offer much improved user experiences. The objective is to significantly improve the user experience for the Future Internet by adding new user input and interaction capabilities, such as interactive 3D graphics, immersive interaction with the real and virtual world (Augmented Reality), virtualizing and thus separating the display from the (mobile) computing device for ubiquitous operations, and many more. The technology is based on the Web technology stack, as the Web is quickly becoming THE user interface technology supported on essentially any (mobile) device while already offering advanced rich media capabilities (e.g. well-formatted text, images, video). First devices are becoming available that use Web technology even as the ONLY user interface technology. The Web design and programming environment is well-known to millions of developers that allow quick uptake of new technology, while offering a proven model for continuous and open innovation and improvement.

These two areas have been combined because highly interactive (2D/3D) user interfaces making use of service oriented architectures have strong latency, bandwidth, and performance requirements regarding the middleware implementations. An example is the synchronization service for real-time shared virtual worlds or the machine control on a factory floor that must use the underlying network and computing hardware as efficiently as possible. Generic Enables are

provided to make optimal use of the underlying hardware via Software Defined Networking in the Middleware (SDN, using the GEs of the Interface to Networks and Devices (I2ND) Architecture chapter) and the Hardware Support in the 3D-UI GE (see FIWARE.ArchitectureDescription.MiWi.3D-UI) that provides access to GPU and other parallel compute functionality that may be available. The following diagram shows the main components (Generic Enablers) that comprise the first release of FI-WARE Data/Context chapter architecture.

The following diagram shows the main components (Generic Enablers) that comprise FI-WARE Advanced Middleware and Web-based User Interfaces chapter architecture.



More information about the Advanced Middleware and Web-based UI Chapter and FI-WARE in general can be found within the following pages:

http://wiki.fi-ware.org

Advanced Middleware and Web UI Architecture

Materializing Advanced Middleware and Web User Interfaces in FI-WARE

1.5 Structure of this Document

The document is generated out of a set of documents provided in the public FI-WARE wiki. For the current version of the documents, please visit the public wiki at http://wiki.fi-ware.org/

The following resources were used to generate this document:

D.13.3.3_Installation_and_Administration_Guide_front_page

Middleware - Installation and Administration Guide

Middleware - KIARA - Installation and Administration Guide

Middleware - RPC over DDS - Installation and Administration Guide

Middleware - RPC over REST - Installation and Administration Guide

Middleware - Fast Buffers - Installation and Administration Guide

2D-UI - Installation and Administration Guide

3D-UI - XML3D - Installation and Administration Guide

3D-UI - WebTundra - Installation and Administration Guide

Synchronization - Installation and Administration Guide

Cloud Rendering - Installation and Administration Guide

Display As A Service - Installation and Administration Guide

GIS Data Provider - Installation and Administration Guide

POI Data Provider - Installation and Administration Guide

2D-3D Capture - Installation and Administration Guide

Augmented Reality - Installation and Administration Guide

Real Virtual Interaction - Installation and Administration Guide

Virtual Characters - Installation and Administration Guide

Interface Designer - Installation and Administration Guide

1.6 Typographical Conventions

Starting with October 2012 the FI-WARE project improved the quality and streamlined the submission process for deliverables, generated out of our wikis. The project is currently working on the migration of as many deliverables as possible towards the new system.

This document is rendered with semi-automatic scripts out of a MediaWiki system operated by the FI-WARE consortium.

1.6.1 Links within this document

The links within this document point towards the wiki where the content was rendered from. You can browse these links in order to find the "current" status of the particular content.

Due to technical reasons part of the links contained in the deliverables generated from wiki pages cannot be rendered to fully working links. This happens for instance when a wiki page references a section within the same wiki page (but there are other cases). In such scenarios we preserve a link for readability purposes but this points to an explanatory page, not the original target page.

In such cases where you find links that do not actually point to the original location, we encourage you to visit the source pages to get all the source information in its original form. Most of the links are however correct and this impacts a small fraction of those in our deliverables.

1.6.2 Figures

Figures are mainly inserted within the wiki as the following one:

```
[[Image:....|size|alignment|Caption]]
```

Only if the wiki-page uses this format, the related caption is applied on the printed document. As currently this format is not used consistently within the wiki, please understand that the rendered pages have different caption layouts and different caption formats in general. Due to technical reasons the caption can't be numbered automatically.

1.6.3 Sample software code

Sample API-calls may be inserted like the following one.

```
http://[SERVER_URL]?filter=name:Simth*&index=20&limit=10
```

1.7 Acknowledgements

The current document has been elaborated using a number of collaborative tools, with the participation of Working Package Leaders and Architects as well as those partners in their teams they have decided to involve. The following partners contributed to this deliverable: ADMINO, CYBER, UOULU,LUDOCRAFT,PLAYSIGN, ZHAW, EPROS, USAAR-CISPA, DFKI.

1.8 Keyword list

FI-WARE, PPP, Architecture Board, Steering Board, Roadmap, Reference Architecture, Generic Enabler, Open Specifications, I2ND, Cloud, IoT, Data/Context Management, Applications/Services Ecosystem, Delivery Framework , Security, Developers Community and Tools , ICT, Middleware, 3D User Interfaces.

1.9 Changes History

Release	Major changes description	Date	Editor
v1	Initial Version	2014-06-23	ZHAW

1.10 Table of Contents

2 Middleware - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

2.1 Introduction

In contrast to other GEs, the FI-WARE Middleware GE is not a standalone service running in the network, but a set of compile-/runtime tools and a communication library to be delivered with the application. Not all applications have the same requirements regarding dynamic code integration and protocol/network negotiation. For this reason we deliver, beside the integrated full dynamic runtime KIARA middleware framework also other components, which serve basic middleware functionalities.

The **KIARA middleware suite** Release 3.3 consists of the following components:

KIARA framework

The KIARA framework is a middleware that realizes a novel approach for connecting application code with a framework. It supports currently remote procedure calls (RPC) but support for publish/subscribe communication is planned as well.

Instead of forcing an application to use data types predefined by the middleware (as e.g. in Thrift, CORBA, etc.) applications describes its own native data types to the KIARA middleware. The middleware then generates the necessary code to access those data structures at run-time using an embedded compiler. Because due to the negotiation of the optimal protocol for the targeted server, KIARA can generate the optimal code to serialize the native data structures for the chosen protocol.

This approach allows to combine KIARA with arbitrary applications without rewriting major parts of the application itself or adding redundant copy operations between the native and middleware generated data structures, as required by other middleware.

RPC over DDS

RPC over DDS is based on the Data Distribution Service (DDS) specifications, an OMG Standard defining the API and Protocol for high performance publish-subscribe middleware. eProsima RPC over DDS is an Remote Procedure Call framework using DDS as the transport and is based on the ongoing OMG RPC for DDS standard. In this release of the middleware for FI-WARE (R3.3), we are providing the basic assets, DDS and eProsima RPC for DDS, and other modules that are building blocks of the KIARA Middleware suite.

RPC over REST

One of the main FI-WARE middleware requirements is to offer backwards compatibility with Web Services, specifically RESTful Web Services.

eProsima RPC over REST enable the creation and the invocation of RESTful Web Services through the common API also used in eProsima RPC over DDS, and the future eProsima RPC over TCP, both part of the FI-WARE middleware suite.

FastBuffers & Dynamic Fast Buffers

eProsima Fast Buffers is an open source serialization engine optimized for performance, beating alternatives such as Apache Thrift and Google Protocol Buffers in both Simple and Complex Structures.

It generates serialization code for your structured data from its definition in an Interface Description Language (IDL).

2.2 Installation and Administration Guides

For each of the components we provide separate Installation and Administration Guides

- Middleware - KIARA - Installation and Administration Guide
- Middleware - RPC over DDS - Installation and Administration Guide
- Middleware - RPC over REST - Installation and Administration Guide
- Middleware - Fast Buffers - Installation and Administration Guide

3 Middleware - KIARA - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

3.1 Introduction

This is an installation guide for DFKI KIARA SDK, a software distribution for the Advanced FI-WARE Middleware GE, code named KIARA.

In this first release of the middleware for FI-WARE (R3.2), we are providing following features:

- IDL with a syntax based on Thrift IDL
- Advanced API for declaration of user-defined C/C++ data types
- Generation, compilation, and execution of code for accessing C/C++ data types
- Generation, compilation, and execution of code for marshalling and unmarshalling of C/C++ data types

The system is currently not meant for production systems but mainly for evaluation and testing purposes. Not all of the KIARA features are available yet and not all features are optimized yet.

3.2 System Requirements

3.2.1 Hardware Requirements

Hardware requirements depend on the application to be developed.

Tested on X86 32 and 64bit systems.

3.2.2 Operating System

KIARA is developed to be easily portable, but it is tested and delivered on following platforms:

- Windows 7 and 8, 32 & 64 Bits (Windows XP might work, but was not tested)
- Linux 32 & 64 Bits, Fedora 19 and Ubuntu 12.04 (Also it should work on most distributions)

3.2.3 Software Requirements

- Windows:
  - MSVC 2008 or 2010
  - python 2.x for running the test suite
- Linux:
  - gcc or clang for compiling your applications
  - python 2.x for running the test suite

3.3 Software Installation and Configuration

The packages corresponding to the KIARA middleware can be found in the FI-WARE Files area. Look for the „MIWI-KIARA" entry. Download and extract appropriate KIARA SDK tar archive

named *kiara-sdk-** to the desired location. Run init.sh script on Linux and init.bat on Windows in order to setup KIARA SDK environment.

Examples provided with the SDK are in the `examples` directory along with the building instructions in the README file.

3.4 Sanity Check Procedures

3.4.1 End to End testing

After you executed init.sh or init.bat you can run `kiara-version` to check if KIARA SDK is properly installed.

3.4.2 List of Running Processes

KIARA Advanced Middleware itself do not install any kind of daemon or service. There are no running processes, but libraries to link to your applications.

3.4.3 Network interfaces Up & Open

The KIARA Middleware itself does not open or provide services, therefore has no open Ports or Interfaces. Applications using KIARA can open any ports or interfaces and firewalls have to be configured accordingly.

The provided TestServer is opening and listening by default on Port 8080.

3.4.4 Databases

N/A

3.5 Diagnosis Procedures

3.5.1 Resource availability

This middleware requires very few resources, any typical PC should be enough to run the regular examples.

3.5.2 Remote Service Access

N/A

3.5.3 Resource consumption

Depends on your application, it can be as low of 256 Kbytes of heap space and almost zero cpu use. The amount of RAM depends on your data types size and the different persistence options, please read the user manual for more information.

3.5.4 I/O flows

N/A

4 Middleware - RPC over DDS - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

4.1 Introduction

FI-WARE Middleware GE (code named KIARA) is a new middleware based on the Data Distribution Service(DDS) specifications, an OMG Standard defining the API and Protocol for high performance publish-subscribe middleware, and RPC over DDS, an Remote Procedure Call framework using DDS as the transport and based on the ongoing OMG RPC over DDS standard.

A quick DDS introduction is provided here

In contrast to other GEs, the FI-WARE Middleware GE is not a standalone service running in the network, but a set of compile-/runtime tools and a communication library to be delivered with the application.

In this release of the middleware for FI-WARE (R3.2), we are providing the basic assets, DDS and eProsima RPC over DDS, and other modules that are the building blocks of the KIARA Middleware Suite.

eProsima RPC over DDS was updated to include several planned features for KIARA, such as asynchronous calls and a high performance dispatching agent.

In this release RPC over DDS is fully compatible with other building block called RPC over REST. Using the same IDL and API you can call Remote Procedure Calls using DDS as the underlying transport, or call REST services. This feature is one of the key requirements of FI-WARE

4.2 System Requirements

4.2.1 Hardware Requirements

There are no special hardware requirements to install DDS or RPC over DDS.

For most apps, a regular PC is more than enought. Both DDS and RPC over DDS are developed for real-time systems, and the implementations are highly optimized. The amount of RAM required depends of the size of your data type. For small types, DDS can be setup to work with less than 1 MByte of Heap.

Disk Requirements: 300 MB of free space to install the products (libs, examples, documentation..)

4.2.2 Operating System

This product is developed to be easily portable, but it is tested officially at:

- Windows 32 & 64 Bits (Specifically Windows 7, but it should work on Windows XP or higher)
- Linux 32 & 64 Bits (Specifically Fedora 17, but it should work on most distributions)

4.2.3 Software Requirements

A basic implementation of DDS is required. Two different implementations of DDS can be selected:

1. RTI DDS 5.0 or later (Free, Open Community Source License)

2. OpenDDS 3.4.1 or later (Free, Open Source License)

We recommend RTI DDS because its performance and ease of use.

For the remote procedure calls framework use

- eProsima RPC over DDS 0.3.0 (free and Open Source License).

4.3 Installation

The user can choose among two different implementations of DDS, and install on top eProsima RPC over DDS.

4.3.1 DDS

4.3.1.1 *RTI DDS*

To download RTI DDS, visit the RTI DDS Download Page, and request for a RTI DDS Open Infrastructure Community License. When asked for the Infrastructure Community ID indicate "FIWARE_IC01"

If you want to perform just a quick evaluation, download a 30 days evaluation of RTI Connext complete suite here

RTI DDS documentation is published online here, specifically there is a step by step getting started and installation guide here

The installation is straight-forward: RTI DDS comes with automated installation packages for both windows and linux.

4.3.1.2 *OpenDDS*

To download openDDS, visit the OpenDDS download page

openDDS documentation is published online here and there is a guide to build openDDS here

4.3.2 RPC over DDS

To download eProsima RPC over DDS, visit the product page or the Fi-WARE Repository (login required)

The documentation is published online and the installation is straight-forward:

- Installation Manual (PDF)

If you need assistance, please contact eProsima Support

4.4 Sanity Check Procedures

4.4.1 End to End testing

To verify eProsima RPC over DDS is installed, just execute:

```
rpcddsgen -version
```

if installation is correct it should print:

RPCDDSGEN Version 0.2.0

eProsima RPC over DDS runs on top of the selected implementation of DDS, to test the correct installation of DDS, just invoke the IDL compiler in each case:

RTI DDS:

```
Rtiddsgen –version
```

OpenDDS:

```
Opendds_idl –v
```

In both cases if DDS is correctly installed you should get a message stating the product version

4.4.2 List of Running Processes

Both eProsima RPC over DDS and DDS itself do not install any kind of daemon or service. There are no running processes, but libraries to link to your applications.

4.4.3 Network interfaces Up & Open

To run any example or application using this middleware, please disable the firewall or setup the firewall ports according to these rules

4.4.4 Databases

The middleware does not install any databases.

4.5 Diagnosis Procedures

The middleware installers just copy a set of libraries and documentation to your hard drive and setup some environment variables. If the end to end testing procedures described above fail, just reinstall.

4.5.1 Resource Availability

This middleware requires very few resources, any typical PC should be enough to run the regular examples.

4.5.2 Remote Service Access

N/A

4.5.3 Resource consumption

Depends on your application, it can be as low of 256 Kbytes of heap space and almost zero cpu use. The amount of RAM depends on your data types size and the different persistence options, please read the user manual for more information.

4.5.4 I/O flows

This middleware uses RTPS protocol, a lightweight protocol adding little overhead. The amount of data transmitted it is in the same order of magnitude of the data size of the data types involved.

5 Middleware - RPC over REST - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

5.1 Middleware - RPC over REST - Installation and Administration

One of the main FI-WARE middleware requirements is to offer backwards compatibility with Web Services, specifically RESTful Web Services.

eProsima RPC over REST enable the creation and the invocation of RESTful Web Services through the common API used in eProsima RPC over DDS, and the future eProsima RPC over TCP, both part of the FI-WARE middleware suite.

This product ease the integration or migration of existing web services with the FI-WARE middleware.

eProsima RPC over REST supports WADL (Web Application Definition Language) as the IDL to define RESTful Web Services.

5.2 System Requirements

5.2.1 Hardware Requirements

There are no special hardware requirements.

For most apps, a regular PC is more than enought. RPC over REST is developed for real-time systems, and the implementation is highly optimized.

The amount of RAM required depends of the size of your data type.

Disk Requirements: 100 MB of free space to install the products (libs, examples, documentation..)

5.2.2 Operating System

This product is developed to be easily portable, but it is tested officially at:

- Windows 32 & 64 Bits (Specifically Windows 7, but it should work on Windows XP or higher)
- Linux 32 & 64 Bits (Specifically Fedora 17, but it should work on most distributions)

5.2.3 Software Requirements

Supported Compilers & dev Environments?

5.3 Software Installation and Configuration

To download eProsima RPC over REST, visit the Fi-WARE Repository (login required)

The documentation is published online and the installation is straight-forward:

- Installation Manual (PDF)

If you need assistance, please contact eProsima Support

5.4 Sanity check procedures

5.4.1 End to End testing

To verify eProsima RPC over REST is installed, just execute:

```
rpcddsgen –version
```

if installation is correct it should print:

rpcddsgen Version 0.2.0

5.4.2 List of Running Processes

eProsima RPC over REST does not install any kind of daemon or service. There are no running processes, but libraries to link to your applications.

5.4.3 Network interfaces Up & Open

You should open the TCP ports you use to serve the RESTful services you create. The port is a parameter in the HttpTransport object. See the user manual for more information.

5.4.4 Databases

This product does not install any databases.

5.5 Diagnosis Procedures

The product installers just copy a set of libraries and documentation to your hard drive and setup some environment variables. If the end to end testing procedures described above fail, just reinstall.

5.5.1 Resource availability

This middleware requires very few resources, any typical PC should be enough to run the regular examples.

5.5.2 Remote Service Access

N/A

5.5.3 Resource consumption

Depends on your application, it can be as low of 256 Kbytes of heap space and almost zero cpu use. The amount of RAM depends on your data types size and the different persistence options, please read the user manual for more information.

5.5.4 I/O flows

This middleware uses HTTP protocol. The amount of data transmitted it is in the same order of magnitude of the data size of the data types involved.

6 Middleware - Fast Buffers - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

6.1 Middleware - Fast Buffers - Installation and Administration

eProsima Fast Buffers is an open source serialization engine optimized for performance, beating alternatives such as Apache Thrift and Google Protocol Buffers in both Simple and Complex Structures.

eProsima Fast Buffers generates serialization code for your structured data from its definition in an Interface Description Language (IDL).

eProsima Fast Buffers is the high performance serialization mechanism of the KIARA middleware suite

6.2 System Requirements

6.2.1 Hardware Requirements

There are no special hardware requirements.

For most apps, a regular PC is more than enought. Fast Buffers is developed for real-time systems, and the implementation is highly optimized.

The amount of RAM required depends of the size of your data type.

Disk Requirements: 100 MB of free space to install the products (libs, examples, documentation..)

6.2.2 Operating System

This product is developed to be easily portable, but it is tested officially at:

- Windows 32 & 64 Bits (Specifically Windows 7, but it should work on Windows XP or higher)
- Linux 32 & 64 Bits (Specifically Fedora 17, but it should work on most distributions)

6.2.3 Software Requirements

FastBuffers is officially supported/tested for the following compilers:

- Linux: gcc 4.8
- Windows: Visual Studio 2010

But it should work in the following compilers

- Linux: gcc 4.4.5 or higher
- Windows: Visual Studio 2010 higher

6.3 Software Installation and Configuration

To download eProsima Fast Buffers, visit the product page or the Fi-WARE Repository (login required)

The documentation is published online and the installation is straight-forward:

- Installation Manual (PDF)

If you need assistance, please contact eProsima Support

6.4 Sanity check procedures

6.4.1 End to End testing

To verify eProsima Fast Buffers is installed, just execute:

```
fastbuffers -version
```

if installation is correct it should print:

FastBuffers Version 0.2.0

6.4.2 List of Running Processes

eProsima Fast Buffers do not install any kind of daemon or service. There are no running processes, but libraries to link to your applications.

6.4.3 Network interfaces Up & Open

No required.

6.4.4 Databases

The serialization library does not install any databases.

6.5 Diagnosis Procedures

The product installers just copy a set of libraries and documentation to your hard drive and setup some environment variables. If the end to end testing procedures described above fail, just reinstall.

6.5.1 Resource availability

This product requires very few resources, any typical PC should be enough to run the regular examples.

6.5.2 Remote Service Access

N/A

6.5.3 Resource consumption

Depends on your application. To serialize structure data you need a buffer big enough for this data, therefore the resource consumption is equivalent to your data size.

6.5.4 I/O flows

N/A

7 2D-UI - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

7.1 Introduction

This document describes using the Input API and Web Component features of the 2D-UI GE.

7.2 System Requirements

The 2D-UI components run in common web browsers and are independent of hardware and operating system.
A list of detailed browser compatibility for the Polymer Web Components can be found here http://www.polymer-project.org/resources/compatibility.html.
The Input API requires the JQuery core library Version 1.9 or higher. All other 3rd-party libraries are delivered with the GEi.

7.2.1 Hardware Requirements

N/A

7.2.2 Operating System Support

N/A

7.2.3 Software Requirements

Common web browser.

7.3 Software Installation and Configuration

All 2D-UI components are collections of JavaScript libraries which have to be included in your web-project. Most of the components and example run locally in the web browser. Some components have to be installed and delivered from a web-server.

7.3.1 Input API

To install the Input API you can clone the current version from the Git repository on GitHub or download release forge.fi-ware.org. The **`lib`** directory contains the 3rd-party JavaScript libraries which are required for the InputAPI (jQuery plugins, Signals and Classy). The InputAPI JavaScript source files can be found in the **`src`** directory. The **`test`** folder finally contains an example implementation **`input.html`** which shows how the InputAPI has to be initialized and used. If you use the structure of the cloned git repository keep the file in the **`test`** folder and leave script paths as they are. If you use a different structure in your web project or create your own programs you have to change the references accordingly. Remember that all plugin file references must be after the main InputAPI.js reference which contains the definition of InputPlugin.
Details how to use the InputAPI can be found in the 2D-UI - User and Programmers Guide.

7.3.2 Input Abstraction

Input Abstraction is part of the Input API implementation. To install follow the installation guide of InputAPI. The **`test`** folder contains an example implementation **`inputState.html`** of Input

Abstraction which shows how the abstraction is initialized and used. Details how to use the Input Abstraction can be found in the 2D-UI - User and Programmers Guide.

7.3.3 Polymer Web Component

To install the Polymer Web Component you can download ZIP archive from forge.fi-ware.eu, extract the downloaded archive and add the files to your web project on your local file system or on a web server. The polymer browser components are packaged in the `browser_component` directory. The `chat` directory contains components to implement an example chat client implemented using the Polymer library version 0.1.1. In the root directory of the ZIP archive you'll also find three examples how to implement a chat web application.

index.html

Shows how to use the standard way how to instantiate the chat-client referencing the library in the `browser_components` folder and the chat-components in the `chat` directory.

vulcanized.html

The vulcanized version works similar to the standard version, but instead of referencing individual libraries and components int includes a *vulcanized* import file, which concatenates a set of Polymer components in a single file

dynamic.html

One of the biggest issues with Polymer is the lack of ability to load web components dynamically. However all needed functions are already there but the Polymer Project itself has not yet created a convinient way of using them. This is discussed topic inside the project and will be part of the project in the future. In this example we show a way to do it currently.

The difference to the above examples is that this has to be installed on a web server. The example sets default paths to http://localhost. If you install it on your web server you have to remember to set the path's in `dynamic.html` and `chat/polymer-chat.html` accordingly.

7.4 Sanity check procedures

7.4.1 End to End testing

7.4.1.1 *InputAPI*

- Open the Input.html file from the cloned repository `test` folder with your desired web browser. If InputAPI is working correctly you will see a log on the web page which shows every key press/release and all mouse events. Key events in the displayed log start with `Key` and mouse events start with `Mouse`.
- To test touch events place the file on a web server and use your mobile device web browser to open the file and test touch events. All touch event logs start with `Touch`.
- To test gamepad, connect your XBox or Playstation USB controller to your computer. NOTE: At this stage only the latest Chrome browser is supported. If gamepad is supported by your browser it is told in the display log. Once the gamepad is connected and the browser supports it, the display log will start to show gamepad button and axis states on every browser animation frame.

7.4.1.2 *InputAbstraction*

- Open the InputState.html file from the cloned repository `test` folder with your desired web browser.

- If InputState is working correctly you will see a web page where you can create and update an InputState.
- The page is divided to 2 sections with a grey line. On the left is the InputState creation and on the right is the log display.
- Once the InputState is saved and its conditions are true the log display shows that the InputState with your given name is fired.

7.4.1.3 *Polymer Web Component*

To test the example polymer-chat web component open one or all of these included example files in the root of the ZIP archive:

- index.html
- vulcanized.html
- dynamic.html (must be run from a web server, see above)

Once the file is open in the browser and the web component is displayed correctly, the user will see a chat widget on the bottom left corner of the opened page.

7.4.2 List of Running Processes

N/A

7.4.3 Network interfaces Up & Open

N/A

7.4.4 Databases

N/A

7.5 Diagnosis Procedures

N/A

7.5.1 Resource availability

N/A

7.5.2 Remote Service Access

N/A

7.5.3 Resource consumption

N/A

7.5.4 I/O flows

N/A

8 3D-UI - XML3D - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

8.1 Introduction

The purpose of this documentation is to provide information how to use XML3D in 3D Web-application and how to enable webcl for xflow.

8.2 System Requirements

There are no special requirements to the system for XML3D. A standard desktop PC, Laptop or even mobile device is sufficient.

8.2.1 Hardware Requirements

The following requirements must be fulfilled for hardware accelerated Xflow:

- **CPU**:
  - x86 compatible with SSE3 extensions or later
- **GPU**:
  - Requires the appropriate OpenCL drivers. You can find a list of available drivers for different platforms here: http://www.khronos.org/conformance/adopters/conformant-products#opencl

8.2.2 Operating System Support

xml3d.js is supported in any browser that supports WebGL, that is

- **Windows** : Firefox, Chrome
- **Linux** : Firefox, Chrome
- **Android**: Chrome

For hardware accelerated Xflow, we currently support the following platforms:

- **Windows**: It does not work on 64-bit Firefox, though, but that is not even available through regular distribution channels.
- **Ubuntu**: Works on 32-bit Firefox with 32-bit OpenCL drivers, and as of Firefox 19, it also works with 64-bit Firefox and OpenCL.
- **Mac OS X**

8.2.3 Software Requirements

- Web browser that supports OpenGL to display the XML3D scene
- Latest version of Nokia's WebCL Extension for FireFox
- OpenCL SDK

8.3 Software Installation and Configuration

XML3D does not need to be installed on the system, but is available as polyfill implementation xml3d.js and can be linked to the webpage by the following link:

```
   <script type='text/javascript'
src='http://www.xml3d.org/xml3d/script/xml3d.js'>
```

XML3D provides also a camera controller that allows basic mouse- and keyboard interaction:

```
   <script type='text/javascript'
src='http://www.xml3d.org/xml3d/script/tools/camera.js'>
```

8.3.1 Installing WebCL for hardware accelerated Xflow

8.3.1.1 *Installing instructions for Ubuntu 13.04 + firefox 25*

- install NVIDIA's OpenCL SDK (https://developer.nvidia.com/cuda-downloads)
- install Intel's OpenCL SDK(http://software.intel.com/en-us/vcsource/tools/opencl-sdk-xe)

Here are steps used with Ubuntu 13.04 and the new multi-package distribution Intel® SDK for OpenCL* Applications XE 2013:

- Download the 64-bit distribution from Intel: intel_sdk_for_ocl_applications_2013_xe_sdk_3.0.67279_x64.tgz*
- Download the Intel public key *Intel-E901-172E-EF96-900F-B8E1-4184-D7BE-0E73-F789-186F.pub*:

```
  $ sudo apt-get install -y rpm alien libnuma1
  $ sudo rpm --import Intel-E901-172E-EF96-900F-B8E1-4184-D7BE-0E73-
F789-186F.pub
  $ tar -xvf
intel_sdk_for_ocl_applications_2013_xe_sdk_3.0.67279_x64.tgz
  $ cd intel_sdk_for_ocl_applications_2013_xe_sdk_3.0.67279_x64/
  $ fakeroot alien --to-deb opencl-1.2-base-3.0.67279-1.x86_64.rpm
  $ fakeroot alien --to-deb opencl-1.2-intel-cpu-3.0.67279-1.x86_64.rpm
  $ sudo dpkg -i opencl-1.2-base_3.0.67279-2_amd64.deb
  $ sudo dpkg -i opencl-1.2-intel-cpu_3.0.67279-2_amd64.deb
```

The above installs the library files and intallable client driver registration in /opt/intel/opencl-1.2-3.0.67279. Two more steps are needed to run an OpenCL program.

- Add library to search path:

```
  $ sudo touch /etc/ld.so.conf.d/intelOpenCL.conf
```

- Edit this file, add the line:

```
  /opt/intel/opencl-1.2-3.0.67279/lib64
```

- Link to the intel icd file in the expected location:

```
  $ sudo ln /opt/intel/opencl-1.2-3.0.67279/etc/intel64.icd
/etc/OpenCL/vendors/intel64.icd
  $ sudo ldconfig
```

- If doing developmnent, install the developer headers and tools:

```
  $ fakeroot alien --to-deb opencl-1.2-devel-3.0.67279-1.x86_64.rpm
  $ fakeroot alien --to-deb opencl-1.2-intel-devel-3.0.67279-
1.x86_64.rpm
  $ sudo dpkg -i opencl-1.2-devel_3.0.67279-2_amd64.deb
  $ sudo dpkg -i opencl-1.2-intel-devel_3.0.67279-2_amd64.deb
```

- It is worth noting the include path for headers is: /opt/intel/opencl-1.2-3.0.67279/include
- The linking path for libraries is: /opt/intel/opencl-1.2-3.0.67279/lib64
- The developer tool binaries are installed in: /opt/intel/opencl-1.2-3.0.67279/bin
- install Nokia's extension (http://webcl.nokiaresearch.com/extensions/firefox/multiplatform/latest/webcl-1.0.xpi)

8.3.1.2 *Installing instructions for Win8 + firefox 25*

```
* install Intel's OpenCL SDK (http://software.intel.com/en-us/vcsource/tools/opencl-sdk)
* install NVIDIA's OpenCL SDK (https://developer.nvidia.com/cuda-downloads)
* install AMD's OpenCL SDK (http://developer.amd.com/tools-and-sdks/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/downloads/)
* install Nokia's extension (http://webcl.nokiaresearch.com/extensions/firefox/multiplatform/latest/webcl-1.0.xpi)
```

8.4 Sanity check procedures

- Follow the user guide tutorial in the User Guide
  - Your browser should display a teapot model.
  - If no teapot model is displayed, perform the Sanity Checks above
  - If your browser displays the error message "*Your Browser does not support WebGL*", switch to a browser that supports WebGL
- check that you have WebCL&WebGL enabled (http://webcl.nokiaresearch.com/)

8.4.1 End to End testing

N/A

8.4.2 List of Running Processes

N/A

8.4.3 Network interfaces Up & Open

N/A

8.4.4 Databases

N/A

8.5 Diagnosis Procedures

8.5.1 Resource availability

8.5.1.1 *XML3D with software Xflow*

- Check if the XML3D script is loaded correctly:
  - Open your Web-page that uses XML3D
  - Open the browser debug console
  - Type "*XML3D.version*"
  - The console should show the current version number of xml3d.js . If it prints an error ("*Namespace XML3D is not defined*"), the script was not loaded correctly
- Check if Camera script is loaded correctly:
  - Open your Web-page that uses camera.js camera controller and the browser debug console
  - Type "*XML3D.Camera*"
  - If the script was not loaded correctly, the console will prompt *undefined* as result.
- If objects in your page are not rendered, or not rendered correctly, check the correctness of the XML3D DOM tree:
  - All XML3D nodes (*<group>, <transform>, <view>*, etc.) must be child nodes of the '<xml3d>' node
  - There must be only one '<xml3d>' element per page
  - Check if id values of referenced elements are correct, and references are given with the '#'-operator. That is, a transform with id *myTransform* must be referenced via *#myTransform*
  - Check if external references are loaded correctly. If the URL of external documents is incorrect or the documents are no longer available, the console will print a 404 error
- If objects that contain Xflow animations are not animated or displayed incorrect, check if the XML3D operators are applied correctly:
  - Check if *<data>*-elements that work as source for the computation are resolved correctly. If not, console will print an error (*"Could not find data element for ID"*)
  - Check if there are any inconsistencies in your Xflow tree. Check if introduced variables are used by their correct name. If not, console will print an error that variable names are not defined in the tree

8.5.1.2 *Hardware accelerated Xflow*

- If sanity checks for hardware acceleration fail, check if OpenCL is working correctly.

8.5.2 Remote Service Access

N/A

8.5.3 Resource consumption

N/A

8.5.4 I/O flows

N/A

9 3D-UI - WebTundra - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

9.1 Introduction

The purpose of this documentation is to provide information how to install WebTundra, a browser client-side library for making 3D Web applications.

9.2 System Requirements

For graphical use WebTundra requires a computer capable of running a modern web browser. Any desktop or laptop computer made since approximately 2007 should do. Also mobile devices such as smart phones and tablets work, possibly with limitations.

For networked connectivity WebSockets support is required in the browser.

Good quality rendering of detailed 3D scenes and effects requires WebGL support. In the following subsections we list the current status of WebGL availability on various platforms.

9.2.1 Hardware Requirements

Without WebGL: Nothing special, any Web capable device works.

For WebGL it is required to have graphics driver for which WebGL is enabled in the browser used. These are live lists that the browser vendors can change at any time, current status is tracked at https://www.khronos.org/webgl/wiki/BlacklistsAndWhitelists

9.2.2 Operating System Support

Any operating system capable of running a Web browser works. This includes Windows, Mac, Linux, Android and iOS with restrictions. The status with game consoles (xbox, play station) is to be investigated.

9.2.3 Software Requirements

Web browser with support for the chosen renderer on the host operating system is required. Three.js has renderers for WebGL, Canvas2D, CSS and SVG backends.

Browser and operating support for WebGL. For Firefox and Chrome we refer to current stable version as they typically auto-update nowadays and use of current versions is recommended also for security reasons.

Windows: Firefox, Chrome, Internet Explorer 11, Opera

Mac: Firefox, Chrome, Safari only by manually enabling WebGL in the Developer menu (which is hidden by default)

Linux: Firefox, Chrome

Android: Firefox, Chrome

9.3 Installation

9.3.1 Networked operation for multi-user applications

WebTundra has 3D-UI (using Three.js in this implementation) integrated with the Synchronization GE for a networked realXtend web client to be connected to a Tundra server.

To install WebTundra clone the git repository at https://github.com/realXtend/WebTundra

For the networked operation you need to run the Tundra server, as instructed in Synchronization - Installation and Administration Guide#Server installation

To host your own content you also need a web server to host the web client html, javascript and 3d scene asset files. Any web server works, for example Apache or IIS. In the instructions here we use Python's bundled simple server which is suitable for local testing and is available by default on Mac and Linux and by installing Python on Windows (the example commands are for Python 2.x).

9.3.2 Standalone mode for single user applications

It is also possible to run WebTundra as a standalone web application, without a Tundra server. In this case the scene can be authored with declarative XML3D or by imperative Javascript programming (for example to load a scene by calling the scene loading function). This is however, at least currently, of limited usefulness: the benefit of the Entity-System provided by Tundra and WebTundra is largely in the integration with the networking. Non-networked single user applications can well be developed using the underlying Three.js WebGL engine directly, or alternatively using the XML3D.js implementation for full declarative XML3D support.

9.3.3 Preparing own scenes

To show 3d assets from external files the client currently expects them in the Three.js JSON format in the URL to where the Mesh components on the server point at. Exporters and instructions for Blender, Max and Maya are available from https://github.com/mrdoob/three.js/tree/master/utils/exporters/ . Detailed instructions for exporting and using the models is available in the User Guide.

To test connecting to your Tundra scene / application without converting assets there is a fallback option 'useCubes' to display all objects in the scene as cubes instead -- that works without any asset file dependencies. To test this, change the default value to "useCubes = true" in src/view/ThreeView.js.

For more information on application development using the WebTundra library please refer to the associated Programmer's Guide.

9.4 Sanity check procedures

9.4.1 End to End testing

An example scene is included in the repository for easy testing. We also host it on-line on the Web to enable most straightforward testing.

Test with provided web hosting for scene data To see the test scene in the network synchronized 3d client, first start the server process with this command in the server repository.

```
 ./Tundra --file
http://playsign.tklapp.com:8000/WebTundra/examples/Physics2/scene.txml
--server
```

This opens the server with GUI enabled so you see a 3d view (with Ogre3d) also for the server state. This is required for interactivity in this test, as it is now only implemented on the server side.

To open the client from web hosted version, browse to this address on the same machine:

```
http://playsign.tklapp.com:8000/WebTundra/examples/Physics2/
```

This loads the WebTundra client code to your browser and connects to the server in localhost. You should see the scene both in the native Tundra window and in the WebTundra web browser window.

You can then click the big sphere at the top to make it fall on the blocks. The clicking works both in the server GUI window and in the Web client view. The resulting falling of the blocks is synchronized in realtime to the WebTundra browser view as well. You can also grab objects in the server view by first pressing shift-e to enable editing mode and using the mouse to select and move objects -- again the movements form the server are synchronized to the client.

Test with self-hosted web client and scene assets

Start the server with your local data, in the server directory:

```
./Tundra --file /path/to/WebTundra/scenes/physics2/scene.txml --server
```

If you are not already running a web server you can start one:

```
cd /path/to/WebTundra
python -m SimpleHTTPServer
```

Then you can open the client locally by navigating to:

```
http://localhost:8000/examples/Physics2/
```

Test local standalone application with XML3D

For non-networked single user application mode no servers are needed: they are like any web page, so you can simply open a HTML file to see a 3d scene. This works by WebTundra supporting reading scenes from a XML3D file. Examples are provided in the GIT repository: examples/xml3d/ . For the meshes and materials the Three.js formats are used also in this case.

9.4.2 List of Running Processes

When running against a Tundra server, a process called 'Tundra' must be in the list.

For serving the files over http, your web server should show in the process list. For example 'apache2', or 'python' if you are using the SimpleHTTPServer when testing / developing as instructed above.

9.4.3 Network interfaces Up & Open

Tundra server uses the port 2345 by default. WebTundra connects to that by default, with WebSockets for the realtime scene synchronization.

For HTML and 3d asset file transfers HTTP is used normally, defaulting to port 80 but any port works as the asset fetch URLs are relative to what is used to open the client HTML.

9.4.4 Databases

N/A

9.5 Diagnosis Procedures

The basic helpful diagnostic tool for Web applications are the developer tools of Web browsers. If there are problems with the applications first step is to check the Javascript console for errors.

9.5.1 Resource availability

If any resources are not available they will show as errors in the browser console. This includes:

1. Necessary graphics context (webgl, canvas2d)

2. File access problems (3d meshes, textures)

3. Missing file format support, for example DXT texture compression is not available in all environments

4. Graphics driver fails to offer 3D, or is known bad ("blacklisted") by browser. See browser graphics diagnostics. In Chrome, that's chrome://gpu/

9.5.2 Remote Service Access

Connection error with WebSockets to a Tundra server is reported as an error in the console.

The diagnosis procedure is to ensure that the server is up and reachable in the address used with the WebSocketServer plugin and server mode enabled.

9.5.3 Resource consumption

Chrome task manager is a good tool for analysing memory (both central and GPU) and other resource use. It is normal for 3d scenes to use hundreds of megabytes of both central and graphics memory. With all but most simple scenes it is normal for rendering with WebGL to use much or all available processing power of a single core & the GPU. For I/O there typically is a heavy load in the beginning when all the 3d geometry and texture files are downloaded (can be tens of megabytes) but in simple cases no additional heavy network I/O later. There can be substantial WebSocket traffic during the session if there is a lot of updates in the scene (for example moving objects). In large or dynamic scenes where content is downloaded on-demand there can be substantial downloads (megabytes) almost constantly during the use.

9.5.4 I/O flows

Browser developer tools can display the I/O flows. On the networking level HTTP is used for downloads and WebSockets for realtime synchronization of the scene state. It is typical to have a lot of downloads at the start but none or less later on. The WebSocket connection typically stays open during the whole session.

10 Synchronization - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

10.1 Introduction

This document describes installing and running the server and client parts of the Synchronization GE.

10.2 System Requirements

The server part of the Synchronization GE is based on the realXtend Tundra SDK, therefore its hardware requirements are the same as of the Tundra SDK.

10.2.1 Hardware Requirements

Server requirements

- Minimum of 1GB memory
- About 40GB free hard disk space when building Tundra, 1GB free when running
- Intel-compatible CPU with SSE instruction support, minimum processor speed 1.66 GHz
- If running graphics, a GPU with minimum of 256MB memory

10.2.2 Operating System Support

Server requirements

- Windows XP or newer
- Mac OS X 10.6 or newer
- Ubuntu Linux 12.04 or newer. Other distributions may work but are not guaranteed.

The client code is operating system independent, as it's JavaScript code running in a browser.

10.2.3 Software Requirements

The client part of the Synchronization GE requires a browser that has WebSocket support, such as

- Internet Explorer 10.0+
- Firefox 6.0+
- Chrome 14.0+
- Safari 6.0+
- Opera 12.1+
- iOS Safari 6.0+
- Android Browser 4.4+
- Opera Mobile 12.1+
- Chrome for android 31.0+
- Firefox for Android 25.0+
- Internet Explorer Mobile 10.0+

For serving the synchronization client HTML and JavaScript code, an HTTP server such as Apache can be used. For testing use the browser can also be pointed to local files.

10.3 Software Installation and Configuration

10.3.1 Server installation

The server part is based on version 2.5.3RC of realXtend Tundra SDK. There are two options, to download a prebuilt binary package, or build from source.

10.3.1.1 *Prebuilt binaries (Linux)*

32-bit and 64-bit packages for Debian are provided. See these download links:

- https://forge.fi-ware.org/frs/download.php/1141/MIWI-Synchronization-Server-2.5.3RC-ubuntu-12.04-i386.deb
- https://forge.fi-ware.org/frs/download.php/1140/MIWI-Synchronization-Server-2.5.3RC-ubuntu-12.04-amd64.deb

To install a Tundra package and its dependencies, use these commands. You need to have root privileges and an active Internet connection for downloading dependencies.

- dpkg -i <package file name>
- apt-get -f install

After successful installation, the Tundra binary and examples scenes are placed in the /opt/realxtend-tundra directory.

10.3.1.2 *Prebuilt binaries (Windows)*

32-bit and 64-bit installers are provided. These require an active Internet connection for downloading Microsoft redistributables.

- https://forge.fi-ware.org/frs/download.php/1143/MIWI-Synchronization-Server-Tundra-2.5.3RC-Windows-x86.exe
- https://forge.fi-ware.org/frs/download.php/1142/MIWI-Synchronization-Server-Tundra-2.5.3RC-Windows-x64.exe

After successful installation, Tundra can be run from the installation directory using the command prompt. This is the recommended way of running instead of Start Menu shortcuts, to be able to specify the command line parameters. On Windows there are Tundra.exe and TundraConsole.exe executables, where TundraConsole will display a console window and is therefore recommended for server usage.

10.3.1.3 *Building from source*

Clone the Tundra git repository at https://github.com/realXtend/tundra.git, then clone the TundraAddons repository from https://github.com/realXtend/TundraAddons.git into the subdirectory src/TundraAddons of the Tundra clone. Then follow the README.md file in the root directory of the Tundra clone for build instructions. More detailed instructions are also found from the files (use Doxygen to generate html format documentation)

- https://github.com/realXtend/tundra/blob/tundra2/doc/dox/BuildOnMac.dox
- https://github.com/realXtend/tundra/blob/tundra2/doc/dox/BuildOnUbuntu.dox
- https://github.com/realXtend/tundra/blob/tundra2/doc/dox/BuildOnWindows.dox

After a successful build of Tundra, it can be run from the bin subdirectory. On Windows there are Tundra.exe and TundraConsole.exe executables, where TundraConsole will display a console

window and is therefore recommended for server usage. On Linux / OS X there is just an executable called Tundra.

10.3.1.4 *Testing the installation (all methods)*

To load an example scene with server mode (both the Real-time Synchronization & SceneAPI services) enabled for testing, run the following command. On Windows, replace Tundra with TundraConsole:

```
Tundra --config tundra.json --config tundra-addons.json --file
scenes/Avatar/scene.txml --server --headless --httpport 2346
```

If running from prebuilt binaries, the extra configuration file tundra-addons.json is already enabled. Therefore the --config command line options can be omitted.

```
Tundra --file scenes/Avatar/scene.txml --server --headless --httpport
2346
```

10.3.2 Client installation

The client part of the Synchronization GE is supplied as part of the WebTundra JavaScript libraries. There are two options for getting it:

- Download a packaged release from https://forge.fi-ware.org/frs/download.php/1139/MIWI-Synchronization-Client-WebTundra-3.3.3.zip
- Clone the git repository at https://github.com/realXtend/WebTundra

To test connecting to a localhost synchronization server, open the file index.html in your browser from the source code clone or extracted package.

10.4 Sanity check procedures

10.4.1 End to End testing

For both the server and the client, it is recommended to have at least 1 GB free RAM and 1 GB free hard disk space, a broadband network connection, and to ensure that the system is not unnecessarily loaded with other processes.

10.4.1.1 *Real-time Synchronization*

When the Tundra server is running using the command line above, and the client HTML & JavaScript is locally available, you should be able to test opening index.html from the client code clone's root directory using a web browser, at which point the JavaScript synchronization code library should connect to the Tundra server. You should see the following in Tundra server console:

```
[WebSocketServer]: Connection ID 1 login successful
```

The connection ID may not be exactly the same.

Also, when inspecting the index.html page using eg. Firebug and opening the console, you should see among the first messages

```
"Sending login message"
```

Seeing these prints confirms that a WebSocket connection between the Tundra server and the JavaScript client code is functioning, and synchronization can be used.

10.4.1.2 *SceneAPI REST service*

When the Tundra server is running using the command line above, point a web browser to the following local address to confirm that the SceneAPI is operational:

```
http://localhost:2346/entities
```

This should display a dump of the scene loaded on the Tundra server as XML data.

10.4.2 List of Running Processes

Server: Tundra or TundraConsole

10.4.3 Network interfaces Up & Open

Port 2345, both TCP & UDP. This is the default Tundra port for serving scenes using the real-time synchronization and can be changed with the --port <portnumber> startup parameter.

Port 2346, TCP (HTTP). SceneAPI REST service. This was specified on the Tundra server command line (parameter --httpport 2346). There is no default value; if this parameter is not specified the SceneAPI service will not start up.

10.4.4 Databases

N/A

10.5 Diagnosis Procedures

If connection fails between the server and the client, verify the following:

- that the server was started with the --server command line parameter
- that the WebSocketServerModule is loaded on the server; the row "Loading plugin WebSocketServerModule" should be printed to the server console during server startup, with no error messages following it
- that firewall is not blocking port 2345
- if the client and server are not on the same machine, or the server port is not the default 2345, modify the following row in the file Test.js in the client JavaScript code. Replace localhost and 2345 with the actual address and port of the server

```
client.connect("localhost", 2345, loginData);
```

If connection to the SceneAPI service fails, verify the following:

- that the http server port was specified on the server command line parameter, eg. --httpport 2346
- that the HttpServerModule is loaded on the server; the row "Loading plugin HttpServerModule" should be printed to the server console during server startup, with no error messages following it. The loading of this module is controlled by the configuration file tundra-addons.json, which is enabled by the parameter --config tundra-addons.json on the server command line. This should have been copied to the Tundra executable directory as part of the TundraAddons build process
- that firewall is not blocking port 2346

10.5.1 Resource availability

The server memory consumption should be around 50 MB when running a simple scene without rendering. With rendering enabled, it may be between 100 MB and 200 MB. A typical desktop web browser typically uses around 100 MB of memory when showing a Synchronization client application. CPU usage depends on the amount of activity in the networked scene, but should be when there is nothing happening.

10.5.2 Remote Service Access

N/A

10.5.3 Resource consumption

As resource consumption is dependent on the scene complexity, the scripts that are running in it, the amount of client connections to the server and their activity, it is hard to give directions to what kind of eg. a memory consumption on the server would be considered abnormal. One guideline would be to note the amount of memory in use once a server has started up and loaded the scene: if memory use for example rapidly doubles from that figure and continues to grow unbounded, then it's likely that an application script is misbehaving and leaking memory.

In a demanding application or scene that uses physics and heavy scripting, it is not uncommon to see high CPU use figures by the server (up to full utilization of one processor core) with high numbers of active clients. For an example, consider a high number (50-100) of clients connected to a scene running an "avatar application": each client gets an avatar to move around, which is physically simulated and uses scripts for movement, collision response etc. Rectifying high CPU utilization in such case will require either using more powerful hardware, simplifying the application (for example not using actual physics simulation, or simplifying the scripts) or limiting the amount of users that can simultaneously connect.

Note that this is considering the processing on the Tundra server process as a whole. The network synchronization itself is fairly lightweight in its processing requirements.

10.5.4 I/O flows

The Synchronization GE will by default use TCP (WebSocket) traffic on the port 2345. Additionally native (Tundra) clients will use UDP traffic on the same port. The SceneAPI REST service will use TCP (HTTP) traffic on the chosen port. The amount of traffic on the server will depend the scene being served and the amount of connected clients; typical would be in the order of tens to hundred kilobytes per second.

11 Cloud Rendering - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

11.1 Introduction

This document describes the installation and administration of the reference implementation provided by the Cloud Rendering GE.

The Cloud Rendering GEi is split into three distinct components. Each of them will be covered, when appropriate, in the subsections of this page.

- **WebService** is a web service that does signaling between the renderer(s) and the web client(s). Facilitating the all important function of connecting two peers that want to communicate with WebRTC. Additionally application level messaging can be sent through it.
- **Renderer** is the application that delivers 3D rendering results to web clients via WebRTC video. In our case this is a realXtend Tundra based 3D virtual world client.
- **WebClient** is the application running in the end users web browsers that wants to receive 3D rendering from a renderer.

11.2 System Requirements

11.2.1 Hardware Requirements

WebService

- Any hardware that is able to run a nodejs web server.

Renderer

- Any hardware that is able to run realXtend Tundra.
    - Minimum 256 MB of graphics card memory
    - Minimum Processor 1.66 Ghz
    - Minimum 1 GB of RAM

WebClient

- Can be ran on web browsers that, at minimum, support WebRTC peer connections and the MediaStream API.
    - General support table http://iswebrtcreadyyet.com
    - Peer connection http://caniuse.com/rtcpeerconnection
    - MediaStream API http://caniuse.com/#feat=stream

11.2.2 Operating System Support

WebService

- Windows
- Mac OS X
- Linux (build supported but ready built binary not part of the release)

Renderer

- Windows: Windows Vista or newer
- Mac: OS X 10.6 (Snow Leopard) or newer and Intel CPU

WebClient

- WebClient requires a web browser that supports the following
  - RTCPeerConnection (WebRTC media streaming)
  - WebSocket (WebSocket two way communication)

11.2.3 Software Requirements

WebService

- node.js
- grunt

Renderer

- Windows: Latest DirectX 9 or OpenGL, latest C++ 2008 runtime and OpenAL sound library. All of this is bundled and will be installed with the released .msi installer.
- Mac: The release .pkg binary will include everything you need.

WebClient

- Recommended browsers are latest Google Chrome or Mozilla Firefox.

11.3 Software Installation and Configuration

11.3.1 WebService

You can find the WebService source code from https://github.com/Adminotech/fiware-cloud-rendering-service, you should start by reading the README.md. Here are the first steps to get you started.

Alternatively you can download the latest release as a zip package from MIWI-CloudRendering_Service.zip

11.3.1.1 *Development environment*

Node.js and build preparations

The web service uses node.js as the platform. It uses the provided `npm` tool to fetch and install the needed dependency components. Setup the development environment by installing the required dependencies for webapp, signalinservice and jsapp.

In order to proceed you need node.js installed on your system. Head over to http://nodejs.org/download/ and follow the instructions.

```
// Install neede dependencies from the node.js library repository
cd <web_service_source_code>
npm install

cd webservice
npm install

cd ../signalingservice
```

```
npm install


cd ../jsapp
npm install
```

Building

The web service uses grunt task automation framework to build and run the service.

If you don't have grunt installed head over to http://gruntjs.com/getting-started to get more information. Once node.js is installed, you can install the grunt tooling by running the following. The -g option will make the grunt command available globally in your system. You can run this install command anywhere on the system, you don't need to be in the web service source directory.

```
npm install -g grunt-cli
```

To build the web service run

```
cd <web_service_source_code>
grunt build
```

11.3.1.2 *Running*

As noted above the web service needs node.js and grunt to build and to run the application. Note that node.js applications are normally not shipped as binary distributions but built and ran from the source code.

Start the WebService running the command

```
cd <web_service_dir>
grunt run
```

Enable production mode which uses port 80 for HTTP and 443 for HTTPS with

```
// Windows
SET NODE_ENV=production
// Mac/Linux
export NODE_ENV=production
```

You should see the following

```
[2014-04-10 15:59:47.154] [INFO] console - Signaling server started
[2014-04-10 15:59:47.157] [INFO] console - Express server listening on
port 3000 in development mode
```

and with production mode

```
[2014-04-10 16:06:25.111] [INFO] console - Signaling server started
[2014-04-10 16:06:25.114] [INFO] console - Express server listening on
port 443 in production mode
[2014-04-10 16:06:25.115] [INFO] console - Redirecting http requests
on port 80 in production mode to https port 443
```

If you are going to run the service in production, we recommend using a deamon, such as `pm2`, `forever` or similar depending on your operating system. But this is outside of the scope of this tutorial.

Limiting dependencies to those needed in production can be done using the ´production´-flag (`npm install --production`) on the above mentioned steps.

11.3.2 Renderer

11.3.2.1 *Building from sources*

The renderer implementation is a realXtend Tundra plugin. This means that you need to build Tundra with the Cloud Rendering plugin. Building Tundra has been automated to be a relatively simple process, though it will take a lot of time to build. We are talking about a highly complex 3D (virtual world) SDK with a lot of 3rd party dependencies. Some prior experience is expected on Git, CMake and the C++ tooling of your operating system. If you are not a familiar building C++ projects it is recommended to use the prebuild binaries.

Building realXtend Tundra

You can find the Tundra source code from `https://github.com/realXtend/tundra`, you should start by reading the `README.md`. Here are the first steps to get you started.

1. `cd <tundra-build-destination>`
2. `git clone git@github.com:realXtend/tundra.git`
3. `cd tundra`
4. See here how to continue on your platform.

Building CloudRenderingPlugin

You can find the Renderer source code from https://github.com/Adminotech/fiware-cloud-rendering-renderer You can also find the FIWARE 3.3.3 Renderer source code release from MIWI-CloudRendering_Renderer_Sources.zip

1. `cd <tundra-build-destination>/src`
2. `git clone git@github.com:Adminotech/fiware-cloud-rendering-renderer.git`
3. Checkout the latest 3.3.3 release tag `cd fiware-cloud-rendering-renderer` and `git checkout fiware-cloudrendering-3.3.3`
4. See `fiware-cloud-rendering-renderer/CloudRenderingPlugin/README.md` how to build the needed dependencies
5. Add `add_subdirectory(src/fiware-cloud-rendering-renderer)` to the end of the main CMake prebuild configuration file `<tundra-build-destination>/CMakeBuildConfig.txt`
6. The cloud rendering plugin is now configured to be a part of the Tundra build. Run CMake and compilation on Tundra again to build it.

11.3.2.2 *Using prebuilt binaries*

11.3.2.2.1 *Windows*

There are several options on how to download. For Windows users the easiest option is the .zip release MIWI-CloudRendering_Renderer_Meshmoon-Rocket-2.5.3.1.zip but there is also a .msi installer available here MIWI-CloudRendering_Renderer_Meshmoon-Rocket-2.5.3.1.msi, which will automatically install the needed dependencies (listed in Software Requirements) . Both will install the same exact binary, just in a different manner, you will have more freedom with the .zip package but you'll need to install the dependencies by hand.

Here are the steps to install using the .zip package.

1. Download MIWI-CloudRendering_Renderer_Meshmoon-Rocket-2.5.3.1.zip
2. Extract the contained `MIWI-CloudRendering_Renderer_Meshmoon-Rocket-2.5.3.1` folder to a location of your choosing on your hard drive.

3. Make sure you have the needed dependencies installed from the #Software Requirements section.
4. See User and Programmers Guide for use instructions.

11.3.3 WebClient

The bundled example web client requires a running web service. In a development environment the web client can be accessed at http://localhost:3000

11.4 Sanity check procedures

11.4.1 End to End testing

Full systems test can be executed by adding a `test=true`-parameter to the WebClient URL. http://localhost:3000/service/receiver?test=true

11.4.2 List of Running Processes

Renderer

- Tundra executable (alternatively TundraConsole executable on Windows)

WebService

- Node-application that includes:
  - HTTPS-server that serves the client application
  - WebSocket-server for webrtc signaling
  - HTTP-server that redirects traffic to the HTTPS-server
- For universal access, a STUN/TURN service should be provided for the user. This is however out of the scope of this GE.

11.4.3 Network interfaces Up & Open

WebService

Web service requires one open port at minimum for the signaling server. Production uses the default HTTPS-port (443) both for the web server and the web socket connections but it can also redirect HTTP-traffic from port 80 to port 443.

In development mode port 3000 and 3001 are used for HTTP and HTTPS, consecutively.

STUN-server (not included in packages) requires two distinct IP-addresses. The standard listening port number for a STUN server is 3478 for UDP and TCP, and 5349 for TLS.

11.4.4 Database

N/A

11.5 Diagnosis Procedures

11.5.1 Resource availability

11.5.1.1 ***Web Service***

- 128 MB of RAM. Free disk space is not required for runtime operations.

11.5.1.2 *Renderer*

- 512 MB of RAM. Free disk space is used for caching assets, recommended to have 1 GB of free disk space.

11.5.2 Remote Service Access

N/A

11.5.3 Resource consumption

11.5.3.1 *Web Service*

- Low CPU utilization. Can be ran with normal process prioritization.

11.5.3.2 *Renderer*

- Potentially significant CPU, GPU and memory usage. This depends on the 3D environment that is being rendered. Should be ran with high process prioritization.

11.5.4 I/O flows

11.5.4.1 *Web Service*

- TCP ports 443, 80 and 8080 should be open. WebRTC signaling via STUN servers will take care of the rest.

11.5.4.2 *Renderer*

- TCP and UDP ports 2345-2350 should be open.

12 Display As A Service - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

12.1 Introduction

This document explains the installation of the DaaS GE. The installation includes the DaaS SDK, including a set of use cases, and the installation on mobile devices that should connect to the system.

12.2 System Requirements

12.2.1 Hardware Requirements

For the server part any standard Intel/AMD based X86 64bit hardware is sufficient.

Android virtual display devices need to support Android 4.3 or higher.

12.2.2 Operating System Support

PC:

```
Windows 7, 64 bit
Windows 8, 64 bit
```

Mobile:

```
Android 4.3+
```

12.2.3 Software Requirements

```
"x264", rev. 2363 (Commercial): http://www.videolan.org/developers/x264.html
"Ice", 3.4.2 (Commercial): http://www.zeroc.com/ice.html / Ice-E
"Boost", 1.54.0 ("Boost Software License 1.0": http://opensource.org/licenses/BSL-1.0): http://www.boost.org Asio, Date_Time, Filesystem, Log, Program_Options, Regex, Smart_Ptr, System, Thread
"Live555 Streaming Media", 2013-09-18 ("LGPLv3": http://opensource.org/licenses/lgpl-3.0.html): http://www.live555.com/liveMedia
"ffmpeg", 2.1 ("LGPLv3": http://opensource.org/licenses/lgpl-3.0.html): http://ffmpeg.org/
"OpenCV", 2.4.5 ("BSD 3-clause": http://opensource.org/licenses/BSD-3-Clause): http://opencv.org
```

12.3 Software Installation and Configuration

12.3.1 Install the Display As A Service SDK

1. Download unzip and the binaries (TBD)

12.3.2 Install APK's on Android

1. Enable installation of non-market apps on each device:

```
Settings -> Security -> Unknown Sources
```

2. Copy .apk files to device memory, e.g. over USB connection.

3. Locate .apk in any file browser on device.

```
e.g. My Files, Root Browser
```

4. Opening .apk files in file browsers starts the installation process automatically.

12.3.3 Configuration

To start and configure Display As A Service, enter the folder you extracted the binaries to and perform the following steps:

12.3.3.1 *restfulController*

- Start the restfulController.exe
- The restfulController listens to port 80. You can specify a different port by using the -p command.

```
e.g. restfulController.exe -p 8080 will listen to port 8080
```

12.3.3.2 *Virtual Display (VD)*

- Copy netvfbdisplay.cfg to the extracted netvfb folder
- You can configure the virtual display properties through the netvfbdisplay.cfg
- If your controller application is not running on the same machine you have to replace the IP in the DisplayControllerService section

```
e.g:
    [DisplayControllerService]
    url = netvfb://192.168.1.2:61000 if your controller runs on the
IP 192.168.1.2
```

- Start the virtual display by double-clicking netvfbdisplay.exe

12.3.3.3 *Virtual Frame Buffer (VFB)*

- Copy the colorstream.cfg to the extracted netvfb folder
- You can configure the VFB properties through the colorstream.cfg
- If your controller application is not running on the same machine you have to replace the IP in the ControllerService section

```
e.g:
    [ControllerService]
```

```
        url = netvfb://192.168.1.2:60000 if your controller runs on the
IP 192.168.1.2
```

- Start a VFB by double-clicking colorstream.exe

12.4 Sanity check procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. For Display As A Service, these Sanity Checks split up into checks for the different elements described in *Configuration*

12.4.1 End to End testing

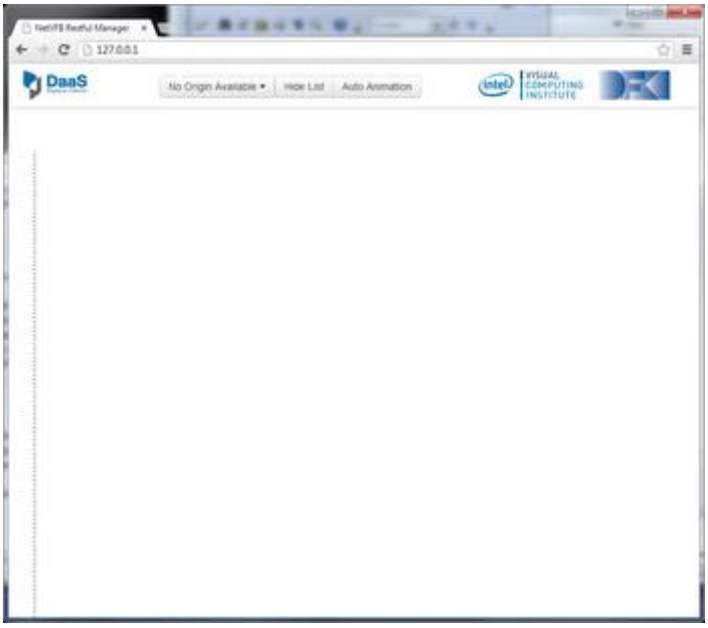
The following sanity checks can be performed right after having completed the respective step during configuratio

12.4.1.1 *restfulController*

Apart from the REST interface, restfulController also introduces a web frontend. After having started restfulController, perform the following steps:

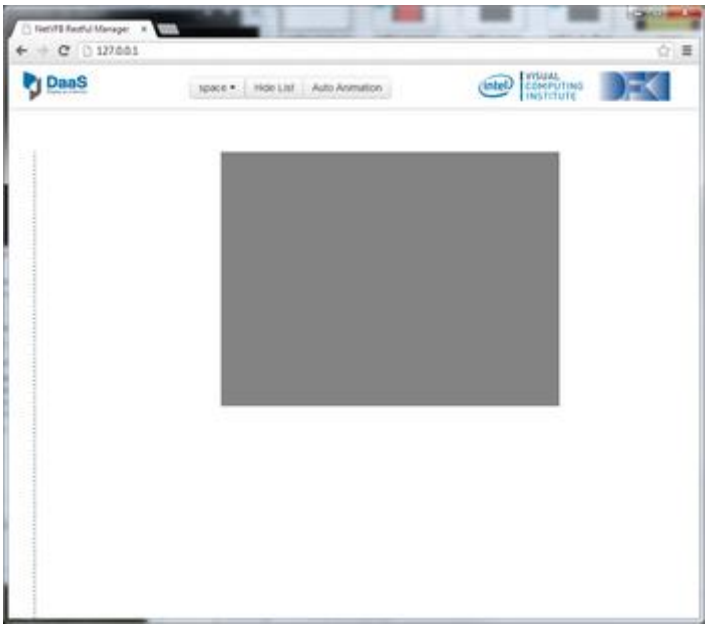
- Open a web browser and go to the following url: http://127.0.0.1
- if you configured a different port use http://127.0.0.1:<PORT>
- the restfulController binds to all available network interfaces. In order to use the web interface from a remote machine you have to specify the IP.
- e.g: http://192.168.1.2

The web interface should now look like this:



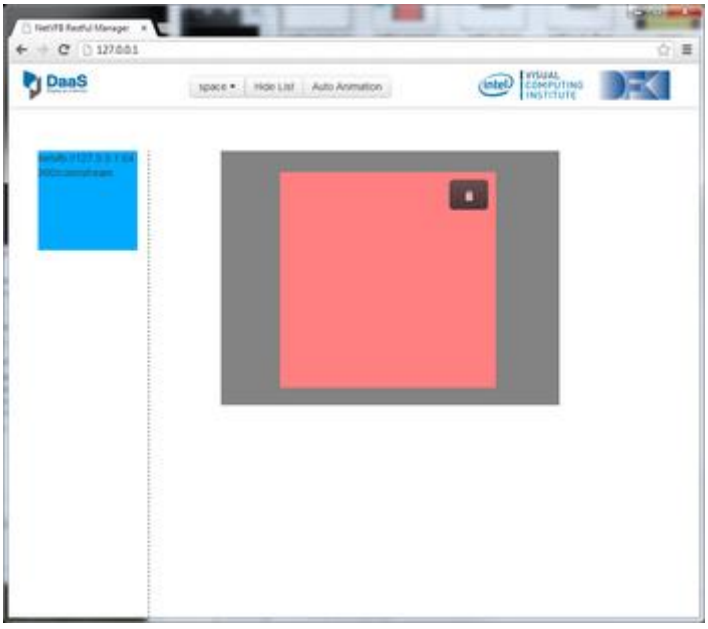
12.4.1.2 *Virtual Display*

After configuration and launch of Virtual Display, the display should also appear in the web frontend:

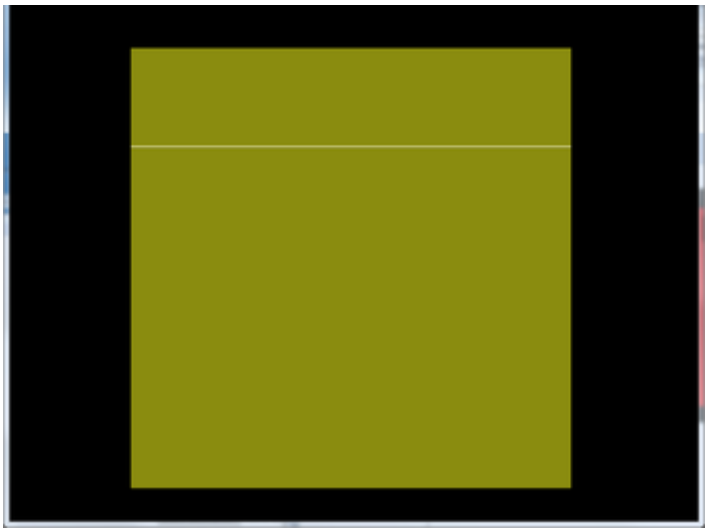


12.4.1.3 *Virtual Framebuffer*

After launch, the Virtual Framebuffer should appear in the web interface (left):



Drag the VFB (left) onto the Virtual Display. The Virtual Display should now contain the content of the virtual Framebuffer:



12.4.2 List of Running Processes

The applications described above (i.e. applications providing the controller or managing VFBs or displays) do not spawn any additional processes than the actual application process. That is, when you start your controller / VFB / display applications, you only have to check for the processes with the respective applications' names

12.4.3 Network interfaces Up & Open

- 6790 - 7500 UDP (RTP)
- 80 TCP (HTTP REST)
- 8554 - 8889 TCP (RTSP)
- 60000 - 60100 TCP (ICE-Services)
- 61000 - 61100 TCP (ICE-Services)
- 62000 - 62100 TCP (ICE-Services)
- 63000 - 63100 TCP (ICE-Services)
- 64000 - 64100 TCP (ICE-Services)

12.4.4 Databases

N/A

12.5 Diagnosis Procedures

12.5.1 Resource availability

Resource consumption of Display as a Service depends on the streamed content and the number of target displays the content is streamed to. The minimal requirement for content of a frame buffer to be streamed relates to the size of the streamed image. For example, a 200 x 200 pixel large image with 32 bit RGB will consume 200x200x4 bytes of process memory. As a rule of thumb, this memory consumption multiplies with the number of target displays the buffer is streamed to.

12.5.2 Remote Service Access

N/A

12.5.3 Resource consumption

The memory consumption should roughly scale with the size of the source image and the number of connected displays. Same is valid for CPU load. Display as a Service should not consume more than 20% of CPU on a 2GHz Quad Core.

12.5.4 I/O flows

N/A

13 GIS Data Provider - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

13.1 Introduction

The purpose of this document is to provide information how to set up GIS data service.

13.2 System Requirements

Setup has been verified with:

- Ubuntu 13.04
- OpenJDK 7
- Maven 2
- Git

13.2.1 Hardware Requirements

Configuration has been tested with:

- Intel i5
- 8GB RAM

13.2.2 Operating System Support

System configuration has been verified with Ubuntu versions 12.10 and 13.04.

13.2.3 Software Requirements

Required and verified toolset to run the GIS Data Provider:

- OpenJDK 7
- Maven 2
- Git

13.3 Software Installation and Configuration

13.3.1 Required toolset

- Tools
    - OpenJDK 7
    - Maven 2
    - Git
    - PostGis

Tools can be installed with following command:

```
sudo apt-get install maven2 openjdk-7-jdk git
```

13.3.1.1 *Setup PostgreSQL with PostGis*

Following shell script installs needed libraries for Posgresql and PostGis. PostGis source code is downloaded and automatically compiled, after compile PostGis is installed. Script also install pgAdmin3 which is GUI for managing PostGIS database. If *python-qgis* installation fails *UbuntuGis* PPA needs to be added to system.

Add *UbuntuGis* PPA to system:

```
--------------------------------

sudo apt-get install python-software-properties // Needed only if apt-
add-repository fails
sudo apt-add-repository http://ppa.launchpad.net/ubuntugis/ppa/ubuntu
sudo apt-get update
```

Shell script installs needed libraries for Posgresql and PostGis. Create in example *install_postgis.sh* script file and assign to as executable.

```
-------------------------------------------------------------------------
-------------------------------------------------------------------------
------------------

#!/bin/bash

sudo aptitude install libgdal-dev libpq-dev \
postgresql postgresql-server-dev-all postgresql-contrib postgresql-
client-common \
libgeos-dev libproj0 libgdal-dev libgdal1 libgeos-c1 libgeos-dev
python-qgis python-qgis-common pgadmin3 libproj-dev

wget http://download.osgeo.org/postgis/source/postgis-2.1.0.tar.gz
tar xzf postgis-2.1.0.tar.gz
cd postgis-2.1.0
./configure
make -j4
sudo make install
```

Default username for PostgrSQL installation is *postgres*. **createuser** -command in terminal shell creates new user to PostgreSQL so that Linux / UNUX IDENT authentication can be used.

13.3.2 Setup GeoServer with W3DS community module

13.3.2.1 *Compile GeoServer and run it with Jetty server*

For the latest version of the GeoServer it is recommended to build it from the sourcecodes found from Github.

First Java, Maven and Git needs to be installed. After this GeoServer codes can be cloned from Github.

GIS GE related changes are in xml3d brach and it can be cloned from command line with command

```
clone:

git clone -b xml3d https://github.com/Cyberlightning/geoserver.git
```

After GeoServer project is cloned it needs to be build with Maven. In addition to default GeoServer build command **-Pw3ds** switch needs to be given. This enables build process also for

W3DS module which contains XML3D support. Build is done in **src**-folder in root of GeoServer file structure.

```
Example:
cd geoserver/src
mvn install -DdownloadSources=true -Pw3ds
```

Above command executes also automated tests. If Automated tests wants to be skipped, it can be done by using **-DskipTests=true** -switch.

```
mvn install -DdownloadSources=true -DskipTests=true -Pw3ds
```

Build generates *geoserver.war* package to *"Geoserver_root_directory"/src/web/app/target*. After building GeoServer succesfull build is indicated similarly as below examle shows:

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] GeoServer ..... SUCCESS [0.757s]
[INFO] Core Platform Module ..... SUCCESS [0.465s]
[INFO] Open Web Service Module ..... SUCCESS [0.314s]
[INFO] Main Module ..... SUCCESS [2.328s]
[INFO] Web Feature Service Module ..... SUCCESS [0.295s]
[INFO] Web Coverage Service Module ..... SUCCESS [0.103s]
[INFO] Web Map Service Module ..... SUCCESS [0.315s]
[INFO] GeoServer Web Modules ..... SUCCESS [0.039s]
[INFO] Core UI Module ..... SUCCESS [0.935s]
[INFO] Security UI Module ..... SUCCESS [0.262s]
[INFO] GeoServer Security Modules ..... SUCCESS [0.095s]
[INFO] GeoServer JDBC Security Module ..... SUCCESS [0.235s]
[INFO] GeoServer LDAP Security Module ..... SUCCESS [0.165s]
[INFO] Web Coverage Service 1.0 Module ..... SUCCESS [0.105s]
[INFO] Web Coverage Service 1.1 Module ..... SUCCESS [0.219s]
[INFO] KML support for GeoServer ..... SUCCESS [0.114s]
[INFO] GeoWebCache (GWC) Module ..... SUCCESS [0.164s]
[INFO] REST Support Module ..... SUCCESS [0.078s]
[INFO] REST Configuration Service Module ..... SUCCESS [0.158s]
[INFO] WMS UI Module ..... SUCCESS [0.101s]
[INFO] GWC UI Module ..... SUCCESS [0.142s]
[INFO] WFS UI Module ..... SUCCESS [0.074s]
[INFO] Demoes Module ..... SUCCESS [0.094s]
[INFO] WCS UI Module ..... SUCCESS [0.093s]
[INFO] Community Space ..... SUCCESS [0.028s]
[INFO] Web 3D Service ..... SUCCESS [0.111s]
```

```
[INFO] GeoServer Web Application ..... SUCCESS [1.178s]
[INFO] GeoServer Extensions ..... SUCCESS [0.174s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 9.834s
[INFO] Finished at: Mon Dec 02 13:43:31 EET 2013
[INFO] Final Memory: 59M/726M
[INFO] ------------------------------------------------------------------------
```

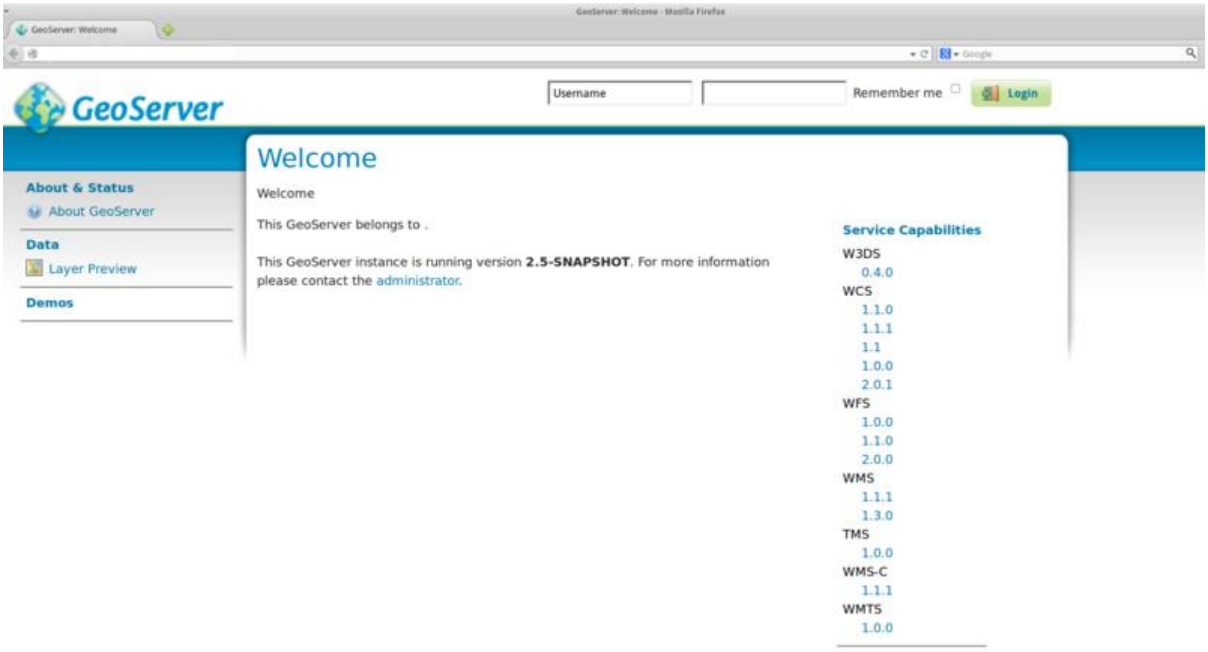
Make sure that **Web 3D Service** build is successful, it is W3DS module with XML3D extension. If building fails, verify that tools are correctly installed and git repository is successfully cloned.

Now when GeoServer is successfully build, it can be launched. GeoServer.war -package can be deployed in example to Tomcat, but since Jetty is bundled to GeoServer source code package using it is straightforward. To start GeoServer first go to *"GeoServer_root_directory"/src/web/app* -directory and execute following command:

```
mvn -Djetty.port=9090 jetty:run -Pw3ds
```

This starts GeoServer with W3DS module and assigns port 9090. Default port for Jetty is 8080. When terminal shows `"[INFO] Started Jetty Server"` -text jetty server has started Geoserver. GeoServer admin panel can be find from the URI: `localhost:9090/geoserver`. Default username is **admin** and password **geoserver**.

The default control panel is shown below:



13.3.2.2 *Deploy provided geoserver.war*

GIS Data Provider release contains generated geoserver.war packet which is snapshot version from the released GE content. Content is based on [commit b647e55fa3549785f5726b3307482f1aff3e37e5]. war-package can be deployed in example to the Tomcat. For the latest version of the GIS Data Provider implementation it is recommended to get GeoServer source codes and build GeoServer as described in *Compile GeoServer and run it with Jetty server* -section.

13.3.3 Configuration

13.3.3.1 *Memory configuration*

Allocated memory usage in Jetty can be defined with DMAVEN_OPTS="-Xmx2048m -Xms1024m". This can be done when Jetty server is launched. For example:

```
DMAVEN_OPTS="-Xmx4096m -Xms2048m" mvn -Djetty.port=9090 jetty:run -Pw3ds
```

13.3.3.2 *Configuration for allowing cross-origin requests*

By default server doesn't allow cross-origin requests. Cross-origin request can be allowed by adding following options to **src/web/app/pom.xml** and **src/web/app/src/main/webapp/WEB-INF/web.xml**

src/web/app/pom.xml:

```
<dependency>
  <groupId>org.eclipse.jetty</groupId>
  <artifactId>jetty-servlets</artifactId>
  <version>9.0.6.v20130930</version>
</dependency>
```

src/web/app/src/main/webapp/WEB-INF/web.xml:

```
<filter>
  <filter-name>cross-origin</filter-name>
  <filter-class>org.eclipse.jetty.servlets.CrossOriginFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>cross-origin</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

13.4 Sanity check procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

13.4.1 End to End testing

If GeoServer started with suggested command, it is currently running on localhost 9090 -port. To verify that GeoServer is running type http://localhost:9090/geoserver to your web browser. Geoserver admin panel should be opened. Default username is *admin* and password *geoserver*. Login should be possible with these credentials. In the main page there is **Service Capabilities**-section. Press 0.4.0 under **W3DS**, link requests GeoServer W3DS capabilities, server responds

this with XML-response. If response is successfully loaded to new page GeoServer capability requests are done correctly.

GeoServer logs can be see in Geoserver logs. Log information shows possible error situations which help to find actual problem.

GeoServer status page contains useful buttons to free memory, clear cache or reload configuration. Using these can also help solving problem situation.

13.4.2 List of Running Processes

Verify that **Jetty** is running with following command:

```
ps aux | grep jetty
```

13.4.3 Network interfaces Up & Open

If proposed Jetty command is used, port 9090 should be free. If other free port needs to be used, assign it correctly when starting Jetty server.

13.4.4 Databases

It is possible to store GIS data with 3D information straight to GeoServer itself so separate database is not necessary required. If separate database wants to be installed, PostGis is one option. More information of the PostGis usage with GeoServer can be found here.

As the database selection is user definable, the sanity check for it is the same. However, if PostGIS has been installed as indicated by this guide, the easiest way to sanity check its functionality is to use pgAdmin3 tool, and login to the database with default used "postgres". Having a successful connection with matching information about the database structure visible via the tool will satisfy the sanity check requirement.

13.5 Diagnosis Procedures

13.5.1 Resource availability

Just after start memory consumption is about 1,5GB. During testing memory usage variation was between 3-4GB which was seen as normal behavior.

GeoServer with modified W3DS module in Git requires ~300MB hard disk space. Compiled geoserver.war package is ~60MB which can be deployed to web server.

13.5.2 Remote Service Access

The purpose of this GE is to serve geographical data for clients, either browser or native based. There are no dedicated links for other enablers for this purpose, however, nothing stops other enablers from using the service via the same HTTP interface as the regular clients do.

13.5.3 Resource consumption

Just after start memory consumption is about 1,5Gb. Normal memory usage is between 3-4Gb. This estimation is based on the GIS GE test data usage.

CPU usage is related to operations, e.g. CPU usage can vary between 0-100%.

13.5.4 I/O flows

I/O flow amount is highly depend of the bounding box area and what is the detail level inside bounding box. Therefore I/O flow can drastically change even though requested bounding box area is same. I/O flow type is HTTP and admin parameters can be encrypted.

14 POI Data Provider - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

14.1 Introduction

The purpose of this document is to provide the required information for a system administrator in order to install and configure the POI (Points of Interest) Data Provider generic enabler reference implementation. The POI GE is implemented as a RESTful web service using PHP programming language. It is described in more detail in FIWARE.OpenSpecification.MiWi.POIDataProvider.

14.2 System Requirements

14.2.1 Hardware Requirements

The POI Data Provider should work on any modern PC computer that is capable of running Ubuntu Server 12.04. Therefore, the bare minimum requirements are:

- 300 MHz x86 processor
- 128 MiB of system memory (RAM) (256 MiB for a virtual installation)
- 1 GB of disk space

For a small practical deployment, the recommended system is:

- Dual core CPU
- 4 GB of system memory (RAM)
- 50 GB of disk space

The hardware needs of the POI Data Provider are mainly dominated by the databases (PostgreSQL and MongoDB), and as such the two most important factors are:

- memory size (the bigger the better)
- disk size
- disk speed (e.g. using SSD drives)

You can have a rough estimate for the required disk space by using the following formula:

```
size_on_disk = number_of_pois * 10KB
```

So, for example for one million (1000000) POIs, you get the following estimate for data size on disk:

```
size_on_disk = 1000000 * 10KB = 10000000 KB ~ 10GB
```

However, this estimate may be inaccurate for many cases, as the mean POI size can be much smaller or larger than 10 kilobytes.

14.2.2 Operating System Support

The implementation has been tested with Ubuntu 12.04 and Ubuntu 12.10. It SHOULD, however, work with any Linux distribution where the required software is available.

These instructions can also be very easily applied to any Debian-based Linux distribution, where the 'apt-get' installation utility is available.

14.2.3 Software Requirements

In order to have the POI Data Provider up and running, the following software is required:

- PostgreSQL 9.1
  - PostGIS 1.5.x spatial database extension
- MongoDB 2.0.x
- Apache HTTP Server 2.2.x
  - (Or basically any HTTP server with PHP support)
- PHP 5.x
  - PostgreSQL module
  - MongoDB module
  - JSON Schema for PHP [1]

14.3 Software Installation and Configuration

14.3.1 Installing required packages

The required software packages can be installed using the 'apt-get' command-line package installation tool:

```
 $ sudo apt-get install postgresql postgresql-contrib postgis
postgresql-9.1-postgis
 $ sudo apt-get install mongodb
 $ sudo apt-get install apache2
 $ sudo apt-get install php5 php5-pgsql
 $ sudo apt-get install git
```

The installation of the MongoDB module for PHP5. (see [2] for more information):

```
 $ sudo apt-get install php-pear php5-dev gcc make
 $ sudo pecl install mongo
 $ sudo touch /etc/php5/conf.d/mongo.ini
```

Add this line of content to /etc/php5/conf.d/mongo.ini:

```
 extension=mongo.so
```

Restart Apache web server:

```
  $ sudo /etc/init.d/apache2 restart
```

14.3.2 Configuring PostGIS

1. Create GIS database user:

```
 $ sudo -u postgres createuser gisuser
```

2. Create database owned by that user

```
 $ sudo -u postgres createdb --encoding=UTF8 --owner=gisuser
poidatabase
```

3. Activate PostGIS on the created database:

```
 $ sudo -u postgres psql -d poidatabase -f
/usr/share/postgresql/9.1/contrib/postgis-1.5/postgis.sql
 $ sudo -u postgres psql -d poidatabase -f
/usr/share/postgresql/9.1/contrib/postgis-1.5/spatial_ref_sys.sql
 $ sudo -u postgres psql -d poidatabase -f
/usr/share/postgresql/9.1/contrib/postgis_comments.sql
 $ sudo -u postgres psql -d poidatabase -c "GRANT SELECT ON
spatial_ref_sys TO PUBLIC;"
 $ sudo -u postgres psql -d poidatabase -c "GRANT ALL ON
geometry_columns TO gisuser;"
```

4. Enable UUID functions for that database:

```
 $ sudo -u postgres psql -d poidatabase -c 'create extension "uuid-
ossp";'
```

5. Grant local access to the database:

Before you can access the database, you must edit PostgreSQL configuration to allow local unix socket connections (from the same computer where the database is running) without password.

Edit the file /etc/postgresql/9.1/main/pg_hba.conf and change the following line:

```
# "local" is for Unix domain socket connections only
local   all             all                                     peer
```

---> Change to --->

```
# "local" is for Unix domain socket connections only
local   all             all                                     trust
```

6. Restart PostgreSQL

```
  $ sudo /etc/init.d/postgresql restart
```

14.3.3 Installing POI Data Provider

1. Fetch the POI Data Provider from git:

```
 $ git clone https://github.com/Chiru/WeX
```

2. Create required database tables using the provided script:

```
 $ cd WeX/poi/install_scripts
 $ ./create_tables.sh
```

3. Copy the folder "WeX/poi/php" from the cloned project under /var/www, e.g. to /var/www/poi_dp

Installation of JSON Schema for PHP

In order to install the JSON Schema for PHP implementation, follow istructions available at [3]. After the library and the required dependencies have been downloaded, copy the 'vendor' subdirectory under the same directory where you place the POI-DP PHP implementation, e.g. /var/www/poi_dp/

14.3.4 Importing POI data from OpenStreetMap

You can populate the POI database using for example the open map data available from OpenStreetMap. Here are the high-level guidelines for the import process.

1. Download OpenStreetMap data extract(s) (e.g. from [4])

2. Import the data to PostGIS (e.g. using [5])

3. Convert imported data to POI-DP table format using SQL.

14.3.5 Enabling Cross-origin Resource Sharing in Apache

Cross-origin Resource Sharing (CORS) is required if the POI-DP client is a web application that is hosted on a different domain than the POI-DP backend. In practice this means that the POI-DP Apache server needs to add the following HTTP header for each response:

```
Access-Control-Allow-Origin "*"
```

In order to enable CORS in Apache, please follow the instructions found in [6].

14.4 Sanity check procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

14.4.1 End to End testing

You can do a quick test to see if everything is up and running by accessing the following URL:

```
http://hostname/poi_dp/spatial_search?lat=1&lon=1&category=test_poi
```

You should get a JSON structure representing a test POI as a response.

14.4.2 List of Running Processes

You can use the following command to check if Apache HTTP server, PostgreSQL and MongoDB are running:

```
$  ps ax | grep 'postgres\|mongo\|apache2'
```

The output of the command should be something like the following:

```
 8404 ?        Ssl   37:07 /usr/bin/mongod --config /etc/mongodb.conf
12380 ?        S      0:00 /usr/sbin/apache2 -k start
12381 ?        S      0:00 /usr/sbin/apache2 -k start
```

```
12382 ?        S      0:00 /usr/sbin/apache2 -k start
12383 ?        S      0:00 /usr/sbin/apache2 -k start
12384 ?        S      0:00 /usr/sbin/apache2 -k start
17966 ?        Ss     0:20 /usr/sbin/apache2 -k start
18845 ?        S      0:00 /usr/sbin/apache2 -k start
21262 ?        S      0:00 /usr/sbin/apache2 -k start
21263 ?        S      0:00 /usr/sbin/apache2 -k start
27956 ?        S      0:00 /usr/lib/postgresql/9.1/bin/postgres -D
/var/lib/postgresql/9.1/main -c
config_file=/etc/postgresql/9.1/main/postgresql.conf
27958 ?        Ss     0:00 postgres: writer process
27959 ?        Ss     0:00 postgres: wal writer process
27960 ?        Ss     0:00 postgres: autovacuum launcher process
27961 ?        Ss     0:00 postgres: stats collector process
28100 pts/0    R+     0:00 grep --color=auto postgres\|mongo\|apache2
```

14.4.3 Network interfaces Up & Open

The only required port open to the Internet is TCP port 80, used by HTTP protocol.

14.4.4 Databases

The POI Data Provider utilizes two database systems:

- PostgreSQL/PostGIS for storing and accessing data components with spatial data (fw_core data component)
- MongoDB for storing and accessing all other data components that do not require spatial searches

PostgreSQL

PostgreSQL has a database named 'poidatabase'. It contains a table called 'fw_core' and it contains the core information, such as name and location, about the POIs.

You can test if this table is succesfully created with the following commands:

```
$ psql -U gisuser poidatabase
poidatabase=> SELECT count(*) FROM fw_core;
```

If the table was created succesfully, this query should return '4', as there sould be four test POI entries created by the installation.

MongoDB

MongoDB should also contain a database named 'poi_db'. It should contain a collection named 'testData' containing a single test POI entry, created by the installation.

You can test if MongoDB was succesfully configured with the following commands:

```
$ mongo
```

```
> use poi_db
> db.collections
```

The db.collections command should list five POI data component collections created by the installation: fw_contact, fw_marker, fw_media, fw_relationships and fw_time.

14.5 Diagnosis Procedures

The Diagnosis Procedures are the first steps that a System Administrator will take to locate the source of an error in a GE. Once the nature of the error is identified with these tests, the system admin will very often have to resort to more concrete and specific testing to pinpoint the exact point of error and a possible solution. Such specific testing is out of the scope of this section.

14.5.1 Resource availability

The amount of available resources depends on the size of the database and the usage rate of the service. The minimum recommended available resources are:

- Available memory: 256 MB
- Available disk space: 10 GB

14.5.2 Resource consumption

The load value reported e.g. by the 'top' utility should not exceed the number of CPU cores in the system. If this happens, the performance of the system can dramatically drop.

14.5.3 Remote Service Access

Check that the HTTP port (80) is open and accessible from all the networks from which POI-DP will be used.

14.5.4 I/O flows

All the incoming and outgoing data of the POI Data Provider will go through TCP port 80. The size of the flow is entirely dependant on the usage of the service, e.g. number of users.

15 2D-3D Capture - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

15.1 Introduction

This Document describes the System requirements for 2D 3D capture components. This component consists of 2 parts in the form of

- JavaScript API and
- python based server component.

JavaScript API does not need installation and usage of the component is addressed in the user guide section thoroughly. This document provide the basic set up to run the web based demos and Majority of this document discusses preparation, installation and running sanity tests on Python server.

15.2 System Requirements

15.2.1 Hardware Requirements

REST server, the python script, is an image and meta data capturing server. In order to run the service on a server successfully for a long time, server required to have sufficient memory allocations based on the number of clients and amount of data per client required to be handled. Images uploads require sufficient network capabilities for the software to run smoothly. requirements to the software as well as data storage facilities. Code available at the moment is not optimized for scalability. Code is implemented as a proof of concept hence load handling is not also considered to be out of scope. In order to run the run the server a computer with more than 1TB of storage with 8GB RAM is preferable.

15.2.2 Operating System Support

15.2.2.1 ***Server Side***

Python server is verified on following operating systems.

- Ubuntu 12.10
- Ubuntu 13.04
- Ubuntu 13.10

15.2.2.2 ***Client side for Web Demo Components***

Theoretically all platforms supporting Mobile Firefox or Mobile Chrome. Code is tested to work on android devices API level higher that 4.0

15.2.3 Software Requirements

15.2.3.1 ***Python Rest Server***

- MySQL
- Python 2.7
- PIP

The Python server also depends on following modules.

- Flask 0.10.1
- MySQLdb

15.2.3.2 *Image tagging services*

Image tagging service of the 2D3D capture server is running on a tomcat server as a Servlet. Following are the dependencies of this service:

- Apache tomcat
- json_simple 1.1
- pngj 2.0

json simple and pngj jars are included in the deployable war file along with the sources located at GitHub.

15.2.3.3 *Web Demo Components*

- Tomcat Server
- Mobile Firefox/Mobile Chrome

15.3 Software Installation and Configuration

15.3.1 Installing Pre requisits for the server code

- Following commands will install these software

```
$sudo apt-get install mysql-server
$sudo apt-get install mysql-client
$sudo apt-get install python-dev libmysqlclient-dev
$sudo apt-get install python-pip
$sudo pip install -U pip
$sudo pip install Flask
$sudo pip install MySQL-python
$sudo apt-get install libjpeg-dev libfreetype6 libfreetype6-dev
zlib1g-dev
```

- Latest MySQL-python 1.2.4 depends on distribute and the version 0.6.28 or higher. If this is not available it is possible to install it or upgrade the installed version. Commands are as follows.

```
sudo easy_install -U distribute
 OR
sudo pip install --upgrade distribute
```

- Python Image library depends on the zip library and need to link properly. Incase of a install time error this can be fixed by following this link.
- Installation of tomcat server can be carried out using the instructions on this page.
- Create Databases. This is explained thoroughly in the database section.

15.3.2 Installation of the Database

Python Rest Server depends on a MySQL database. Installation instructions for installing MySQL in Ubuntu can be found in Ubuntu Site. Advanced installation is not required for the demo application for further developments it may be useful. Once this is carried out MySQL client need to be installed as indicated in the section above.

Once this is completed application needs a user for the database and adding and removing of users is explained in the official MySQL website.

Then following commands can be executed to create the database. Prepared sql script to create tables is also included along with the code in the repository. Following are the steps to carry out. First steps are to create database and then tables need to be created.

```
mysql -h localhost -u twijethilake -p
Enter password:
mysql> create database 2d3dcapture;
mysql> exit;
$ mysql -h localhost -u twijethilake -p 2d3dcapture <
script_file.sql
Enter password:
$ mysql -h localhost -u twijethilake -p
Enter password:
mysql> source script_file.sql
```

15.3.3 Installation of the server

- Installing ImageTaggingServlet
    - Image tagging service, a main part of the server component is implemented as a java servlet. Tomcat server is expected to run on the same physical server as the python Rest server and Tomcat server is expected run on port 9090 for the war file to run out of the box. By default tomcat is started on the port 8080.
    - This can be changed by changing the server.xml located in the {TomcatDIR}/conf folder. Changing the port parameter of the following line from 8080 to 9090. Tomcat server needs restarting once this is done and the installation instructions page of tomcat includes information about restarting the server. {TomcatServer}/RUNNING.txt file contains this information as well.

```
<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000"
redirectPort="8443" />
```

- Source code can be downloaded as follows-

```
$ git clone https://github.com/Cyberlightning/Cyber-WeX.git
```

- Set up properties file.

Below is the example of the server.properties file. Absolute path to the folder that images are saved can be and should be changed by file.path of the server.properties file. In order to changes to take effect both file.path and local_image_repo parameters need to be changed.

```
[SectionJava]
test.path=.   //Need to be set to run the unit tests.
```

```
   file.path=/home/user/Software/apache-tomcat-8.0.0-
RC1/webapps/images/  //path the the image repository. This should be a
location that is accessible by the web browser via a public URL

   [SectionPython]

   tomcatport = 9090 //port of the tomcat server

   wsport = 17322 //port of the web socket

   restport = 17321 //Port for the web socket

   lhost = localhost

   serverurl=localhost // public host name where the applications are
expected to run eg. "dev.cyberlightning.com"

   repo_url =localhost:8080/images/

   dbuser = dbuser   //database username

   dbpassword =dbpassword //database password

   dbname = 2d3dcapture //database name.

   local_image_repo = /home/user/Software/apache-tomcat-8.0.0-
RC1/webapps/images //same as file.path indicates the location where the
images are saved.
```

- There are two ways to carry out the java component installation.

  1. By copying the war file to the Tomcat webapps folder and prepared for localhost testing. In this way installation can be tested out of the box which allows the application to be tested on a desktop but can not be run on a mobile desktop.

```
 $ cp -r Cyber-WeX/2D-3D-Capture/MavenProject/TwoDThreeDCapture/
{tomcat.folder}/webapps
```

  2. Using maven for installation.

In order to run the demos explained in the user guide this method should be over the first option. This page covers the basics of maven which also includes installation.

- Follow This page to activate tomcat manager application. Maven depend on the tomcat manager application to upload the the war file in to the webapps folder. Advantage of using maven is when necessary parameters set, remote tomcat server can be used to tun the client as well.
- With maven installed in the system, run the following command on command prompt for the first deployment. Credentials to the tomcat server should be inserted to the configuration files as indicate by the official documentation . For further deployments, tomcat7:deploy tag should be replaced with tomcat7:redeploy. Following commands assume execution from the location where the git command is executed.

```
   cd Cyber-WeX/2D-3D-Capture/MavenProject

   mvn clean compile tomcat7:deploy -Dtomcat.username=mavenuser -
Dtomcat.password=thisisthemavenpassword
```

- Python script for the server is available in GitHub. It is also available in the parent directory of the web application available for download. Following is the way to start the server. Assuming the user is still in the MavenProject Folder.

```
   python EventService.py --log debug
```

Uploaded images are saved in a web server and ImageTaggingServlet.java class assumes that "{TomcatServer}/webapps/images" as the location where all the images are saved. Server file also depend on the parameters of the server.properties file. By the default parameters in the server.properties server component is set to run to run on a local host environment hence the parameters are set accordingly. In order to change default parameters service.properties file need to be set up correct. Example file is available with the server source files. An example of the server.properties file is also shown previously in this document.

--log parameter indicates the output logging required from the server. This is directly related to python logging levels and can be set to debug, info, critical or error(Case insensitive). Text logs are by default to info this --log concerns only the screen output level.

15.3.4 Installing Web Demo Components

Client web components are installed in the server along with the servlets in the steps described in the section above. There are two demo web clients.

1. Data capturing mobile web client
   This component resembles an example implementation of 2D3D capture mobile client. It subscribes for events and provides means to capture video and image from the browser and upload them to the Rest server.
2. Data Visualizing web component
   This is a sub component of the Rest Server where a user can browse how the data located in the database and visualize how the sensor information can be mapped to a 3D environment.

Web pages that are related to both the components are deployed as a single war file. Both steps that described are valid in this section as well.

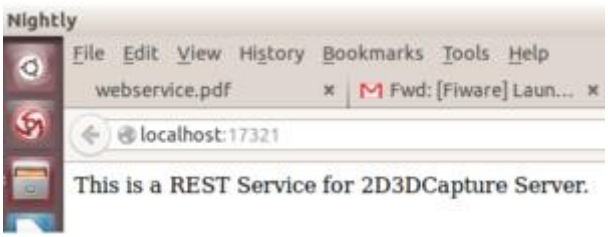
GitHub contains sources for the web components along with the deploy-able war file. Copying the war file to Tomcat webapps folder completes the deployment. Tomcat server supports hot deployment but may be required to restart. This Tomcat server is expected run on a public host which is accessible over the internet for device to access via public URLs. By default the service s are set to access information from the localhost server and need to be adjusted using the properties file described above.

15.4 Sanity check procedures

15.4.1 End to End testing

15.4.1.1 *Web Component*

- Once the RestServer is deployed(After running the EventService.py) accessing root of the server (e.g. localhost:17321) should look like the following image.



In order to test the web socket server following webpage is prepared. After the step above

```
http://www.example.org/DesktopSubscriptClient_Demo1.html
```

- Press "Start server"

- Press "Init"

- Press "Close Server"

- In the back end server logs should look like as follows.

15.4.1.2 *Python Script*

Run the python server according to the instructions given in section above.

Web Clients are preferably run on Firefox nightly builds or on Chrome. Once the web application and the server is installed properly a user should be able to access the following page and upload a image to the specified server with out errors. Failing to do so means an error in configuration.

```
   1. Open http://<Link to the tomcat
server>/TwoDThreeDCapture/Camerafeed.html
   2. Press Video start button
   3. Press Take and upload a snapshot.
```

Check the server logs for errors. This test confirms that the server is able to communicate the server.

15.4.2 List of Running Processes

```
   *python
   *tomcat7
   *chrome/Firefox
```

15.4.3 Network interfaces Up & Open

```
 By default
   *TCP : 17321
   *TCP : 17322
   *TCP : 9090 Tomcat
```

15.4.4 Databases

Implementation depends on mysql localhost database. There are 3 tables created in order to manage image metadata and to manage to event subscriptions. Check that MySQLd is running and is available with the created user/password credentials: Then following commands can be executed to create the database.

```
  tharanga@tharanga-CG8270:~/git/Cyber-WeX/2D-3D-Capture/MavenProject$
mysql -h localhost -u XXXXXXXX -p
  Enter password: XXXXXXXXXX

  mysql> use 2d3dcapture
  Reading table information for completion of table and column names
  You can turn off this feature to get a quicker startup with -A

  Database changed
  mysql> show tables;
  +-----------------------+
  | Tables_in_2d3dcapture |
  +-----------------------+
  | Imagedata             |
  | SubscriptionData      |
  | Subscriptions         |
  +-----------------------+
  3 rows in set (0.00 sec)

  mysql> describe Imagedata;
  +------------------+-------------+------+-----+------------------+-
---------------------------+
  | Field            | Type        | Null | Key | Default          |
Extra                      |
  +------------------+-------------+------+-----+------------------+-
---------------------------+
  | imagename        | varchar(45) | NO   | PRI | NULL             |
|
  | location         | point       | NO   | MUL | NULL             |
|
  | altitude         | float       | YES  |     | 0                |
|
  | accuracy         | float       | YES  |     | 0                |
|
  | accelerationgx   | float       | YES  |     | 0                |
|
```

```
  | accelerationgy    | float       | YES  |     | NULL                |
|
  | accelerationgz    | float       | YES  |     | 0                   |
|
  | rotationalpha     | float       | YES  |     | 0                   |
|
  | rotationbeta      | float       | YES  |     | 0                   |
|
  | rotationgamma     | float       | YES  |     | 0                   |
|
  | screenorientation | float       | YES  |     | 0                   |
|
  | deviceorientation | varchar(10) | YES  |     | NULL                |
|
  | time              | timestamp   | NO   |     | CURRENT_TIMESTAMP |
on update CURRENT_TIMESTAMP |
  | OS                | varchar(30) | YES  |     | NULL                |
|
  | Browser           | varchar(20) | YES  |     | NULL                |
|
  | devicetype        | varchar(30) | YES  |     | NULL                |
|
  +-------------------+-------------+------+-----+-------------------+-
---------------------------+
  16 rows in set (0.05 sec)


  mysql> describe Subscriptions;
  +------------------+-----------+------+-----+-------------------+----
------------+
  | Field            | Type      | Null | Key | Default           |
Extra          |
  +------------------+-----------+------+-----+-------------------+----
------------+
  | subid            | int(11)   | NO   | PRI | NULL              |
auto_increment |
  | time             | timestamp | NO   |     | CURRENT_TIMESTAMP |
|
  | subscriptiontype | int(11)   | NO   |     | NULL              |
|
  +------------------+-----------+------+-----+-------------------+----
------------+
  3 rows in set (0.00 sec)


  mysql> describe SubscriptionData;
```

```
+----------+---------+------+-----+---------+----------------+
| Field    | Type    | Null | Key | Default | Extra          |
+----------+---------+------+-----+---------+----------------+
| dataid   | int(11) | NO   | PRI | NULL    | auto_increment |
| subid    | int(11) | NO   | MUL | NULL    |                |
| location | point   | NO   |     | NULL    |                |
| pitch    | float   | YES  |     | NULL    |                |
| yaw      | float   | YES  |     | NULL    |                |
| roll     | float   | YES  |     | NULL    |                |
+----------+---------+------+-----+---------+----------------+
6 rows in set (0.00 sec)

mysql>
```

15.5 Diagnosis Procedures

15.5.1 Resource availability

Resource requirements for the server is minimal for testing purposes. Currently load testing has not been carried out hence considered out of the scope for large uploads.

15.5.2 Remote Service Access

For diagnosis the server is able to run on the debug mode where it produces necessary errors on occurring an error. Relevant command is as follows.

```
python EventService.py --log debug
```

On the Mobile web client it is possible to start a debug window which is described on the user guide.

This GE is not directly used by other GEs, hence no actual remote service access is described.

15.5.3 Resource consumption

CPU consumption stays less than 1% during single image upload.

15.5.4 I/O flows

TCP ports 17321,17322,9090 should be accessible by default. TCP port 3306 (MySQL) has to be available locally only (localhost).

16 Augmented Reality - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

16.1 Introduction

The purpose of this documentation is to provide information how to install and administer the Augmented Reality Generic Enabler.

16.2 System Requirements

16.2.1 Hardware Requirements

The Augmented Reality GE should work on any modern Intel X86 compatible computer and high-end Android based mobile devices.

16.2.2 Operating System Support

The Augmented Reality GE has been tested against the following Operating Systems:

```
Windows
  Windows 8 (Desktop)
Android
  version 4.2.2
Ubuntu
  version 12.04
```

16.2.3 Software Requirements

```
Browser
   Mozilla Firefox 25.0a1 or higher
```

16.3 Software Installation and Configuration

The Augmented Reality GE is a collections of JavaScript source files which have to be included in a web application.

Latest version can be cloned from the repository at https://github.com/Chiru/FIWARE-AugmentedReality.
The repository encloses a demo directory, which contains example applications that show how the needed source files are included. More Detailed information how to use the GE can be found in the Augmented Reality - User and Programmers Guide.

16.4 Sanity check procedures

16.4.1 End to End testing

One can test Augmented Reality GE by opening any of the demo or/and test files from the cloned repository, with Mozilla Firefox web browser. Just navigate to the directory where the repository is

cloned and start a web server (e.g *python SimpleHTTPServer*, or *node.js http-server*). After that, open Firefox browser and navigate to the address where the server is running (e.g. localhost:8000 if it is running on the same device) and click either the Demos or Tests folder. It is recommended to try demos first.

Demos
The demo folder includes four demos in subfolders *AR_POI*, *house*, *markerDetection* and *plane*. Each subfolder has an index.xhtml file, which starts the demo.
Tests
The tests folder includes two test suit files, *Location_based_regAndTrack_unitTest.html* and *Vision_based_regAndTrack_unitTest.html*.

16.4.2 List of Running Processes

N/A

16.4.3 Network interfaces Up & Open

N/A

16.4.4 Databases

N/A

16.5 Diagnosis Procedures

16.5.1 Resource availability

N/A

16.5.2 Remote Service Access

N/A

16.5.3 Resource consumption

The amount of resources used by the Augmented Reality GE vary a lot depending on the AR application, RAM in respect of the 3D content in a virtual scene and CPU in respect of marker tracking and rendering.

Example of resource consumption: Amount of resources used in tracking markers at 25 fps speed using the following setup. HP EliteBook 2760p, CPU: Intel i5-2540M, 4 GB RAM, GPU: Intel HD Graphics 3000, 1,6 GB RAM, Operating system: windows 8, Browser: Mozilla Firefox 28.0a1

Device	CPU usage	Ram usage
HP EliteBook 2760p	38%	200 MB

16.5.4 I/O flows

N/A

17 Real Virtual Interaction - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

17.1 Introduction

This guide if for installing and administering Real Virtual Interaction Server as described in the Open Specification. Although this guide is mainly for Real Virtual Interaction Backend software, we have included some instructions about how to get the client application working as well.

17.2 System Requirements

Minumum requirements follow official Java SE 1.7 requirements. In excess the real virtual backend requires around 300KB of hard disk space + space for databases (2KB upwards depending on amount of sensor data). With one android application feeding sensor data the estimated size of data base should not arise over 1MB. Real virtual backend requires about heap 50MB.

17.2.1 Hardware Requirements

Hardware with Windows and MacOS X

- RAM: 128 MB; 64 MB for Windows XP (32-bit)
- Disk space: 124 MB

Hardware with Linux:

- RAM: 64 MB
- Disk space: 58 MB

17.2.2 Operating System Support

Windows:

- Windows 8 (Desktop)
- Windows 7
- Windows Vista SP2
- Windows XP SP3 (32-bit); Windows XP SP2 (64-bit)
- Windows Server 2008
- Windows Server 2012 (64-bit)

Mac OS X:

- Intel-based Mac running Mac OS X 10.7.3 (Lion) or later.
- Administrator privileges for installation

Linux:

- Oracle Linux 5.5+
- Oracle Linux 6.x (32-bit)*, 6.x (64-bit)**
- Red Hat Enterprise Linux 5.5+, 6.x (32-bit)*, 6.x (64-bit)**
- Ubuntu Linux* 10.04 and above

- Suse Linux Enterprise Server* 10 SP2, 11.x

17.2.3 Software Requirements

17.2.3.1 *Client Side*

Browsers with Windows OS:

- Firefox 3.6 and above,
- Chrome

Browser with Mac OS X:

- 64-bit browser
- A 64-bit browser (Safari or Firefox, for example) is required to run Java 7 on Mac OS X. 32-bit browsers such as Chrome do not support Java 7 on the Mac platform.

Browsers with Linux:

- All OS that support versions Firefox 3.6 and above

17.2.3.2 *Real Virtual Interaction Server*

The binary file contains dependencies, but if you are build from the source code then you need to acquire the following Dependencies:

- json-simple-1.1.1.jar
- commons-codec-1.8.jar
- JRE 1.7

17.2.3.3 *Publish/Subscribe Web Client*

The .html file contains the neccessary source code and can be deployed locally or on remote server. There are following JavaScipt files as dependencies:

```
 <link href="css/eggplant/jquery-ui-1.10.3.custom.css"
rel="stylesheet">
 <script src="js/jquery-1.9.1.js"></script>
 <script src="js/jquery-ui-1.10.3.custom.js"></script>
 <script src="js/three.min.js"></script>
```

You can manually download them from following links and place them on the server .

- jquery-1.9.1.js
- jquery-ui-1.10.3.custom.css
- jquery-ui-1.10.3.custom.js
- three.min.js

17.3 Software Installation and Configuration

In order to change the configuration of the real virtual interaction backend, you will need to make your own build. All server configurations can be changed from StaticResources.java class file. Please refer to software requirements chapter for dependencies. Any Java supported IDE is fine for importing the source code. We recommend Eclipse IDE. Egit is a good plugin for Eclipse and allows importing projects directly from the GitHub. Before this you should register an account to

GitHub.com and also follow instructions on how to install git on to your machine. In Eclipse you can just choose File->Import.. and then scroll to Git folder and choose "Projects from Git" and press "next". Then choose "URI" and press "next". If you have copied the github repository link (which is for real virtual interaction backend repo: https://github.com/Cyberlightning/WeX-RealVirtual-Backend.git) to clip board, the information should automatically be pasted to required fields. After this choose "master" branch and clone to project. In the last screen you can use new project wizard to import the project as a Java project. After these steps you can simply choose run or debug to test the import.

Otherwise you can deploy the binary .jar file to any server. You can use FTP software like Filezilla to upload the binary file to desired location. You can start the server in Linux for instance:

```
$ screen java -jar realvirtualinteractionbackend.jar
```

Type Ctrl+A+D to detach screen and leave it running.

17.4 Sanity Checks

17.4.1 End to End testing

To ensure that the real virtual interaction backend functions as required, the administrator can run a simulation test by starting the binary .jar file with following parameters:

```
$ java -jar realvirtualinteractionbackend.jar -simulate
```

The simulate parameter will launch a test routine that will simulate a connected device and sends a test packet every 3 seconds to the UDP port of the server. This data will be stored in the database and will be available for subscription/publishing as well as request response. Below example is a simulation run in Windows through console:

```
Microsoft Windows [Version 6.0.6001]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\java -jar realvirtualbackend.jar -simulate
Test Packet send to 10.20.217.101:61616
Test Packet send to 10.20.217.101:61616
Test Packet send to 10.20.217.101:61616
Serialized data is saved in deviceDB.ser
Serialized data is saved in BaseStationRefernceDB.ser
Test Packet send to 10.20.217.101:61616
```

When the simulation is running you can connect to the server with the PubSubClient or use ReqResClient to test whether the server is accessible by remote clients. This is also a fast way to test those clients. You can also send messages to the simulator by using "d23c058698435eff" as UUID. If on a localhost, a sample query could be:

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: fi-fi,fi;q=0.8,en-us;q=0.5,en;q=0.3
Connection: keep-alive
Host: 127.0.0.1:44446
```

```
User-Agent: Mozilla/5.0 (Windows NT 6.0; rv:25.0) Gecko/20100101
Firefox/25.0

Content-Length: 81

Content-Type: application/x-www-form-urlencoded


action=update&device_id=d23c058698435eff&sensor_id=test&parameter=test&
value=test
```

The server should print the following in console if the message is being received:

```
Packet received from
server:action=update&device_id=d23c058698435eff&sensor_id=test&paramete
r=test&value=test
```

You can use the HttpQueryTester.html included in the html folder of the real virtual interaction backend source code to test it. Just make sure that you type in localhost address (127.0.0.1).

17.4.2 List of Running Processes

The real virtual interaction backend will start one process with Java. You can for instance in Linux use

```
top -U <user>
```

The process will show as "java" process run by the user.

17.4.3 Network interfaces Up & Open

- UDP port: 61616 (by default)
- UDP port: 61617 (only needed for testing and is by default one greater than set UDP port)
- TCP port: 44445 (by default)
- TCP port: 44446 (by default)

17.4.4 Databases

The real virtual interaction backend will create the database files if they do not exist. There is no need for any SQL solution currently.

Default names for data base files:

- DeviceDB.ser
- BaseStationReferenceDB.ser

17.5 Diagnosis Procedures

17.5.1 Resource availability

The GE is rather lightweight and is capable in running on a decent hardware with 1-2GB of RAM. Amount of harddisk is dedicated for serializing incoming sensor traffic hence a few GB is completely ok for starters. The complete scaling of these attributes comes from the real amount of events the service needs to be able to handle.

17.5.2 Remote Service Access

Using a SSH enabled console to check the status of server. Otherwise NO external administration interface is being provided along with the deliverable.

17.5.3 Resource consumption

The CPU load should be around 0-1% when there is not I/O events on server. Temporarily the server may use more CPU but if there is a stable 100% there is likely something wrong and server should be restarted. In linux use following command to check CPU/Memory consumption:

```
top -U <user>
```

17.5.4 I/O flows

- UDP port 61616 (by default) does not accept larger than 1024 byte UDP packets. This is because larger than 1024 bytes may not be supported network MTU. The payload may be compressed using Gzip to increase amount of information. There can be a mechanism developed to include data being divided in to multiple packets, but due to nature of UDP traffic this is not recommended. Rather divide data into separate packets from device end.

18 Virtual Characters - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

18.1 Introduction

This document describes the installation & administration requirements for the Virtual Characters GE.

18.2 System Requirements

18.2.1 Hardware Requirements

A GPU that supports WebGL vertex and pixel shaders is required.

18.2.2 Operating System Support

This GE is operating system independent, as it's JavaScript code running in a browser.

18.2.3 Software Requirements

A web browser that supports WebGL rendering is required, such as

- Firefox 4.0+
- Chrome 8.0+
- Opera 15.0+
- Opera Mobile 12.0+
- Chrome for Android 31.0+
- Firefox for Android 25.0+

For serving the client HTML and JavaScript code, an HTTP server such as Apache can be used. For testing the browser can also be pointed to local files.

18.3 Software Installation and Configuration

The Virtual Characters GE is delivered as part of the WebTundra JavaScript libraries, which also implement the scene model, rendering and network synchronization. It depends on the scene model and rendering code but only optionally of the networking code.

There are two options for getting it:

- Download a packaged release from https://forge.fi-ware.org/frs/download.php/1151/MIWI-VirtualCharacters-3.3.3.zip
- Clone the git repository at https://github.com/realXtend/WebTundra

The JavaScript code can be run from local files for testing, ie. it does not need to be served from an actual HTTP server. Note that several examples in the WebTundra codebase require a Tundra server to be running for network synchronization; these belong to the 3D-UI and Synchronization GE's and not to the Virtual Characters GE, which itself does not require client-server networking.

18.4 Sanity check procedures

18.4.1 End to End testing

To test that the necessary functionality is implemented by the browser, run the following HTML file from the WebTundra directory structure: examples/Avatar/index.html. You should see a robot character with attached clothes and swords playing a running animation. On a browser with lacking WebGL support (such as Internet Explorer 10) you will see a blank page.

18.4.2 List of Running Processes

As the Virtual Character GE is client-only, typically the only process that is running is the web browser, for example Firefox or Chrome.

18.4.3 Network interfaces Up & Open

N/A

18.4.4 Databases

N/A

18.5 Diagnosis Procedures

The most typical failure is that on browsers that do not support WebGL or do not have WebGL enabled a 3D web application or library such as the Virtual Characters GE will fail to run at all. Therefore check WebGL support first.

18.5.1 Resource availability

Exact amounts of resources required depend on the scene that is being run, but rough estimation is that 1 GB of system RAM and 1 GB of hard disk space should be available. Considerably less is required for simple scenes. For example, the examples/Avatar/index.html test scene consumes 160 MB memory when run on a Firefox browser on Windows, and 5-10% CPU usage on an eight-core, 3 GHz Intel Core-i7 system. There should be no disk I/O after the scene has loaded.

18.5.2 Remote Service Access

N/A

18.5.3 Resource consumption

Because resource consuption (memory and CPU) depends on the scene being run, absolute abnormal values are not possible to give. Note the amount of memory use by the browser process once the scene has started/loaded, and see whether it starts to grow rapidly, for example doubles in a few seconds. That case would indicate a likely memory leak within the GE or the scene/application using it.

18.5.4 I/O flows

N/A

19 Interface Designer - Installation and Administration Guide

You can find the content of this chapter as well in the wiki of fi-ware.

19.1 Introduction

This document describes the installation and administration of the Interface Designer GE.

19.2 System Requirements

19.2.1 Hardware Requirements

Since this GE is application-oriented, and serves for manipulating a 3D scene, any recent PC/Mac configuration that is capable of 3D rendering is considered as a minimum system requirement.

19.2.2 Operating System Support

Any operating system that is supported by the target WebGL-enabled browser.

19.2.3 Software Requirements

The interface designer GE is currently implemented specifically for Web Rocket / Web Tundra and XML3D scenes.

A WebGL-enabled browser, which includes, but not limited to:

- Google Chrome 9+
- Mozilla Firefox 4.0+
- Safari 6.0+ (WebGL disabled by default)
- Opera 11+ (WebGL disabled by default)

Other internal dependencies are

- Classy.js v1.4 (https://github.com/mitsuhiko/classy/releases/tag/1.4)
- jQuery v2.0.3 (https://jquery.org/)
- jQuery UI v1.10.3 (http://jqueryui.com/)
- Fancytree v.2.0.0-5 (http://plugins.jquery.com/fancytree/)
- jquery-contextMenu v1.7 (https://github.com/arnklint/jquery-contextMenu)

19.3 Software Installation and Configuration

- First, download all dependencies from the links above, and place them in, for example, a folder named "lib" that is on the same directory level with your main HTML file. Add the scripts into the <head> section of your HTML file by doing:

```
<script type="text/javascript" src="lib/classy.js"></script>
<script type="text/javascript" src="lib/jquery.js"></script>
<script type="text/javascript" src="lib/jquery-ui.js"></script>
```

```
    <script type="text/javascript" src="lib/jquery.fancytree-
all.min.js"></script>
    <script type="text/javascript"
src="lib/jquery.contextmenu.js"></script>
```

- Add the following style sheets that come from JQuery UI and Fancytree dependencies also in the <head> section of your HTML file by doing:

```
    <link rel="stylesheet"
href="http://code.jquery.com/ui/1.10.3/themes/smoothness/jquery-
ui.css">
    <link rel="stylesheet" href="lib/skin-win8/ui.fancytree.css">
```

- Finally, clone the GIT repository https://github.com/Adminotech/fiware-interface-designer. The directory structure is as follows:
    - resources/
        - ball.xml
        - cone.xml
        - cube.xml
        - cylinder.xml
    - lib/
        - three-TransformControls.js
    - InterfaceDesigner-main.js
    - XML3DEditor.js
    - RocketEditor.webrocketjs

19.3.1 WebRocket

- Add the RocketEditor.webrocketjs script to src/applications in the WebRocket code tree;
- Add InterfaceDesigner-main.js and lib/three-TransformControls.js into src/lib

19.3.2 XML3D

- Download and add xml3d.js from (http://www.xml3d.org/xml3d/script/xml3d.js) to your "lib" folder, and add it BEFORE all other scripts in the <head> section.

```
<script type="text/javascript" src="lib/xml3d.js"></script>
```

- Add XML3DEditor.js and InterfaceDesigner-main.js into your HTML file as follows:

```
    <script type="text/javascript" src="lib/InterfaceDesigner-
main.js"></script>
    <script type="text/javascript" src="lib/XML3DEditor.js"></script>
```

- Copy the "resources" folder into the same directory as your main HTML file. If you want other location, make sure you give the relative path to it in the XML3DEditor constructor (third argument, see below).
- While in the <head> section, add another <script> tag, and add an event listener to the "load" event on the "document" object, to ensure that all the dependencies are loaded before instantiating the editor. Inside the anonymous method, instantiate the editor. The constructor has two required and one optional arguments which are id-s to the main

element and the main <xml3d> element. For best results, do a scene that has a with a <xml3d> DOM tree as a child to said .

```
    <script type="text/javascript">
        document.addEventListener('load', function() {
            var editor = new XML3DEditor("id-of-main-DIV", "id-of-main-
XML3D-element", "path_to_resources");
        }, false);
    </script>
```

- If you do not have your own camera system in your XML3D scene, you can add one by first downloading the xml3d-motion.js javascript file to your "lib" directory (https://raw.githubusercontent.com/xml3d/xml3d-examples/master/script/xml3d-motion.js) and adding it as the rest of the dependencies:

```
    <script type="text/javascript" src="lib/xml3d-motion.js"></script>
```

Then, add the following code inside the "load" event handler, before the instantiation of XML3DEditor:

```
    var cameraController = new XMOT.ExamineController(
    document.getElementById("mainCamera"), { "sceneRadius" : 8 });
    var xml3d = document.getElementById("mainCanvas");
    xml3d.addEventListener("mousedown", function(e) {
        cameraController.start({
            x: e.clientX,
            y: e.clientY
        }, e.button == 2 ? XMOT.ExamineController.DOLLY :
XMOT.ExamineController.ROTATE);
        e.preventDefault();
    }, false);
    window.addEventListener("mouseup", function() {
        cameraController.stop();
    }, false);
    window.addEventListener("mousemove", function(e) {
        cameraController.doAction({
            x: e.clientX,
            y: e.clientY
        });
    }, false);
```

- Default shortcuts for toggling the editor (turning it on or off) is "shift" + "s", and switching between Scene Tree editor and EC editor panels is "shift" + "e". You can override these shortcuts by adding:

```
     editor.setToggleEditorShortcut(metaKey, key1);
     editor.setSwitchPanelsShortcut(metaKey, key2);
```

Accepted values for metaKey are "shift", "ctrl", "meta", and "alt", and key1 and key2 could be any character on the keyboard.

- As a guide, this is an example of how your HTML file should look after following the steps above, assuming that you want change the default shortcuts, and you need a camera setup:

```
<!doctype html>
 <html lang="en">
 <head>
    <link rel="stylesheet"
href="http://code.jquery.com/ui/1.10.3/themes/smoothness/jquery-
ui.css">
    <link rel="stylesheet" href="lib/skin-win8/ui.fancytree.css">


    <script type="text/javascript" src="lib/xml3d.js"></script>
    <script type="text/javascript" src="lib/jquery.js"></script>
    <script type="text/javascript" src="lib/jquery-ui.js"></script>
    <script type="text/javascript" src="lib/jquery.fancytree-
all.min.js"></script>
    <script type="text/javascript"
src="lib/jquery.contextmenu.js"></script>
    <script type="text/javascript" src="lib/classy.js"></script>
    <script type="text/javascript" src="lib/InterfaceDesigner-
main.js"></script>
    <script type="text/javascript" src="lib/XML3DEditor.js"></script>


    <script type="text/javascript" src="lib/xml3d-motion.js"></script>


    <script type="text/javascript">
        window.addEventListener('load', function() {


            <!-- Camera control -->
            var cameraController = new XMOT.ExamineController(
                document.getElementById("mainCamera"), { "sceneRadius"
: 8 });
            var xml3d = document.getElementById("mainCanvas");


            xml3d.addEventListener("mousedown", function(e) {
                cameraController.start({
                    x: e.clientX,
                    y: e.clientY
```

```
                }, e.button == 2 ? XMOT.ExamineController.DOLLY :
XMOT.ExamineController.ROTATE);
                e.preventDefault();
            }, false);

            window.addEventListener("mouseup", function() {
                cameraController.stop();
            }, false);

            window.addEventListener("mousemove", function(e) {
                cameraController.doAction({
                    x: e.clientX,
                    y: e.clientY
                });
            }, false);

            <!-- XML3D editor instance -->
            var editor = new XML3DEditor("mainContent", "mainCanvas",
"resources/");
            editor.setToggleEditorShortcut("shift", "a");
            editor.setSwitchPanelsShortcut("shift", "d");
        }, false);
    </script>
 </head>
 <body>
    <div id="mainContent" style="width:100%; height:100%">
        <xml3d id="mainCanvas" activeView="#mainCameraShape"
class="xml3d" style="width: 100%;height:100%;background-color:white;">
            <defs>
                <transform id="mainCameraTransform" rotation="-
0.5051031 0.83705676 0.21025412 0.92295426" scale="1.0 1.0 1.0"
translation="16.4985 11.1548 18.486"></transform>
            </defs>
            <group id="mainCamera" transform="#mainCameraTransform">
                <view fieldOfView="0.66059315"
id="mainCameraShape"></view>
            </group>
        </xml3d>
    </div>
 </body>
```

```
</html>
```

19.4 Sanity check procedures

19.4.1 End to End testing

As a verification that everything was installed correctly, pressing the shortcut "Shift" + "S" (or the appropriate meta + key combination if overridden as described above) should open up the scene editor, and should be ready to use.

19.4.2 List of Running Processes

N/A

19.4.3 Network interfaces Up & Open

N/A

19.4.4 Databases

N/A

19.5 Diagnosis Procedures

N/A

19.5.1 Resource availability

N/A

19.5.2 Remote Service Access

N/A

19.5.3 Resource consumption

N/A

19.5.4 I/O flows

N/A