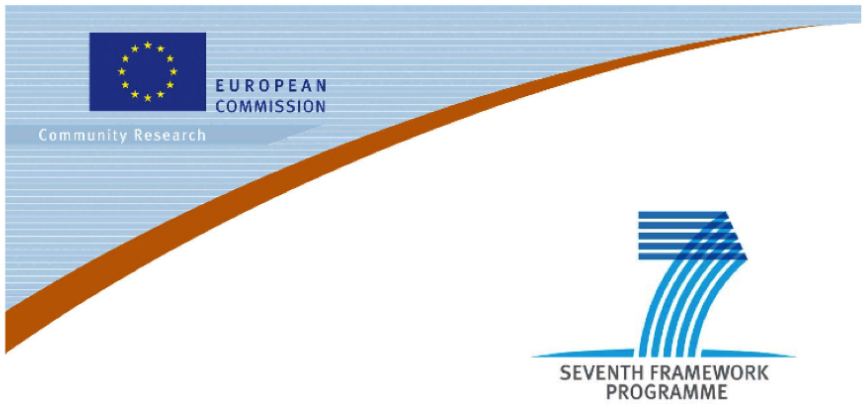

Private Public Partnership Project (PPP)

Large-scale Integrated Project (IP)

Cloud Rendering - Installation and Administration Guide

Project full title: Future Internet Core
Project acronym: FI-Core
Contract No.: 632893

Contents

Cloud Rendering - Installation and Administration Guide

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.

System Requirements

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/

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)

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.

Software Installation and Configuration

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

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
```

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 developmen
    t 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.

Renderer

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 pre-build 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.

Using prebuilt binaries

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.

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/

Sanity check procedures

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/

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.

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.

Database

N/A

Diagnosis Procedures

Resource availability

Web Service

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

Renderer

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

Remote Service Access

N/A

Resource consumption

Web Service

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

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.

I/O flows

Web Service

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

Renderer

- TCP and UDP ports 2345-2350 should be open