

Private Public Partnership Project (PPP)

Large-scale Integrated Project (IP)



D.6.3.3: 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-WP6-D.6.3.3

Project Document Date: 2014-07-09

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 "Data/Context Management Services" 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.3), 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 "Data/Context Management" FI-WARE chapter aims at providing outperforming and platform-like GEs that ease development and provision of innovative Applications that require management, processing and exploitation of context information as well as data streams in real-time and at massive scale. Combined with enablers coming from the [Applications/Services Ecosystem and Delivery](#) chapters, application providers will be able to build innovative business models such as the ones described above and beyond.

FI-WARE Data/Context Management GEs enables to:

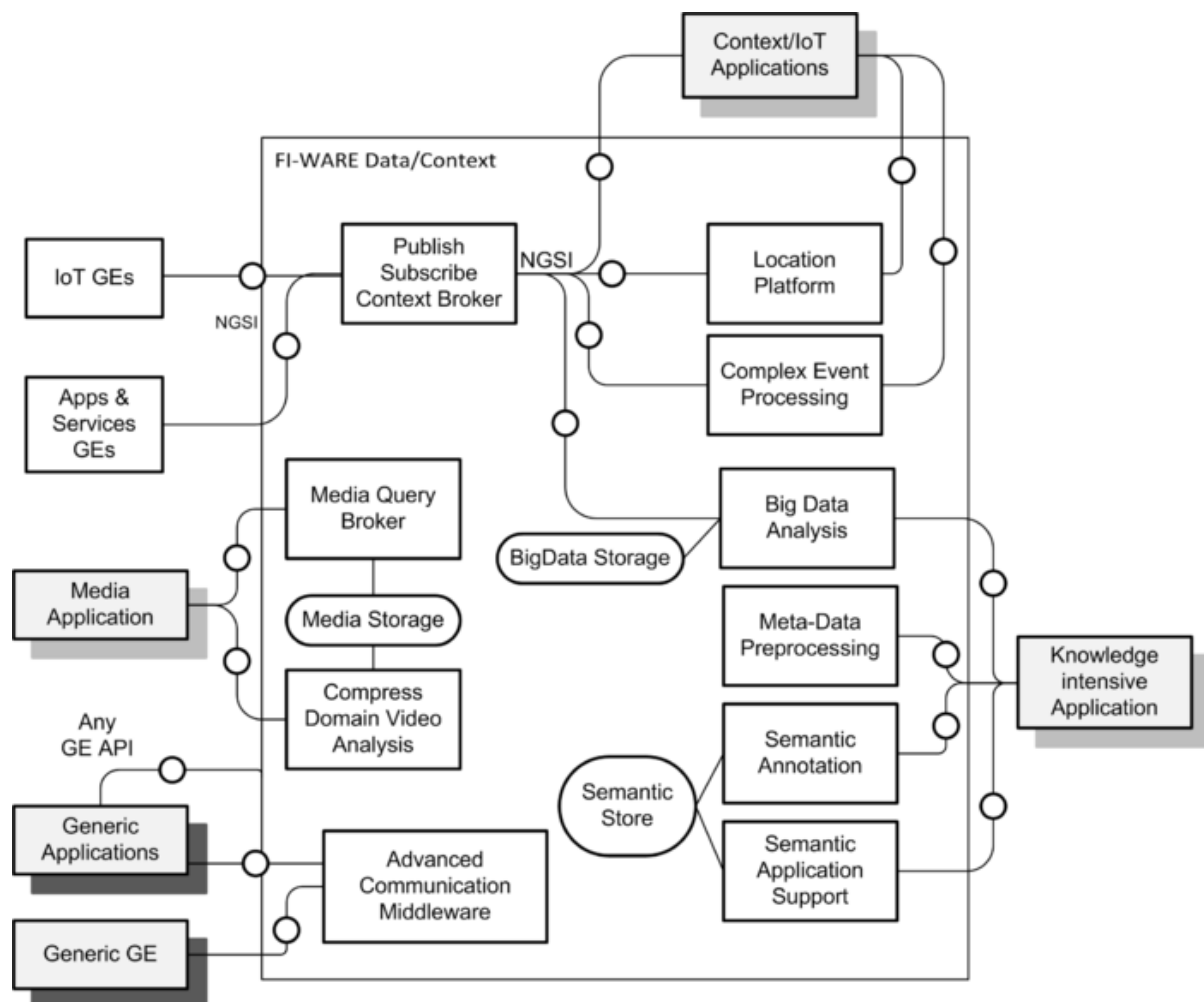
- Record, subscribe for being notified about and query for context information coming from different sources.
- Model changes in context as events that can be processed to detect complex situations that will lead to generation of actions or the generation of new context information (therefore, also treatable as events).
- Processing large amounts of context information in an aggregated way, using map&reduce techniques, in order to generate knowledge that may also lead to execution of actions and/or creation of new context information.
- Process data streams (particularly, multimedia video streams) coming from different sources in order to generate new data streams as well as context information that can be further exploited.
- Process metadata that may be linked to context information, using standard semantic support technologies.

- Manage some context information, such Location information, in a standardized way.

A cornerstone concept within this chapter is the structural definition of Data Elements enclosing its "Data Type", a number of "Data Element attributes" (which enclose the following: Name, Type, Value) and, optionally, a set of "Metadata Elements" (which have also in turn Data-like attributes: Name, Type, Value). However, this precise definition remains unbound to any specific type of representation and is able to represent "Context Elements" and "Events" as "Data Element" structures. More comprehensive information is available at Fi-WARE Data/Context Chapter vision.

"Data" in FI-WARE refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. A cornerstone concept in FI-WARE is that data elements are not bound to a specific format representation.

The following diagram shows the main components (Generic Enablers) that comprise the third release of FI-WARE Data/Context chapter architecture.



More information about the Data Chapter and FI-WARE in general can be found within the following pages:

<http://wiki.fi-ware.eu>

[Data/Context Management Architecture](#)

[Materializing Data/Context Management 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.eu/>

The following resources were used to generate this document:

D.6.3.3_Installation_and_Administration_Guide_front_page

[Publish/Subscribe Broker - Orion Context Broker - Installation and Administration Guide](#)

[Publish/Subscribe GE - Context Awareness Platform - Installation and Administration Guide](#)

[Publish/Subscribe Semantic Extension - Installation and Administration Guide](#)

[CEP GE - IBM Proactive Technology Online Installation and Administration Guide](#)

[BigData Analysis - Installation and Administration Guide](#)

[Compressed Domain Video Analysis - Installation and Administration Guide](#)

[Unstructured Data Analysis - Installation and Administration Guide](#)

[Metadata Preprocessing - Installation and Administration Guide](#)

[LOCS - Installation and Administration Guide](#)

[Query Broker - Installation and Administration Guide](#)

[Semantic Application Support - Installation and Administration Guide](#)

[StreamOriented - 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.

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, es.Internet, Latin American Platforms, Cloud Edge, Cloud Proxy.

1.9 Changes History

Release	Major changes description	Date	Editor
v1	First Version	2014-07-09	TID

1.10 Table of Contents

1.1	Executive Summary.....	2
1.2	About This Document	3
1.3	Intended Audience.....	3
1.4	Chapter Context.....	3
1.5	Structure of this Document	5
1.6	Typographical Conventions.....	5
1.7	Acknowledgements.....	6
1.8	Keyword list.....	6
1.9	Changes History	7
1.10	Table of Contents.....	7
2	Publish/Subscribe Broker - Orion Context Broker - Installation and Administration Guide.....	10
2.1	Introduction	10
2.2	Installation	10
2.3	Running Orion Context Broker.....	12
2.4	Administration Procedures	16
2.5	Database administration.....	23
2.6	Sanity check procedures	35
2.7	Diagnosis Procedures.....	36
3	Publish/Subscribe GE - Context Awareness Platform - Installation and Administration Guide	43
3.1	Description of Context Awareness Platform implementing Publish/Subscribe Context Broker	43
3.2	Software architecture	43
3.3	Deployment architecture.....	44
3.4	Sanity check procedures	53
3.5	Diagnosis Procedures.....	54
4	Publish/Subscribe Semantic Extension - Installation and Administration Guide	56
4.2	Sanity check procedures	62
4.3	Diagnosis Procedures.....	67

5	CEP GE - IBM Proactive Technology Online Installation and Administration Guide.....	68
5.1	CEP GE	68
5.2	Sanity check procedures	71
5.3	Diagnosis Procedures	73
6	BigData Analysis - Installation and Administration Guide	74
6.1	Introduction	74
6.2	Platform installation	74
6.3	Cygnus installation	87
6.4	Sanity check procedures	91
6.5	Diagnosis procedures	103
7	Compressed Domain Video Analysis - Installation and Administration Guide	105
7.1	Prerequisites	105
7.2	Installation	105
7.3	Configuration	106
7.4	Sanity Check Procedures.....	106
7.5	Diagnosis Procedures	108
7.6	References	108
8	Unstructured Data Analysis - Installation and Administration Guide.....	109
8.1	Introduction	109
8.2	System Requirements	109
8.3	Installation guidelines	109
8.4	Sanity check procedures	125
8.5	Diagnosis Procedures	126
9	Metadata Preprocessing - Installation and Administration Guide	127
9.1	Prerequisites	127
9.2	Installation	127
9.3	Configuration	128
9.4	Sanity Check Procedures.....	128
9.5	Diagnosis Procedures	129
9.6	References	130
10	LOCS - Installation and Administration Guide.....	131
10.1	Introduction	131
10.2	Installation and Preparation	131

10.3	Starting Location GE.....	135
10.4	Stopping Location GE	136
10.5	Logging	136
10.6	REST Interface configuration	137
10.7	MLP Interface configuration	137
10.8	Sanity check procedures	137
10.9	Diagnosis Procedures.....	138
11	Query Broker - Installation and Administration Guide	140
11.1	Introduction	140
11.2	System Requirements	140
11.3	Configuration	141
11.4	Sanity check procedures	167
11.5	Diagnosis Procedures.....	177
12	Semantic Application Support - Installation and Administration Guide.....	178
12.1	Introduction	178
12.2	System Requirements	178
12.3	Installation guidelines.....	178
12.4	Sanity check procedures	190
12.5	Diagnosis Procedures.....	192
13	StreamOriented - Installation and Administration Guide.....	193
13.1	Kurento Installation	193
13.2	Sanity check procedures	205
13.3	Diagnosis Procedures.....	208

2 Publish/Subscribe Broker - Orion Context Broker - Installation and Administration Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

2.1 Introduction

This Installation and Administration Guide covers Orion Context Broker versions since 0.5.0 (corresponding to FI-WARE release 2.3.3). Please, pay attention to the "Release Note" fragments along the document, as they contain important information regarding particular versions/releases.

Any feedback on this document is highly welcome, including bugs, typos or things you think should be included but aren't. Please send it to the "Contact Person" email that appears in the [Catalogue page for this GEi](#). Thanks in advance!

2.2 Installation

This section defines the procedure to install the Orion Context Broker, including its requirements.

2.2.1 Requirements

- System resources: see [these recommendations](#)
- Operating system: CentOS/RedHat. The reference operating system is CentOS 6.3 but it should work also in any later CentOS/RedHat version.
- Database: MongoDB is required running either in the same system where Orion Context Broker is going to be installed or in a different host accessible through the network. The recommended MongoDB version are:
 - For Orion versions previous to 0.11.0, the recommended version is 2.2.3 (although it should work with later versions).
 - For Orion 0.11.0 and newer, the recommended version is 2.4.9 (although it should work with later versions). It is possible to run Orion 0.11.0 with 2.2.x, but it is not recommended, as you cannot take advantages of the [geolocation capabilities](#). Check [database upgrade procedures section](#) if you are upgrading from MongoDB 2.2 to 2.4.
- RPM dependencies (some of these packages could not be in the official CentOS/RedHat repository but in EPEL, in which case you have to configure EPEL repositories, see <http://fedoraproject.org/wiki/EPEL>):
 - The contextBroker package (mandatory) depends on the following packages: boost-filesystem, boost-thread, libmicrohttpd-devel (**new requirement in 0.8.0**) and logrotate (**new requirement in 0.13.0**)
 - The contextBroker-test package (optional but highly recommendable) depends on the following packages: python, python-flask, python-jinja2, curl, libxml2, libxslt, nc, mongo-

10gen and contextBroker. The mongo-10gen dependency needs to configure MongoDB repository, check [this piece of documentation about that](#).

- Version 0.12.0 and before: they used mongodbd (from EPEL) instead of mongo-10gen as dependency

2.2.2 Downloading

The RPM packages corresponding to the Orion Context Broker can be found in the [FI-WARE Files area](#). Look for the "DATA&IOT-OrionContextBroker" entry.

Note that versions previous to 0.8.1 (i.e. 0.8.0 and before) are located in [FI-WARE PPP Restricted Files area](#) (available to PPP members only).

2.2.3 Installing

The procedure consists in installing it using the rpm command, e.g:

```
rpm -i contextBroker-0.5.0-1.x86_64.rpm
```

Optionally, if you have the RPM in a yum repository ([as the one provide by FI-WARE](#)) then you can install doing:

```
yum install contextBroker
```

(Sometimes the above commands fails due to yum cache. In that case, run "yum clean all" and try again)

Note that the above commands demand superuser privileges, so you have to run them as root or using the sudo command.

2.2.4 Upgrading

If you have already installed a previous version of the Orion Context Broker, then upgrade using the following command:

```
rpm -U contextBroker-0.5.0-1.x86_64.rpm
```

Optionally, if you have the RPM in a yum repository ([as the one provide by FI-WARE](#)) then you can upgrade doing:

```
yum install contextBroker
```

(Sometimes the above commands fails due to yum cache. In that case, run "yum clean all" and try again)

Note that the above commands demand superuser privileges, so you have to run them as root or using the sudo command.

2.2.5 Optional packages

Apart from the mandatory RPM described above, it is highly recommended that you install the contextBroker-test package, which contain utility tools used in the [User and Programmers Guide](#) (in particular, the accumulator-server.py used in some steps in that manual) and the testing framework being used for [Unit Testing Plan](#).

```
rpm -i contextBroker-test-0.5.0-1.x86_64.rpm
```

2.2.6 Database upgrade

All the published versions of Orion Context Broker use a data model (described in detail in [the following section](#)) that is backward compatible with previous versions. In other words, upgrading from one version of Orion to a newer version doesn't need any data migration action.

Regarding the database engine itself, note that version 0.11.0 recommends MongoDB 2.4 (while previous versions recommend MongoDB 2.2). Please, check [the 2.4 upgrade procedure in the official MongoDB documentation](#).

2.3 Running Orion Context Broker

Once installed, there are two ways of running Orion Context Broker: manually from the command line or as a system service. It is not recommended to mix both ways (e.g. start the context broker from the command line, then use "/etc/init.d/contextBroker status" to check its status).

2.3.1 From the command line

You can run the broker just typing the following command:

```
# contextBroker
```

The broker will run in background by default, so you will need to stop it [using signals](#).

You can use command line arguments, e.g. to specify the port in which Orion Context Broker listens, using the -port option:

```
# contextBroker -port 5057
```

To know all the possible options, have a look at the [command line options](#) section.

2.3.2 As system service

You will typically need superuser privileges to use Orion Context Broker as a system service, so the following commands need to be run as root or using the sudo command.

In order to start the broker service, run:

```
# /etc/init.d/contextBroker start
```

Then, to stop the context broker, run:

```
# /etc/init.d/contextBroker stop
```

To restart, run:

```
# /etc/init.d/contextBroker restart
```

You can use `chkconfig` command to make contextBroker automatically start/stop when your system boots/shutdowns (see [chkconfig documentation](#) for details).

The configuration used by the contextBroker service is stored in the `/etc/sysconfig/contextBroker` file, which typical content is:

```
# BROKER_USER - What user to run orion-broker as
BROKER_USER=orion

# BROKER_PORT - the port/socket where orion-broker will listen for
connections
BROKER_PORT=1026

# BROKER_LOG_DIR - Where to log to
BROKER_LOG_DIR=/var/log/contextBroker

# BROKER_PID_FILE - Where to store the pid for orion-broker
BROKER_PID_FILE=/var/log/contextBroker/contextBroker.pid

## Database configuration for orion-broker
BROKER_DATABASE_HOST=localhost
BROKER_DATABASE_NAME=orion

# Database authentication (not needed if MongoDB doesn't use --auth)
#BROKER_DATABASE_USER=orion
#BROKER_DATABASE_PASSWORD=orion

# Use the following variable if you need extra command line options
```

```
#BROKER_EXTRA_OPS="-t 0-255"
```

All the fields except BROKER_USER and BROKER_EXTRA_OPS map to *one* of the options described in [command line options](#), as follows:

- BROKER_PORT maps to -port
- BROKER_LOG_DIR maps to -logDir
- BROKER_PID_FILE maps to -pidpath
- BROKER_DATABASE_HOST maps to -dbhost
- BROKER_DATABASE_NAME maps to -db
- BROKER_DATABASE_USER maps to -dbuser
- BROKER_DATABASE_PASSWORD maps to -dbpwd

Regarding BROKER_EXTRA_OPS, it is used to specify other options not covered by the fields above, as a string that is appended to the broker command line at starting time. Note that this string is "blindly" appended, i.e. the service script doesn't do any check so be careful using this, ensuring that you are providing valid options here and you are not duplicating any option in other BROKER_* field (e.g. not set BROKER_EXTRA_OPS="-port 1026" as BROKER_PORT is used for that).

Regarding BROKER_USER, it is the user that will own the contextBroker process upon launching it. By default, the RPM installation creates a user named 'orion'. Note that if you want to run the broker in a privileged port (i.e. 1024 or below) you will need to use 'root' as BROKER_USER.

2.3.3 Command line options

Command line options can be used directly (in the case of running [from the command line](#)) or indirectly through the different fields in /etc/sysconfig/contextBroker (in the case of running [as a system service](#)). To obtain a list of available options, use:

```
# contextBroker -u
```

To get more information (including default values), use:

```
# contextBroker -U
```

The list of available options is the following:

- **-u** and **-U**. Shows usage in brief or long format, respectively.
- **--help**. Show help (very similar to previous).
- **--version**. Shows version number
- **-port <port>**. Specifies the port that the broker listens to. Default port is 1026.
- **-ngsi9**. The broker runs only NGSI9 (NGSI10 is not used) (see [programmers guide](#)).

- **-ipv4**. Runs broker in IPv4 only mode (by default, the broker runs in both IPv4 and IPv6). Cannot be used at the same time that **-ipv6**. **New in 0.10.0:** previous versions run in IPv4 only mode always.
- **-ipv6**. Runs broker in IPv6 only mode (by default, the broker runs in both IPv4 and IPv6). Cannot be used at the same time that **-ipv4**. **New in 0.10.0:** previous versions run in IPv4 only mode always.
- **-rush <host:port>**. Use **rush** in *host* and *port*. Default behavior is to *not* use Rush. See section on [using Rush relay](#). (**Since 0.13.0**)
- **-multiservice <off|url|header>**. Enables multiservice/multitenant mode (see [multi service tenant section in the users and programmers guide](#)). Default value is **off**. (**Since 0.13.0**)
- **-db <db>**. The MongoDB database to use or (**since 0.13.0** if **-multiservice** is in use) the prefix to per-service/tenant databases (see section on [service/tenant database separation](#) later in this manual).
- **-dbhost <host>**. The MongoDB host and port to use, e.g. "-dbhost localhost:12345"
- **-dbuser <user>**. The MongoDB user to use. If your MongoDB doesn't use authorization this option must be avoided. See [database authorization section](#).
- **-dbpwd <pass>**. The MongoDB password to use. If your MongoDB doesn't use authorization this option must be avoided. See [database authorization section](#).
- **-https**. Work in secure HTTP mode (See also '-cert' and '-key') (**Since 0.12.0**)
- **-cert**. Certificate file for https (**Since 0.12.0**)
- **-key**. Private server key file for https (**Since 0.12.0**)
- **-logDir <dir>**. Specifies the directory to use for the contextBroker log file.
- **-logAppend**. If used, the log lines are appended to the contextBroker log file, instead of re-creating it when the broker starts.
- **--silent**. Suppress all log output except errors (**Since 0.12.0**)
- **-t <trace level>**. Specifies the initial trace levels for logging. You can use a single value (e.g. "-t 70"), a range (e.g. "-t 20-80"), a comma-separated list (e.g. "-t 70,90") or a combination of them (e.g. "-t 60,80-90"). If you want to use all trace levels for logging, use "-t 0-255". Note that trace levels can be changed dynamically using the [management REST interface](#). The detail of the available tracelevels and its numbers can be found [here](#) (as a C struct).
- **-v, -vv, -vvv, -vvvv, -vvvvv**. Specifies the verbosity level, from less verbose ("-v") to more verbose ("-vvvvv"). Note that the verbosity level can be changed dynamically using the [management REST interface](#).
 - **Release Note (0.5.0 FIWARE 2.3.3):** although you can set the verbosity level using this option, probably we don't appreciate too much actual difference in the logging. The adaptation of the different log messages to the verbosity levels is pending for a next release.
- **-fwdHost <host>**. Forwarding host for NGIS9 registerContext when the broker runs in "ConfMan mode" (see [programmers guide](#)).
- **-fwdPort <port>**. Forwarding port for NGIS9 registerContext when the broker runs in "ConfMan mode" (see [programmers guide](#)).
- **-fg**. Runs broker in foreground (useful for debugging)
- **-localIp <ip>**. Specifies in which IP interface the broker listens to. By default it listens to all the interfaces.

- **-pidpath <pid_file>**. Specifies the file to store the PID of the broker process.

Release Note (0.8.1 FIWARE 3.2.1): old versions used also a -reset switch to clean database, but it was removed due to its danger, in favor of a [administration procedure](#) (very easy to do, by the way).

2.3.4 Log file

The log file of the contextBroker upto release 0.14.0 is /tmp/contextBrokerLog, and from 0.14.1 on, the log file is /tmp/contextBroker.log. Remember that the directory where the log file is stored (/tmp by default) can be changed using the --logDir command line option.

When starting the Orion context broker, if a previous log file exist, then it is renamed, appending the text ".old" to its name.

2.3.5 Using Rush relayer

Since 0.13.0

Apart from running Orion Context Broker in "stand alone" mode, you can also take advantage of Rush (<https://github.com/telefonicaid/Rush>) as notification relayer. Thus, instead of managing the notifications itself (including waiting for the HTTP timeout while the notification receives responses), Orion passes the notification to Rush, which in turn deals with it. Thus, Orion can implement a "fire and forget" policy for notification sending, relying in Rush (a piece of software highly specialized in that task) for that.

In addition, you can send notifications using HTTPS using Rush (see [security section in Users and Programmers manual](#)).

In order to use Rush you need:

- A running Rush instance network-reachable from Orion, e.g. in the same host and reachable using "localhost". The installation of Rush is out the scope of this manual, please check the [Rush documentation](#) for that.
- Run Orion using the -rush command line interface, which value has to be the Rush host and port, eg. "-rush localhost:1234" means that Rush is listening in port 1234 in localhost.

HTTP Rush relayer is used only for notifications. Other cases in which Orion acts as HTTP client (in particular, the one in which [Orion plays configuration manager role](#)) doesn't use Rush and Orion always sends the HTTP request itself.

2.4 Administration Procedures

2.4.1 Backing up and restoring database

The usual procedures for MongoDB are used.

2.4.1.1 **Backup**

Use mongodump command to get a backup of the Orion Context Broker database. It is strongly recommended that you stop the broker before doing a backup.

```
# mongodump --host <dbhost> --db <db>
```

This will create the backup in the dump/ directory.

2.4.1.2 **Restore**

Use the mongorestore command to restore a previous backup of the Orion Context Broker database. It is strongly recommended that you stop the broker before doing a backup and to remove (drop) the database used by the broker.

Let's assume that the backup is in the dump/<db> directory. To restore it:

```
# mongorestore --host <dbhost> --db <db> dump/<db>
```

2.4.2 Management REST interface

Apart from the NGSI 9/10 interface, Orion Context Broker exposes a REST API for management that allows to change the trace and verbosity levels (which initial values are set using the "-v" to "-vvvvv" and "-t" command line options).

In order to manage the verbosity level:

```
curl --request DELETE <host>:<port>/log/verbose
curl --request GET <host>:<port>/log/verbose
curl --request PUT <host>:<port>/log/verbose/off
curl --request PUT <host>:<port>/log/verbose/[0-5]
```

In order to manage the trace level:

```
curl --request DELETE <host>:<port>/log/trace
curl --request DELETE <host>:<port>/log/trace/t1
curl --request DELETE <host>:<port>/log/trace/t1-t2
curl --request DELETE <host>:<port>/log/trace/t1-t2,t3-t4
curl --request GET <host>:<port>/log/trace
curl --request PUT <host>:<port>/log/trace/t1
curl --request PUT <host>:<port>/log/trace/t1-t2
curl --request PUT <host>:<port>/log/trace/t1-t2,t3-t4
```

'PUT-requests' overwrites the previous log settings. So, in order to ADD a trace level, a GET /log/trace must be issued first and after that the complete trace string to be sent in the PUT request can be assembled.

2.4.3 Checking status

Only available if you are running Orion Context Broker [as a system service](#).

In order to check the status of the broker, use the following command with superuser privileges (using the root user or the sudo command):

```
/etc/init.d/contextBroker status
```

If broker is running you will get:

```
Checking contextBroker... Running
```

If broker is not running you will get:

```
Checking contextBroker... Service not running
```

2.4.4 Log rotation

Since 0.13.0

Logrotate is installed as RPM dependency along with contextBroker and the following files are installed related to logrotate:

- /etc/logrotate.d/logrotate-contextBroker-daily: which enables daily log rotation
- /etc/sysconfig/logrotate-contextBroker-size: in addition to the previous rotation, this file ensures log rotation if the log file grows beyond a given threshold (100 MB by default)
- /etc/cron.d/cron-logrotate-contextBroker-size: which ensures the execution of etc/sysconfig/logrotate-contextBroker-size at a regular frequency (default is 30 minutes).

2.4.4.1 *Manual procedure*

This procedure is obsolete since 0.13.0. Take it into account only if you use Orion 0.12.0 or before.

The Orion Context Broker doesn't rotate log itself, but it provides the means to configure **logrotate** to do so:

- Install logrotate, typically using the following (run it as root or using the sudo command):

```
yum install logrotate
```

- Create the file /etc/logrotate.d/contextBroker with the following content for daily rotation with a 7 days persistence. Check the [logrotate documentation](#) for details or if you want to change the configuration (e.g. to rotate weekly instead of daily).

```
/var/log/contextBroker/contextBrokerLog {
```

```

copytruncate
daily
rotate 7
compress
missingok
size 5M
}

```

Many thanks to Massimiliano Nigrelli and his team for all help debugging this configuration.

2.4.5 Watchdog

Although Orion Context Broker is highly stable, it may fail (see the section on [diagnosis procedures](#) for more information about detecting problems with the broker). Thus, it is recommendable to use a watchdog process to detect if the broker process has stopped running, so it can be re-started automatically and/or you get a notification of the problem.

You can write the watchdog program yourself (e.g. a script invoked by cron in a regularly basic that checks `/etc/init.d/contextBroker` status and starts it again if is not working and/or send you a notification email) or use existing tools. This section includes a procedure using [Monit](#).

First of all, install the RPMs available at <http://rpmfind.net/linux/rpm2html/search.php?query=monit>. The following procedure has been prepared considering `monit-5.1.1-4.el6.x86_64.rpm`, although it also should work with other versions of the RPM.

```
sudo rpm -i monit-5.2.5-1.el5.rf.x86_64.rpm
```

Create a directory for monit stuff, eg:

```
/home/orion/monit_CB
```

Create `monitBROKER.conf` file in that directory. In this example, we configure monit to restart `contextBroker` if CPU load is greater than 80% for two cycles or if allocated memory is greater than 200MB for five cycles (that would be a symptom of memory leaking). In addition to resource checking, monit will restart the process if it is down. The duration of a cycle is defined using a monit command line parameter (described below).

```

#####
#####
## Monit control file

```

```
#####
#####

##

## Comments begin with a '#' and extend through the end of the line.
Keywords

## are case insensitive. All path's MUST BE FULLY QUALIFIED, starting
with '/'.

##

##

#####
#####

## Global section

#####
#####

##

set logfile /var/log/contextBroker/monitBROKER.log

set statefile /var/log/contextBroker/monit.state

#####
#####

## Services

#####
#####

##

check host localhost with address localhost

    if failed (url http://localhost:10026/version and content ==
'<version>') for 3 cycles then

        exec "/etc/init.d/contextBroker stop"
```

```
check file monitBROKER.log with path
/var/log/contextBroker/monitBROKER.log

    if size > 50 MB then

        exec "/bin/bash -c '/bin/rm
/var/log/contextBroker/monitBROKER.log; monit -c
/home/localadmin/monit_CB/monitBROKER.conf -p
/var/log/contextBroker/monit.pid reload'"

check process contextBroker with pidfile
/var/log/contextBroker/contextBroker.pid    start program =
"/etc/init.d/contextBroker start"    stop program =
"/etc/init.d/contextBroker stop"

    if cpu > 60% for 2 cycles then alert

    if cpu > 80% for 5 cycles then restart

    if totalmem > 200.0 MB for 5 cycles then restart
```

Make root the owner of that file and set permissions only for owner:

```
sudo chown root:root monitBROKER.conf
sudo chmod 0700 monitBROKER.conf
```

Create monit start script start_monit_BROKER.sh. The "-d" command line parameter is used to specify the checking cycle duration (in the example we are setting 10 seconds).

```
monit -v -c /home/orion/monit_CB/monitBROKER.conf -d 10 -p
/var/log/contextBroker/monit.pid
```

Make root the owner of that file and set execution permissions:

```
sudo chown root:root start_monit_BROKER.sh
sudo chmod a+x start_monit_BROKER.sh
```

To run monit do:

```
cd /home/orion/monit_CB
sudo ./start_monit_BROKER.sh
```

To check that monit is working properly, check that the process exist, e.g.:

```
# ps -ef | grep contextBroker
500      27175      1  0 21:06 ?          00:00:00 monit -v -c
/home/localadmin/monit_CB/monitBROKER.conf -d 10 -p
/var/log/contextBroker/monit.pid
500      27205      1  0 21:06 ?          00:00:00 /usr/bin/contextBroker
-port 10026 -logDir /var/log/contextBroker -pidpath
/var/log/contextBroker/contextBroker.pid -dbhost localhost -db orion;
```

Then, kill contextBroker, e.g.:

```
#kill 27205
```

and check with ps that after a while (less than 30 seconds) contextBroker is up again.

2.4.6 Statistics

New in 0.8.1 FIWARE 3.2.1

The Orion context broker maintains a set of counters for the incoming messages and such. The information is accessed via REST, of course:

```
curl <host>:<port>/statistics
```

A sample *statistics* response:

```
<orion>
  <xmlRequests>16</xmlRequests>
  <jsonRequests>1</jsonRequests>
  <registrations>5</registrations>
  <discoveries>10</discoveries>
  <statisticsRequests>2</statisticsRequests>
</orion>
```

To reset the statistics, the verb DELETE is used:

```
curl -X DELETE <host>:<port>/statistics
```

Resetting the statistics counters yields the following response:

```
<orion>
```

```
<message>All statistics counter reset</message>
</orion>
```

There are a lot of counters, but only those that have been used since the last reset are included in the response to the *statistics* REST request.

2.5 Database administration

Normally you don't need to access MongoDB directly as Orion Context Broker uses it transparently. However, for some operations (e.g. backup, fault recovery, etc.) it is useful to know how the database is structured. This section provides that information.

We assume that the system administrator has knowledge of MongoDB (there are very good and free courses at [10gen education site](#)). Otherwise, we recommend to skip this section and not access the database directly. Even in the case you need to access database directly, be very careful when manipulation it, as some actions could be irreversible ([doing a backup](#) at the beginning it's a good idea).

2.5.1 Database authorization

MongoDB authorization is configured with the `-db`, `-dbuser` and `-dbpwd` options ([see section on command line options](#)). There are different cases to take into account:

- If your MongoDB instance/cluster doesn't use authorization at all, then don't use the `-dbuser` and `-dbpwd` options.
- If your MongoDB instance/cluster uses authorization, then:
 - If you run Orion in single service/tenant mode (i.e. without `-multiservice` option or with `"-multiservice off"`) then you are using only one database (the one specified by the `-db` option) and the authorization is done with `-dbuser` and `-dbpwd` in that database. (This is also the only option for Orion 0.12.0 or before, as these versions don't implement the `-multiservice` feature).
 - If you run Orion in multi service/tenant mode (i.e. with `"-multiservice url"` or `"-multiservice header"`) then the authorization is done at "admin" database using `-dbuser` and `-dbpwd`. As described [later in this manual](#), in multi service/tenant mode, Orion uses several databases (which in addition can potentially be created on the fly), thus authorizing on "admin" DB ensures permissions in all them.

2.5.2 Multiservice/multitenant database separation

New since 0.13.0

Normally, Orion Context Broker uses just one database at MongoDB level (the one specified with the `-db` command line option, typically "orion"). However, when [multitenant/multiservice is used](#) the behaviour is different and the following databases are used (let "`<db>`" be the value of the `-db` command line option):

- The database "<db>" for the default tenant (typically, "orion")
- The database "<db>-<tenant>" for service/tenant "<tenant>" (e.g. if the tenant is named "tenantA" and default -db is used, then the database would be "orion.tenantA").

Per-service/tenant databases are created "on the fly" as the first request involving tenant data is processed by Orion. Note that there is a limitation in MongoDB current versions of 24,000 namespaces (each collection or index in a database consumes a namespace). Orion currently uses 5 collections per database, thus taking into account each collection involves also at least the `_id` index, that will end in a 2,400 services/tenants limit (less if you have more indexes in place).

Finally, in the case of per-service/tenant databases, all collections and administrative procedures (backup, restore, etc.) are associated to each particular service/tenant database.

2.5.3 Collections

Orion Context Broker uses five collections in the database, described in the following subsections.

2.5.3.1 *entities collection*

The *entities* collection stores information about NGSI entities. Each document in the collection corresponds to an entity.

Fields:

- **`_id`** stores the EntityID, including the ID itself and type. Given that we use `_id` for this, we ensure that EntityIDs are unique. The JSON document for this field includes:
 - **`id`**: entity NGSI ID
 - **`type`**: entity NGSI type
 - **`servicePath`** (optional): related with [the service path](#) functionality. **New in 0.14.0.**
- **`attrs`** is an array of the different attributes that have been registered for that entity. Each one of the elements in the array has the following information:
 - **`name`**: the attribute name
 - **`type`**: the attribute type
 - **`value`**: the attribute value (for those attribute that has received at least one update). Up to version 0.10.1, this value is always a string, but in 0.11.0 this value can be also a JSON object or JSON vector to represent an structured value (see section about [structured attribute values in user manual](#)).
 - **`id`**: optional ID. **New in 0.9.0.**
 - **`creDate`**: the timestamp corresponding to attribute creation (as a consequence of append). **New in 0.9.1.**
 - **`modDate`**: the timestamp corresponding to last attribute update. It matches `creDate` if the attribute has not be modified after creation. **New in 0.8.0.**
- **`creDate`**: the timestamp corresponding to entity creation date (as a consequence of append). **New in 0.8.0.**

- **modDate**: the timestamp corresponding to last entity update. Note that it uses to be the same that a modDate corresponding to at least one of the attributes (not always: it will not be the same if the last update was a DELETE operation). It matches creDate if the entity has not been modified after creation. **New in 0.8.0.**
- **location** (optional): geographic location of the entity, composed of the following fields **New in 0.11.0.**:
 - **attrName**: the attribute name that identifies the geographic location in the attrs array
 - **coords**: a tuple containing the latitude and longitude in that order
- **md** (optional): custom metadata. This is a vector of metadata objects, each one with a **name**, **type** and **value**. **New in 0.13.0.**

Since 0.13.0.: an attribute whose metadata changes (although the attribute value stays the same) is considered a change from the point of view of modDate in that attribute or the entity.

Example document:

```
{
  "_id": {
    "id": "E1",
    "type": "T1"
  },
  "attrs": [
    {
      "name": "A1",
      "type": "TA1",
      "value": "282",
      "creDate" : 1389376081,
      "modDate" : 1389376120,
      "md" : [
        {
          "name" : "customMD1",
          "type" : "string",
          "value" : "AKAKA"
        },
        {
          "name" : "customMD2",
          "type" : "integer",
```

```

        "value" : "23232"
    }
]
},
{
    "name": "A2",
    "type": "TA2",
    "value": "176",
    "id": "ID101"
    "creDate" : 1389376244,
    "modDate" : 1389376244
},
{
    "name": "position",
    "type": "location",
    "value": "40.418889, -3.691944",
    "creDate" : 1389376244,
    "modDate" : 1389376244
}
],
"creDate": 1389376081,
"modDate": 1389376244,
"location": {
    "attrName": "position",
    "coords": [ 40.418889 , -3.691944 ]
}
}
}

```

2.5.3.2 *registrations collection*

The *registrations* collection stores information about NGSi9 registrations. Each document in the collection corresponds to a registration.

Fields:

- **_id** is the registration ID (the value that is provided to the user to update the registration). Given that we use **_id** for this, we ensure that registration IDs are unique and that queries by registration IDs will be very fast (as there is an automatic default index in **_id**).
- **fwdRegId**: the ID corresponding to the registration in the forward Context Broker. Used only in "ConfMan mode", (see [programmers guide](#)).
- **expiration**: this is the timestamp for which the registration expires. The expiration is calculated using the duration parameter included in the registerContext operation (basically, sum "now" and duration) and will be recalculated when a registerContext for updating (i.e. using a not null registration ID in the request) is received (see [programmers guide](#)).
- **contextRegistration**: is an array whose elements contain the following information:
 - **entities**: an array containing a list of entities (mandatory). The JSON for each entity contains **id**, **type** and **isPattern**.
 - **attrs**: an array containing a list of attributes (optional). The JSON for each attribute contains **name**, **type** and **isDomain**.
 - **providingApplication**: the URL of the providing application for this registration (mandatory)

Example document:

```
{
  "_id": ObjectId("5149f60cf0075f2fabca43da"),
  "fwdRegId": "5149f60cf0075f241bca22f1",
  "expiration": 1360232760,
  "contextRegistration": [
    {
      "entities": [
        {
          "id": "E1",
          "type": "T1",
          "isPattern": "false"
        },
        {
          "id": "E2",
          "type": "T2",
          "isPattern": "false"
        }
      ]
    }
  ]
}
```

```
    ],
    "attrs": [
      {
        "name": "A1",
        "type": "TA1",
        "isDomain": "false"
      },
      {
        "name": "A2",
        "type": "TA2",
        "isDomain": "true"
      }
    ],
    "providingApplication": "http://dummy1.com"
  },
]
}
```

2.5.3.3 *csubs* collection

The *csubs* collection stores information about NGSI10 subscriptions. Each document in the collection corresponds to a subscription.

Fields:

- **_id** is the subscription ID (the value that is provided to the user to update and cancel the subscription). Given that we use **_id** for this, we ensure that subscription IDs are unique and that queries by subscription IDs are very fast (as there is an automatic default index in **_id**).
- **expiration**: this is the timestamp on which the subscription expires. This is calculated using the duration parameter included in the `subscribeContext` operation (basically, sum "now" and duration) and will be recalculated when an `updateContextSubscription` is received (see [programmers guide](#)).
- **lastNotification**: the time when last notification was sent. This is updated each time a notification is sent, to avoid violating throttling.
- **throttling**: minimum interval between notifications.
- **reference**: the URL for notifications

- **entities**: an array of entities (mandatory). The JSON for each entity contains **id**, **type** and **isPattern**.
- **attrs**: an array of attribute names (strings) (optional).
- **conditions**: a list of conditions that trigger notifications. The two different cases currently supported are shown in the example below (we think they're quite straightforward)
- **count**: the number of notifications sent associated to the subscription. **New in 0.8.0.**
- **format**: the format to use to send notifications, either "XML" or "JSON". **New in 0.9.0.**

Example document:

```
{
  "_id": ObjectId("5149fd46f0075f83a4ca0300"),
  "expiration": 1360236300,
  "lastNotification": 1360232700,
  "throttling": 10,
  "reference": "http://notify.me",
  "entities": [
    {
      "id": "E1",
      "type": "T1",
      "isPattern": "false"
    }
  ],
  "attrs": [
    "A1",
    "A2"
  ],
  "conditions": [
    {
      "type": "ONTIMEINTERVAL",
      "value": 60
    },
    {
      "type": "ONCHANGE",
```

```

        "value": [
            "A1",
            "A2"
        ]
    },
    ],
    "count": 27,
    "format": "XML"
}

```

2.5.3.4 *casubs* collection

The *casubs* collection stores information about NGSi9 subscriptions. Each document in the collection corresponds to a subscription.

Fields:

- **_id** is the subscription ID (the value that is provided to the user to update and cancel the subscription). Given that we use **_id** for this, we ensure that subscription IDs are unique and that queries by subscription IDs are very fast (as there is an automatic default index in **_id**).
- **expiration**: this is the timestamp on which the subscription will expire. It is calculated using the duration parameter included in the `subscribeContextAvailability` operation (basically, sum "now" and duration) and will be recalculated when an `updateContextAvailabilitySubscription` is received (see [programmers guide](#)).
- **reference**: the URL to send notifications
- **entities**: an array of entities (mandatory). The JSON for each entity contains **id**, **type** and **isPattern**.
- **attrs**: an array of attribute names (strings) (optional).
- **lastNotification**: timestamp corresponding to the last notification sent associated to a given subscription. **New in 0.8.0.**
- **count**: the number of notifications sent associated to the subscription. **New in 0.8.0.**
- **format**: the format to use to send notifications, either "XML" or "JSON". **New in 0.9.0.**

Example document:

```

{
  "_id": ObjectId("51756c2220be8dc1b5f415ff"),
  "expiration": 1360236300,
  "reference": "http://notify.me",
  "entities": [

```

```

    {
      "id": "E5",
      "type": "T5",
      "isPattern": "false"
    },
    {
      "id": "E6",
      "type": "T6",
      "isPattern": "false"
    }
  ],
  "attrs": [
    "A1",
    "A2"
  ],
  "lastNotification" : 1381132312,
  "count": 42,
  "format": "XML"
}

```

2.5.3.5 *associations collection*

The *associations* collection stores information about associations. See [the associations support section in the User and Programmers guide](#) for additional details.

Fields:

- **_id** the name of the association (has to be unique)
- **srcEnt**: source entity in the association (expressed as a JSON document with **id** and **type**)
- **tgtEnt**: target entity in the association (expressed as a JSON document with **id** and **type**)
- **attrs**: an array of attribute associations. Each attribute association is a JSON document, with a **src** (for the source attribute) and **tgt** (for the target attribute) in the association.

Example document:

```

{
  "_id": "assoc1",

```

```
"srcEnt": {
  "id": "E1",
  "type": "T1"
},
"tgtEnt": {
  "id": "E2",
  "type": "T2"
},
"attrs": [
  {
    "src": "A1",
    "tgt": "B1"
  },
  {
    "src": "A2",
    "tgt": "B2"
  }
]
```

2.5.4 Indexes

Since 0.14.0

Orion Context Broker doesn't ensure any index in the database collection (except for one exception, described at the end of this section) in order to let flexibility to database administrators. Take into account that index usage involves a tradeoff between read efficiency (usage of indexes generally speeds up reads) and write efficiency (the usage of indexes slow down writes) and storage (indexes consume space in database and mapped RAM memory) and that is the administrator (not Orion) who has to decide what to prioritize.

However, in order to help administrator in that task, the following indexes could be recommended:

- Collection [entities](#)
 - `_id.id` (used by `queryContext` and related convenience operations)
 - `_id.type` (used by `queryContext` and related convenience operations)

- `_id.servicePath` (used by `queryContext` and related convenience operations)
- `creDate` (used to provided ordered results in `queryContext` and related convenience operations)
- Collection [registrations](#)
 - `_id` (used to provided ordered results in `discoverContextAvailability` and related convenience operations). We include this index here for the sake of completeness, but the administrator doesn't need to explicitly ensure it, given that MongoDB automatically provides a mandatory index for `_id` in every collection.

The only index that Orion Context Broker actually ensures is the "2dsphere" one in the `location.coords` field in the `entities` collection, due to functional needs (in order [geo-location functionality](#) to work). The index is ensured at Orion startup or when entities are created for first time.

2.5.5 Database management scripts

Orion Context Broker comes along with some scripts that can be use to do some browsing and administrative actions in the database, installed in the `/usr/share/contextBroker` directory.

In order to use them, you need to install the `pymongo` driver (version 2.5 or above) as a requirement to run it, typically using (run it as root or using the `sudo` command):

```
pip-python install pymongo
```

2.5.5.1 *Deleting expired documents*

NGSI specifies an expiration time for registrations and subscriptions (both NGSI9 and NGSI10 subscriptions). Orion Context Broker doesn't delete the expired documents (it just ignores them) due to the fact that expired registrations/subscription can be "re-activated" by an update of their duration.

However, expired registrations/subscriptions consume space in the database, so they can be "purged" from time to time. In order to help you in that task, the `garbage-collector.py` script is provided along with the Orion Context Broker (in `/usr/share/contextBroker/garbage-collector.py` after installing the RPM).

The `garbage-collector.py` looks for expired documents in `registrations`, `csubs` and `casubs` collection, "marking" them with the following field:

```
{
  ...
  "expired": 1,
  ...
}
```

The garbage-collector.py program takes as arguments the collection to be analyzed (**Note in version 0.8.1 FIWARE 3.2.1**: previous version doesn't take any argument and analyze always all collections), e.g. to analyze csubs and casubs, run:

```
# garbage-collector.py csubs casubs
```

After running garbage-collector.py you can easily remove the expired documents using the following commands in the mongo console:

```
# mongo <host>/<db>
> db.registrations.remove({expired: 1})
> db.csubs.remove({expired: 1})
> db.casubs.remove({expired: 1})
```

2.5.5.2 Latest updated document

You can take an snapshot of the latest updated entities and attributes in the database using the latest-updates.py script. It takes up to four arguments:

- Either "entities" or "attributes", to set the granularity level in the updates.
- The database to use (same than the -db parameter and BROKER_DATABASE_NAME used by the broker). Note that the mongod instance has to run in the same machine where the script runs.
- The maximum number of lines to print
- (Optional) A filter for entity IDs, interpreted as a regular expression in the database query.

Ej:

```
# latest-updates.py entities orion 4
-- 2013-10-30 18:19:47: Room1 (Room)
-- 2013-10-30 18:16:27: Room2 (Room)
-- 2013-10-30 18:14:44: Room3 (Room)
-- 2013-10-30 16:11:26: Room4 (Room)
```

2.5.6 Delete complete database

Release Note (before 0.8.1 FIWARE 3.2.1): previous versions use the -reset command line option to do this reset automatically at Orion broker starting time. However, we have removed it in 0.8.1, given it could be dangerous.

This operation is done using the MongoDB shell:

```
# mongo <host>/<db>
```

```
> db.dropDatabase()
```

2.5.7 Delete a single entity

Since release 0.11.0: the recommended procedure is deleting the entities using NGSI REST API, see [this section in the user manual](#) for details.

It is possible to remove entities using MongoDB shell:

```
# mongo <host>/<db>
> db.entities.remove({"_id.id": <entity ID>, "_id.type": <entity
type>})
```

For example, to remove "Room1" entity of type "Room", use the following:

```
# mongo <host>/<db>
> db.entities.remove({"_id.id": "Room1", "_id.type": "Room"})
```

2.6 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.

2.6.1 End to End testing

- Start context broker in default port (1026)
- Run the following command

```
curl localhost:1026/version
```

- Check that you get the version number as output (along with uptime information and compilation environment information):

```
<orion>
  <version>0.8.0-next</version>
  <uptime>0 d, 0 h, 0 m, 11 s</uptime>
  <git_hash>3fdb55b96913b3e4d9f9a344e990164650f69b91</git_hash>
  <compile_time>Wed Oct 30 15:31:29 CET 2013</compile_time>
  <compiled_by>fermin</compiled_by>
  <compiled_in>centollo</compiled_in>
</orion>
```

2.6.2 List of Running Processes

A process named "contextBroker" should be up and running, e.g.:

```
# ps ax | grep contextBroker  
  
8517 ?          Ssl      8:58 /usr/bin/contextBroker -port 1026 -logDir  
/var/log/contextBroker -pidpath  
/var/log/contextBroker/contextBroker.pid -dbhost localhost -db orion
```

2.6.3 Network interfaces Up & Open

Orion Context Broker uses TCP 1026 as default port, although it can be changed using the -port command line option.

2.6.4 Databases

The Orion Context Broker uses a MongoDB database, whose parameters are provided using the command line options "-dbhost", "-dbuser", "-dbpwd" and "-db". Note that "-dbuser" and "-dbpwd" are only used if MongoDB runs using authentication, i.e. with "--auth".

You can check that the database is working using the mongo console:

```
# mongo <dbhost>/<db>
```

You can check the different collections used by the broker using the following commands in the mongo console. However, note that the broker creates a collection the first time a document is to be inserted in it, so if it is the first time you run the broker (or if database was cleaned) and the broker hasn't received any request yet no collection exists. Use "show collections" to get the actual collections list in any given moment.

```
> db.registrations.count()  
> db.entities.count()  
> db.csubs.count()  
> db.casubs.count()  
> db.associations.count()
```

2.7 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 often have to resort to more concrete and specific testing to pinpoint the exact point of failure and a possible solution. Such specific testing is out of the scope of this section.

Please report any bug or problem with Orion Context Broker by email to fermin@tid dot es.

Apart from the procedures described in the following sections, take into account the following ones.

Diagnose database connection problems

The symptoms of a database connection problem are the following ones:

- At start time. The broker doesn't start and the following message appears in the log file:

```
X@08:04:45 main[313]: MongoDB error
```

- During broker operation. Error message like the following ones appear in the responses sent by the broker.

```
...
<errorCode>
  <code>500</code>
  <reasonPhrase>Database Error</reasonPhrase>
  <details>collection: ... - exception: Null cursor</details>
</errorCode>
...

...
<errorCode>
  <code>500</code>
  <reasonPhrase>Database Error</reasonPhrase>
  <details>collection: ... - exception: socket exception
[CONNECT_ERROR] for localhost:27017</details>
</errorCode>
...

...
<errorCode>
  <code>500</code>
  <reasonPhrase>Database Error</reasonPhrase>
  <details>collection: ... - exception: socket exception
[FAILED_STATE] for localhost:27017</details>
```

```

</errorCode>

...

...
<errorCode>

  <code>500</code>

  <reasonPhrase>Database Error</reasonPhrase>

  <details>collection: ... - exception: DBClientBase::findN:
transport error: localhost:27017 ns: orion.$cmd query: { ..
}</details>

</errorCode>

...

```

In both cases, check that the connection to MongoDB is correctly configured (in particular, the `BROKER_DATABASE_HOST` if you are running Orion Context Broker [as a service](#) or the `"-dbhost"` option if you are running it [from the command line](#)) and that the `mongod` or `mongos` process (depending if you are using sharding or not) is up and running.

If the problem is that MongoDB is down, note that Orion Context Broker is able to reconnect to the database once it gets ready again. In other words, you don't need to restart the broker in order to reconnect to the database.

Diagnose spontaneous binary corruption problems

The symptoms of this problem are:

- Orion Context Broker send empty responses to REST requests (e.g. with curl the message is typically "empty response from server"). Note that it could happen that request on some URLs work normally (e.g. `/version`) while in others the symptom appears.
- The MD5SUM of `/usr/bin/contextBroker` binary (that can be get with `"md5sum /usr/bin/contextBroker"`) is not the right one (check list for particular versions at the end of this section).
- The prelink package is installed (this can be checked running the command `"rpm -qa | grep prelink"`)

The cause of this problem is [prelink](#), a program that modifies binaries to make them starting faster (which is not very useful for binary implementing long running services as contextBroker is) but that is known to be incompatible with some libraries (in particular, it seems to be incompatible with some of the libraries used by Context Broker).

The solution fo this problems is:

- Disable prelink, either implementing one of the following alternatives:

- Remove the prelink software, typically running (as root or using sudo): "rpm -e prelink"
- Disable the prelink processing of contextBroker binary, creating the /etc/prelink.conf.d/contextBroker.conf file with the following content (just one line):

```
-b /usr/bin/contextBroker
```

- Re-install the contextBroker package, typically running (as root or using sudo):

```
yum remove contextBroker  
yum install contextBroker
```

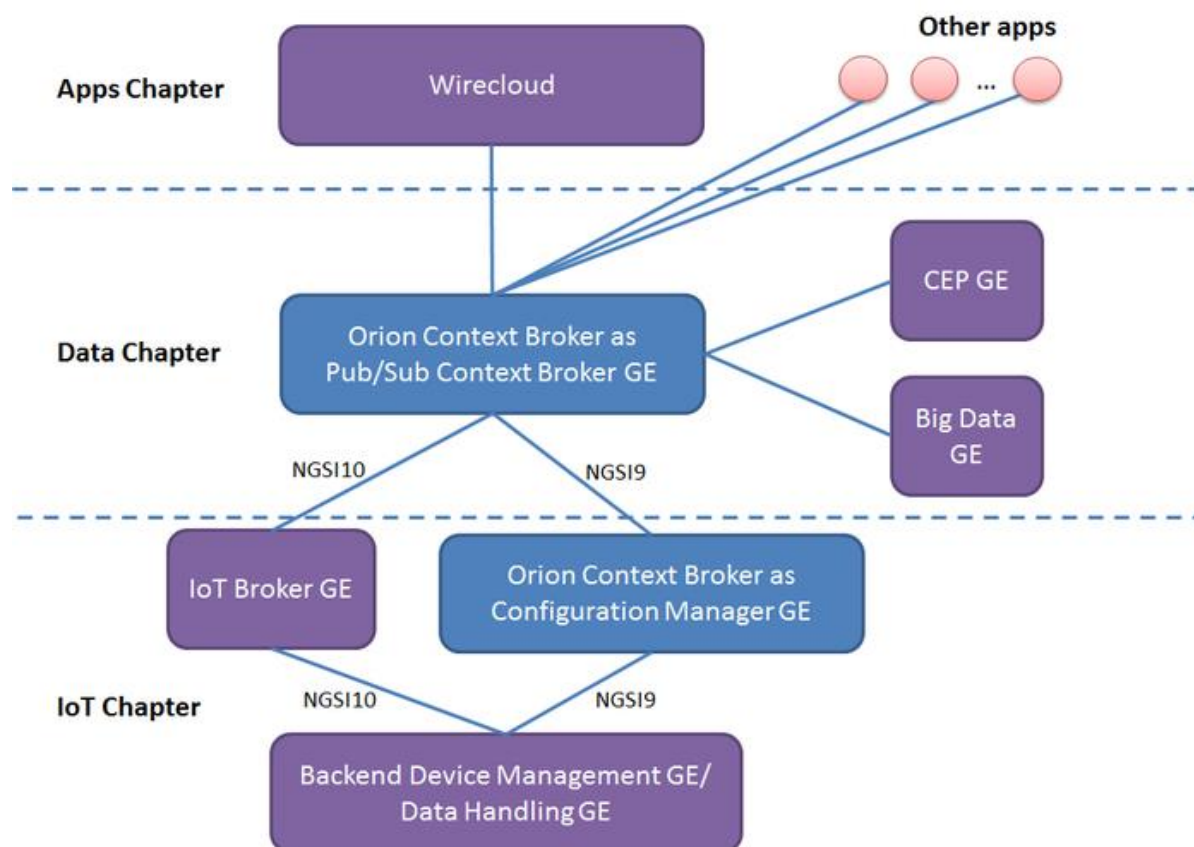
2.7.1 Resource availability

Although we haven't done yet a precise profiling on Orion Context Broker, tests done in our development and testing environment show that a host with 2 CPU cores and 4 GB RAM is fine to run the ContextBroker and MongoDB server. In fact, this is a rather conservative estimation, Orion Context Broker could run fine also in systems with a lower resources profile. The critical resource here is RAM memory, as MongoDB performance is related to the amount of available RAM to map database files into memory.

2.7.2 Remote Service Access

Orion Context Broker can run "standalone", thus context consumer and context producers are connected directly to it through its NGSI interface. Thus, it is loosely coupled to other FI-WARE GEs. However, considering its use in the FI-WARE platform, below is a list of the GEs that typically can be connected to the broker:

- When running as Pub/Sub Context Broker GE, it typically connects to the IoT Broker GE (from IoT chapter GEs), other GEs within the Data Chapter (such as CEP or BigData) and GEs from the Apps chapter (such as Wirecloud). As an alternative, the IoT Broker GE could be omitted (if the things-to-device correlation is not needed) thus connecting Orion Context Broker directly to the Backend Device Management GE or to the DataHandling GE. In this case the broker uses typically port 1026 (the default one).
- When running as ConfMan GE, the Orion Context Broker is connected to the IoT Broker GE, the Pub/Sub Context Broker and the DataHandling GE or the Backend Device Management GE. In this case the broker uses typically port 1027.



2.7.3 Resource consumption

The most usual problems that Orion Context Broker may have are related to abnormal consumption of memory due to leaks and disk exhaustion due to growing log files.

Regarding abnormal consumption of memory, it can be detected by the the following symptoms:

- The broker crashes with a "Segmentation fault" error
- The broker doesn't crash but stops processing requests, i.e. new requests "hang" as they never receive a response. Usually, the Orion Context Broker has only two permanent connections in use as shown below (one with the database server and the listening TCP socket in 1026 or in the port specified by "-port") but in the case of this problem each new request will appear as a new connection in use in the list. The same information can be checked using "ps axo pid,ppid,rss,vsz,nlwp,cmd" and looking to the number of threads (nlwp column), as a new thread is created per request but never released. In addition, you can check the broker log and see that the processing of new requests stops in the access to the MongoDB database (in fact, what is happening is that the MongoDB driver is requesting more dynamic memory to the OS but it doesn't get any and keeps waiting until some memory gets freed, which never happens).

```
# sudo lsof -n -P -i TCP | grep contextBr
```



```
contextBr 7100      orion      6u  IPv4 6749369      0t0  TCP
127.0.0.1:45350->127.0.0.1:27017 (ESTABLISHED)

contextBr 7100      orion      7u  IPv4 6749373      0t0  TCP *:1026
(LISTEN)
```

- The consumption of memory shown by "top" command for the contextBroker process is abnormally high.

The solution to this problem is restarting the contextBroker, e.g. `/etc/init.d/contextBroker restart`

Regarding disk exhaustion due to growing log files, it can be detected by the following symptoms:

- The disk is full, e.g. "df -h" shows that the space available is 0%
- The log file for the broker (usually in `/var/log/contextBroker/contextBrokerLog`) is very big

The solutions for this problem are the following:

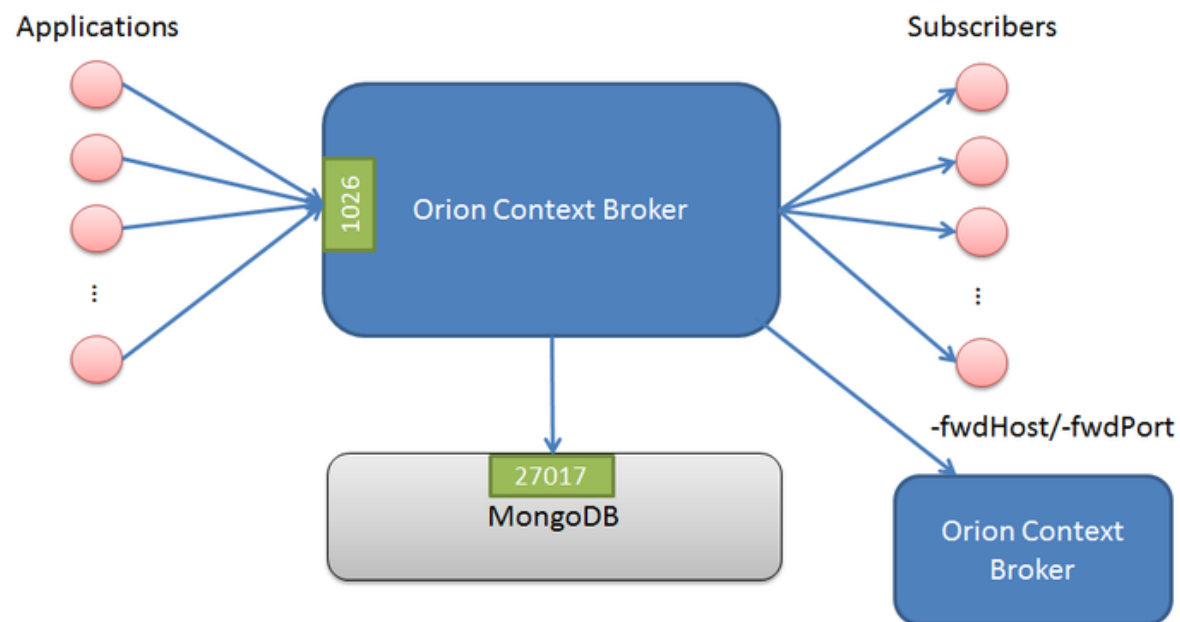
- Stop the broker, remove the log file and start the broker again
- Configure [log rotation](#)
- Reduce the log verbosity level, e.g. if you are using "-t 0-255" the log will grow very fast so, in case of problems, please avoid using unneeded trace levels.

2.7.4 I/O flows

The Orion Context Broker uses the following flows:

- From NGSi9/10 applications to the broker, using TCP port 1026 by default (this is overridden with "-port" option).
- From the broker to subscribed applications, using the port specified by the application in the callback at subscription time.
- From the broker to MongoDB database. In the case of running MongoDB in the same host as the broker, this is an internal flow (i.e. using the loopback interface). The standard port in MongoDB is 27017 although that can be changed in the configuration. Intra-MongoDB flows (e.g. the synchronization between master and slaves in a replica set) are out of the scope of this section and not shown in the picture.
- From the broker to another instance of broker, when registerContext forwarding is in use (see [programmers manual](#)).

Note that the throughput in these flows can not be estimated in advance, as it depends completely on the amount of external connections from context consumer and producers and the nature of the requests issued by consumers/producers.



3 Publish/Subscribe GE - Context Awareness Platform - Installation and Administration Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

3.1 Description of Context Awareness Platform implementing Publish/Subscribe Context Broker

The Context Awareness Platform (referred as CAP in the following), which implements Publish/Subscribe Context Broker, consists of front-end server modules and back-end databases, which could share the same physical machine (in further paragraphs it is assumed to have separate ones).

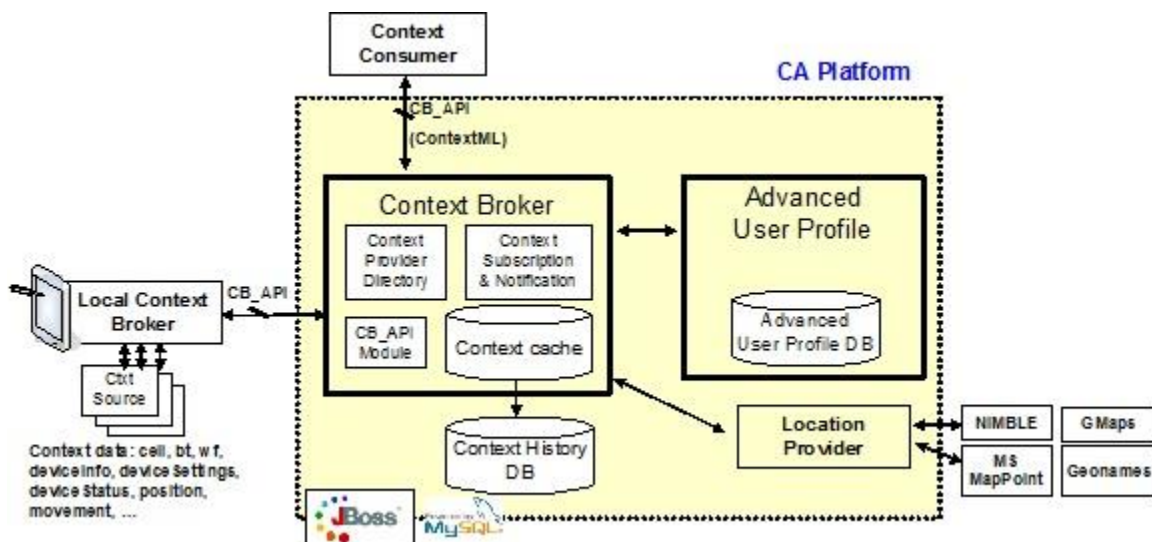
The CAP exposes REST http-based interfaces to publish and to retrieve context information, by means of ContextML or NGSI representation model, in both synchronous or asynchronous mode (i.e. subscription/notification).

Once the service is run it is always on and potentially available for any application or service requesting the context, so for both 3rd parties developers and end-users of the applications or services.

Context Provider could be dynamically registered or unregistered within the CAP using the registering announcement interfaces available. This allows decoupling Context Providers' interfaces within the CAP interfaces exposed towards Context Consumers, which are always available in order to serve any context-aware application or customer using the context.

3.2 Software architecture

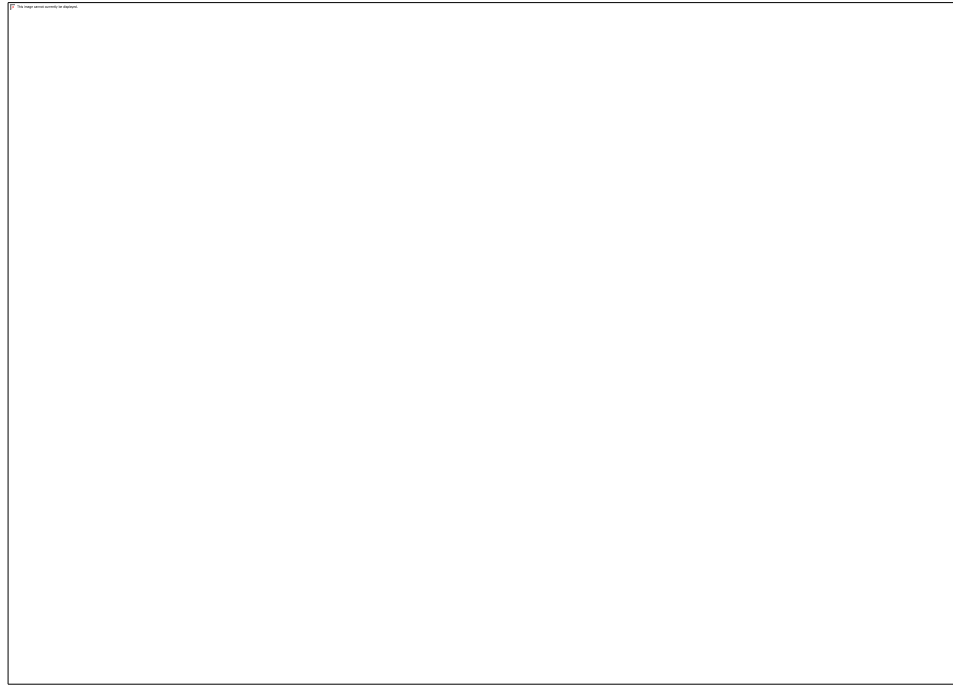
The following figure shows the software architecture of the CAP.



CAP's SW Architecture

3.3 Deployment architecture

The following figure shows the high level deployment architecture of entire CAP including its context handling components and the databases.



CAP Deployment Architecture

Detailed description of the SW components and their operations are reported in the sections below.

3.3.1 Databases

For hardware requirement refer to next section "Software modules".

3.3.1.1 **Software requirements**

- MySQL Server v5.0 DBMS with UTF-8 and InnoDB support

3.3.1.2 **Installation**

1. Launch the "create_user.sql" script available in specific installation package, which:
 - a. Creates a new user "causer" on MySQL DBMS,
 - b. Sets permissions to user "causer" for creating databases, read/write access to data on databases created here below, both locally and remotely
2. Execute scripts from mysql command line. Since some databases may be not present, the procedures applies to scripts available in specific installation package. Scripts include commands to create databases

```
> mysql -u causer -p --default-character-set utf8 -e "source  
filename.sql"
```

- a. cbngsi_ULH.sql (CAP database)
- b. aup.sql
- c. abp.sql
- d. other databases (if present): lp, oauth, ecc....

3.3.1.3 **Configuration**

After databases have been installed the following database configurations must be completed:

1. Copy "mysql-CAP-ds.xml" file into "%JBoss_HOME%\server\ca_cap\deploy" folder.
2. In the file "mysql-CAP-ds.xml" check DB credentials and IP/ports, modifying the following parameters in each defined datasource, for example:

```
<connection-url>jdbc:mysql://DATABASE_HOST:3306/db_name</connection-  
url>  
  
<user-name>username</user-name>  
  
<password>password</password>
```

where DATABASE_HOST is the IP address of the database server, username and password are the access credentials and db_name is the database name for each created database.

3.3.2 Software modules

CAP software modules are Java Enterprise (J2EE) applications to be deployed in a JBossAS instance.

3.3.2.1 **Requirements**

Hardware requirements

- 1 server with 2 CPU;
- 2 GB RAM;
- minimum 50 GB free hard disk space on used partition

Software requirements

- Operating System
 - RedHat CentOS 32-bit (recommended, but also Ubuntu or Windows 2003 Server are supported)
- Software components (recommended versions):

- Java SE Development Kit (JDK) v1.6.0, available from <http://www.oracle.com/technetwork/java/javase/downloads>
- JBoss Application Server v5.1.0GA, available from <http://www.jboss.org/jbossas/downloads/>

Connectivity requirements

- The server where the CAP will be installed requires connectivity with:
 - MySQL Server v5.0 DBMS for database connections
 - back-end systems (external providers)
 - All TCP ports required by JBoss default configuration (8080, etc.) must be available

The CAP by default will provide access to the offered APIs on TCP Jboss port 8080 (or 80 if reverse proxied).

3.3.2.2 **JDK and JBoss installation**

NOTE All the installation procedure should be performed as “root” user

3.3.2.2.1 *CENTOS JDK installation*

1. Download file jdk-6u29-linux-i586-rpm.bin from Oracle site (or newer jdk-6u....-rpm.bin files) in “/root” folder and operate in that folder.

2. Set permissions:

```
> chmod a+x jdk-6u29-linux-i586-rpm.bin
```

3. Execute the file with command:

```
> ./jdk-6u29-linux-i586-rpm.bin
```

4. JDK should be installed in /usr/java/ folder.

5. Test installation with:

```
> java -version
```

3.3.2.2.2 *UBUNTU JDK installation*

1. Check file /etc/apt/sources.list, if necessary uncomment lines:

o deb <http://archive.canonical.com/ubuntu> maverick partner

o deb-src <http://archive.canonical.com/ubuntu> maverick partner

and launch the following command to update the repository configuration:

```
> apt-get update
```

2. Execute the command:

```
> apt-get install sun-java6-jdk
```

3. JDK should be installed in /usr/lib/jvm/java-6-sun folder.

4. Test installation with:

```
> java -version
```

3.3.2.2.3 JBoss installation and first run

1. Download JBoss (jboss-5.1.0.GA-jdk6.zip, from <http://sourceforge.net/projects/jboss/files/JBoss/JBoss-5.1.0.GA/jboss-5.1.0.GA-jdk6.zip/download>) and decompress it in a chosen installation folder (e.g. "/usr/local/jboss-5.1.0"):

```
> cd /tmp
> wget http://sourceforge.net/projects/jboss/files/JBoss/JBoss-5.1.0.GA/jboss-5.1.0.GA-jdk6.zip
> cd /usr/local
> unzip /tmp/jboss-5.1.0.GA-jdk6.zip
> mv ./jboss-5.1.0.GA-jdk6 ./jboss-5.1.0
```

In the following %JBOSS_HOME% will refer to the chosen JBoss installation directory.

2. Copy the following file from the installation package (folder JBoss_bin) in %JBOSS_HOME%\bin:

a. log4j.properties

3. Make a copy of "%JBOSS_HOME%\server\default" and name it as "%JBOSS_HOME%\server\ca_cap":

```
> cp -R %JBOSS_HOME%/server/default %JBOSS_HOME%/server/ca_cap
```

4. Copy the file "jboss-log4j.xml" from the installation package (folder JBoss_instance_conf) into "%JBOSS_HOME%\server\ca_cap\conf"

5. Create a JBoss user, without password (Ubuntu requires you type some ENTER after every subsequent question):

```
> adduser jboss
```

6. Copy init redhat script renamed to /etc/init.d/jboss:

```
> cp %JBOSS_HOME%/bin/jboss_init_redhat.sh /etc/init.d/jboss
```

7. Set script permissions:

```
> chmod 755 /etc/init.d/jboss
```

```
> chown -R jboss /usr/local/jboss-5.1.0
> chown jboss /etc/init.d/jboss
```

8. Modify the script /etc/init.d/jboss:

a. Add the following lines just after line “#!/bin/sh” (verify that java path JAVAPATH and JBOSS_HOME folder match the correct ones):

```
# chkconfig: 2345 95 20
# description: Starts and stops JBossAS
# processname: jboss
JBOSS_HOME="/usr/local/jboss-5.1.0"
JAVAPATH="/usr/java/default"
JBOSS_CONF="ca_cap"
JBOSS_HOST=${JBOSS_HOST:-"0.0.0.0"}
```

9. Verify /etc/hosts file, it should contain the server name in the “127.0.0.1” line, otherwise there could be some UnknownHostException during JBoss startup.

It is recommended to change memory settings, substituting the following line of run.conf file in %JBOSS_HOME%\bin folder:

```
JAVA_OPTS="-Xms128m -Xmx512m -XX:MaxPermSize=256m -Dorg.jboss.resolver.warning=true -
Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000"
```

with this one:

```
JAVA_OPTS="-Xrs -Xms512m -Xmx512m -XX:NewSize=64m -XX:MaxNewSize=64m -XX:SurvivorRatio=6 -
XX:PermSize=256m -XX:MaxPermSize=256m -Xdebug -Dorg.jboss.resolver.warning=true -
Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000"
```

Follow these steps to launch JBoss for the first time from the command line. Check for errors or problems (e.g. port conflicts, etc.), enter command:

```
> /etc/init.d/jboss start
```

From a web browser, verify the JBoss application server opening the following URL:

[http://\[server_IP\]:8080/web-console/](http://[server_IP]:8080/web-console/)

If the page is not reachable from another machine, verify firewall settings (use commands “setup” to disable/configure it, and “iptables -L” to verify).

To stop JBoss, use command:


```
> /etc/init.d/jboss stop
```

To run JBoss at the OS startup/reboot, under CentOS, use commands:

```
> cd /etc/init.d
> chkconfig --add jboss
```

(or use “ntsysv” utility)

Under Ubuntu, use instead the following commands:

```
> cd /etc/init.d
> update-rc.d jboss defaults
```

At the end of this process, JBoss should be started as a system service.

- Common libraries installation

- Copy all the files from "JBoss_instance_lib" folder of the installation package into the folder "%JBOSS_HOME%\server\ca_cap\lib"

- Components installation

1. Copy the following files or folders provided in "JBoss_instance_deploy" folder of the installation package into "%JBOSS_HOME%\server\ca_cap\deploy " folder, where x.x.x is the version number of the component:

- cb-x.x.x.ear

- aup-x.x.x.ear

- schemas.war

- ngssi-cbwrapper-x.x.x.war

- lp-x.x.x.ear (if present)

2. Copy the file ca_commons_config.properties from "JBoss_instance_conf" folder of the installation package into "%JBOSS_HOME%\server\ca_cap\conf" folder

3. If present, copy other *_config.properties file from "JBoss_instance_conf" folder to the same folder.

4. Copy "CB-jboss-destinations-service.xml" and "aup-activation-destination-service.xml" files from "JBoss_instance_deploy/messaging" folder of the installation package into "%JBOSS_HOME%\server\ca_cap\deploy\messaging" folder

3.3.3 Installation behind a reverse proxy

For security purposes, an installation behind an Apache reverse proxy could be an option, since it allows to expose to the public internet only the needed endpoints (e.g. without exposing JBoss administrative console or other backend web applications hosted on the same JBoss instance). In such cases, a simple example of Apache virtual host configuration could be the following (extract of file "httpd.conf", or equivalent for different Apache distributions):

```
...
ProxyRequests off

<VirtualHost *:80>

ProxyPass /ngsicbapi http://[JBOSS_SERVER_IP]:8080/ngsicbapi
ProxyPassReverse /ngsicbapi http://[JBOSS_SERVER_IP]:8080/ngsicbapi

ProxyPass /CB http://[JBOSS_SERVER_IP]:8080/CB
ProxyPassReverse /CB http://[JBOSS_SERVER_IP]:8080/CB
</VirtualHost>
```

Some further material is available in the "apache httpd.conf" folder, provided in the installation packages.

3.3.4 Configuration

The configuration properties are in the file:

"%JBOSS_HOME%\server\ca_cap\conf\ca_commons_config.properties", provided in the installation package. Usually no modification is needed.

3.3.4.1 **Logging Configuration**

Log files can be found in the folder: %JBOSS_HOME%/server/ca_cap/log.

Log files are automatically rotated by the application server.

Configuration files for logging are:

- "%JBOSS_HOME%\bin\log4j.properties" for JBoss boot logging (when launched in manual mode)
- "%JBOSS_HOME%\server\ca_cap\conf\jboss-log4j.xml" for all other logging

To edit logging levels and appenders change the logging configuration in "%JBOSS_HOME%/server/ca_cap/conf/jboss-log4j.xml".

For example, it is possible to modify the following parameters:

```

    <appender name="SIZED_FILE_CB"
class="org.jboss.logging.appender.RollingFileAppender">
        <errorHandler
class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
        <param name="File" value="{jboss.server.log.dir}/cb.log"/>
        <param name="Append" value="true"/>
        <param name="Encoding" value="UTF-8"/>
        <param name="Threshold" value="INFO"/>
        <param name="MaxFileSize" value="10000KB"/> <!-- rotation file
size -->
        <param name="MaxBackupIndex" value="10"/><!-- rotation files --
>

        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d{ISO8601} %-5p [%c]
(%t:%x) %m%n "/>
        </layout>

    </appender>

[...]

<category name="CB" additivity="false">
    <level value="INFO" />
    <appender-ref ref="SIZED_FILE_CB" />
    [...]
</category>

```

It is possible to select manually the threshold for the output of log messages within a single appender by editing the attribute value of the tag param Threshold.

Accepted log levels are:

- DEBUG
- INFO
- WARN
- ERROR
- FATAL

3.3.4.2 *JBoss scripts*

The service.bat and run.conf.bat files provided in CAP installation package can be derived from the original versions in JBoss bin folder, applying the following modifications:

- File service.bat

Comment this row:

```
set JAVA_OPTS=-Xrs
```

In ":cmdStart" and ":cmdRestart" functions , find the string

```
call run.bat < .r.lock >> run.log 2>&1
```

and modify it adding "-c all" and "-b 0.0.0.0":

```
call run.bat -c all -b 0.0.0.0 < .r.lock >> run.log 2>&1
```

In ":cmdStop" and ":cmdRestart" functions , find the string

```
call shutdown < .s.lock >> shutdown.log 2>&1
```

and modify it adding "--server=jnp://localhost:1099" and "--shutdown":

```
call shutdown --server=jnp://localhost:1099 --shutdown < .s.lock >>  
shutdown.log 2>&1
```

- File run.conf.bat

Modify the following row by adding "-Xrs " at the beginning of the string:

```
set "JAVA_OPTS=-Xms128M -Xmx512M -XX:MaxPermSize=256M"
```

3.4 Sanity check procedures

3.4.1 End to End testing

Try to invoke this URL from a browser:

```
http://[serverIP]:8080/CB/ContextBroker/ping
```

If everything is OK, the server should return a response similar to the following one:

```
<?xml version="1.0" encoding="UTF-8"?>

<contextML xmlns="http://ContextML/1.7"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ContextML/1.7
http://contextml.tilab.com/ContextML-1.7.xsd">

<ctxResp>

<contextProvider id="CB" v="1.4.6"/>

<timestamp>2012-07-26T15:33:38+02:00</timestamp>

<method>ping</method>

<resp status="OK" code="200"/>

<dataPart>

<par n="start">2012-07-18T10:41:48+02:00</par>

<parA n="methods">

<parS n="method">

<par n="method">getContext</par>

<par n="requests">0</par>

</parS>

<parS n="method">

<par n="method">contextUpdate</par>

<par n="requests">0</par>

</parS>

</parA><parA n="activeProviders">

<par n="activeProvider">ABP</par>

<par n="activeProvider">AUP</par>

<par n="activeProvider">UPP</par>
```

```
</parA>
<parA n="inactiveProviders" />
</dataPart>
</ctxResp>
</contextML>
```

If something is wrong, the response shows a description of the problem.

3.4.2 List of Running Processes

JBoss runs in a single JVM process (named "java ..."), which should be visible to a "ps -aux" command. It is important to verify that only ONE "java" process is active.

3.4.3 Network interfaces Up & Open

In normal installation CAP services are exposed to standard JBoss port 8080, but for security reasons it is recommended to reverse proxy this port to the standard HTTP 80.

3.4.4 Databases

The ping command checks also the availability of database connection.

For a direct check, use command "telnet localhost 3306" on the database server, or "mysql" command line. Active MySQL databases (user/pwd: causer/cadb) could be checked with the following SQL queries:

- context_broker:
 - check with query: "select count(*) from atomicscope;"
- aup
 - check with query: "select count(*) from user"

3.5 Diagnosis Procedures

3.5.1 Resource availability

A simple command "telnet localhost 8080" should test if JBoss is alive.

A typical host running the CAP shall have at least:

- 1GB of free RAM;
- 5GB of free hard-drive space.

3.5.2 Remote Service Access

There are no remote connections established if no external entities like providers or consumers are connected. Providers or consumers make TCP connections to the CAP server to send HTTP requests, by means of JSON or XML protocol.

3.5.3 Resource consumption

It depends on the platform load. It can be assumed that a memory consumption of 500Mb to 1Gb is related to a normal working state. If the memory associated to JBoss java process is much higher, or the CPU occupancy is near to 100% in idle state, it may be due to a malfunctioning.

3.5.4 I/O flows

The CAP listening port shall be enabled for incoming connections on the local firewall as well as the outgoing connectivity to the CAP server IP and listening port shall be enabled in the remote (consumer or provider, if remote) firewall.

The communication is a TCP session between the CAP server and a remote entity (provider or consumer).

The CAP server firewall shall enable the outgoing connection from the CAP to the subscribed consumer's IP/port in case of context notification related to a subscription.

4 Publish/Subscribe Semantic Extension - Installation and Administration Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

This chapter describes the procedure for installing and deploying the PubSub Semantic extension component.

4.1.1 HW requirements

Server running a linux operating system. The current deployment is based on an Ubuntu 12.04.2 virtual machine, with 2048 Mb of RAM and a 10Gb Disk.

4.1.2 SW prerequisite

A Virtuoso DBMS should be installed. In the current deployment, we use the version 6.1.4 open source version, distributed by OpenLink. The process for installing Virtuoso DBMS is explained in detail in the README file that you'll find in the root directory once you've extract the files from the virtuoso distribution file.

Henceforth, we will call \$HOME the root directory where all software components will be installed.

We recommend that Virtuoso be installed in directory \$HOME/Virtuoso and that Virtuoso commands are made available from the shell. For this latter point, we will add Virtuoso commands directory to the PATH environment variable, using the following unix command:

```
export PATH=$PATH:$HOME/Virtuoso/bin
```

In order to launch Virtuoso DBMS server, do the following from the shell:

```
cd $HOME/Virtuoso/var/lib/virtuoso/db  
virtuoso-t -f &
```

4.1.3 Base SW installation

The semantic extension SW includes 2 components:

- the SemanticWrapper which basically implements the CML2RDF translator functional component.
- the PSFrontEnd which is a web service through which PubSub clients will access the semantic extension of the component. More precisely this component will dispatch the client queries to the SPARQL engine or to the ContextBroker interface depending on the dialect of the query.

4.1.3.1 ***SemanticWrapper installation***

The distribution file should be uncompressed and the service should be launched in the background.

```
cd $HOME
unzip SemanticWrapper.zip
cd SemanticWrapper
./start.sh &
```

If at any stage the service has to be shutdown, simply do:

```
./stop.sh
```

4.1.3.2 ***PSFrontEnd installation***

The distribution file should be uncompressed and the service should be launched in the background.

```
cd $HOME
unzip PSFrontEnd.zip
cd PSFrontEnd
./start.sh &
```

If at any stage the service has to be shutdown, simply do:

```
./stop.sh
```

4.1.4 Binding the SemanticWrapper to the Context Broker

At some stage the Semantic Extension should receive and process CML content produced by the Context Broker. This is done by subscribing the Semantic Extension to the Context Broker. As introduced in the User Guide, this can be done using three alternative methods:

- using curl utility command
- using a REST client
- a Java program

In the following we use the Java program approach. We show the complete source code in the following paragraph. We will later have a look and explain the salient fragments of the code.

```
package orange.fiware.pubsub.cql;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;

import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriBuilder;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;
import com.sun.jersey.api.representation.Form;

/**
 * @author fyur5281
 * this class enables to post requests to the CAP Broker
 */
public class PostSubscriptionRequestClient {

    public static void main(String[] args) {
        System.getProperties().put("proxyHost", "p-
goodway.rd.francetelecom.fr");

        System.getProperties().put("proxyPort", "8080");
        ClientConfig config = new DefaultClientConfig();
        Client client = Client.create(config);
        WebResource service = client.resource(getBaseURI());
    }
}
```

```

        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));

        try {
            System.out.println("enter validity (in minutes):");
            String validityStr = new String(in.readLine());
            Integer validityInteger = Integer.parseInt(validityStr);
            int validity = validityInteger.intValue();
            String response = postSubscriptionRequest(service,
validity);

            System.out.println(response);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

    private static URI getBaseURI() {
        return UriBuilder.fromUri("http://pubsub.lab.fi-
ware.eu").build();
    }

    // this method posts a subscription request
    private static String postSubscriptionRequest(WebResource service,
int validity) {
        System.out.println("in postSubscriptionRequest(), validity
(in minutes) = " + validity);

        String cqlReq =
            " <contextQL> " +

```

```

        "    <ctxQuery> " +
        "        <action type=\"SUBSCRIBE\" /> " +
        "        <entity>imei|123456789123</entity> " +
        "        <scope>position</scope> " +
        "        <validity>" + validity*60 + "</validity> " +
        // validity in seconds
        "    </ctxQuery> " +
        " </contextQL>";

        // for the SemanticWrapper deployed on the testbed V2 (flod1)
        String callbackUrl =
"http://130.206.82.176:1027/pubsubnotify";

        Form form = new Form();
        form.add("cqlReq", cqlReq);
        form.add("callbackUrl", callbackUrl);

        String response =
service.path("CB/ContextBroker/getContextQL").accept(MediaType.APPLICATION_FORM_URLENCODED).post(String.class, form);

        return response;
    }
}

```

Now, let's have a look at the main fragments of the code:

```

...
import com.sun.jersey.api.client.Client;
...

```

Again we use the Jersey API for developing a REST client.

```
...  
public static void main(String[] args) {  
    System.getProperties().put("proxyHost", "p-  
goodway.rd.francetelecom.fr");  
    System.getProperties().put("proxyPort", "8080");  
...}
```

This is to accomodate network configurations, where http interactions are proxified.

```
...  
WebResource service = client.resource(getBaseURI());  
  
...  
}  
  
private static URI getBaseURI() {  
    return UriBuilder.fromUri("http://pubsub.lab.fi-  
ware.eu").build();  
}
```

This is where the PubSub URL is taken into account, as this URL is a REST point which will allow us to post the subscription. Note that we use here the global instance of the PubSub Context Broker.

```
// this method posts a subscription request  
private static String postSubscriptionRequest(WebResource service,  
int validity) {  
    System.out.println("in postSubscriptionRequest(), validity  
(in minutes) = " + validity);  
  
    String cqlReq =  
        " <contextQL> " +  
        "    <ctxQuery> " +
```

```

"      <action type=\"SUBSCRIBE\" /> " +
"      <entity>imei|123456789123</entity> " +
"      <scope>position</scope> " +
"      <validity>" + validity*60 + "</validity> " +
// validity in seconds
"    </ctxQuery> " +
"  </contextQL>";

...

```

This method defines the content of the subscription message. Note that we make it possible to adjust the lifetime of the subscription through validity parameter.

```

// for the SemanticWrapper deployed on the testbed V2 (flod1)
String callbackUrl =
"http://130.206.82.176:1027/pubsubnotify";

...

form.add("callbackUrl", callbackUrl);

String response =
service.path("CB/ContextBroker/getContextQL").accept(MediaType.APPLICATION_FORM_URLENCODED).post(String.class, form);

```

sets the callback service as the global instance of the semantic extention, and posts it to the PubSub Context Broker.

4.2 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.

4.2.1 End to End testing

This is basically quick testing to check that everything is up and running. It may be composed of a single test or a few of them. E.g.: login on a web site and doing a basic query on a web form or API (provide URL and user/password)

4.2.1.1 **Testing the Virtuoso DBMS server**

If your Virtuoso DBMS server has been installed on a machine which IP address is 103.206.22.70, visit the URL:

<http://103.206.22.70:8890>. If you reach Virtuoso Server web page which should look like the following:



4.2.1.2 **Testing the SemanticWrapper service component**

When starting the SemanticWrapper service, traces are produced on the standard output. Make sure that they

look like these:

```
Buildfile: /home/fyur5281/workspace/SemanticWrapper/build.xml

Main:
    [java] Starting grizzly...
```

```
[java] 13 août 2013 15:03:14
com.sun.jersey.api.core.PackagesResourceConfig init

[java] INFO: Scanning for root resource and provider classes in
the packages:

[java]    fiware.data.orange.semext

[java] 13 août 2013 15:03:14
com.sun.jersey.api.core.ScanningResourceConfig logClasses

[java] INFO: Root resource classes found:

[java]    class fiware.data.orange.semext.JerseyExitService
[java]    class fiware.data.orange.semext.CMLContentSource

[java] 13 août 2013 15:03:14
com.sun.jersey.api.core.ScanningResourceConfig init

[java] INFO: No provider classes found.

[java] 13 août 2013 15:03:14
com.sun.jersey.server.impl.application.WebApplicationImpl _initiate

[java] INFO: Initiating Jersey application, version 'Jersey: 1.17
01/17/2013 03:31 PM'

[java] Jersey app started with WADL available at
http://0.0.0.0:1027/application.wadl

[java] Try out http://0.0.0.0:1027/pubsubnotify

[java]

[java] 13 août 2013 15:03:15
org.glassfish.grizzly.http.server.NetworkListener start

[java] INFO: Started listener bound to [0.0.0.0:1027]

[java] 13 août 2013 15:03:15
org.glassfish.grizzly.http.server.HttpServer start

[java] INFO: [HttpServer] Started.
```

4.2.1.3 *Testing the PSFrontEnd service component*

When starting the PSFrontEnd service, traces are produced on the standard output. Make sure that they look

like these:

```
Buildfile: /home/fyur5281/workspace/PSFrontEnd/build.xml
```



```
JerseyQueryServer:
    [java] Starting grizzly...
    [java] 13 août 2013 15:04:34
com.sun.jersey.api.core.PackagesResourceConfig init
    [java] INFO: Scanning for root resource and provider classes in
the packages:
    [java]   olnc.fiware.pubsub.frontend
    [java] 13 août 2013 15:04:34
com.sun.jersey.api.core.ScanningResourceConfig logClasses
    [java] INFO: Root resource classes found:
    [java]   class olnc.fiware.pubsub.frontend.ExitService
    [java]   class olnc.fiware.pubsub.frontend.JerseyQueryService
    [java]   class olnc.fiware.pubsub.frontend.JerseyExitService
    [java]   class olnc.fiware.pubsub.frontend.QueryService
    [java] 13 août 2013 15:04:34
com.sun.jersey.api.core.ScanningResourceConfig init
    [java] INFO: No provider classes found.
    [java] 13 août 2013 15:04:34
com.sun.jersey.server.impl.application.WebApplicationImpl _initiate
    [java] INFO: Initiating Jersey application, version 'Jersey: 1.17
01/17/2013 03:31 PM'
    [java] Jersey app started with WADL available at
http://0.0.0.0:9000/application.wadl
    [java] Try out http://0.0.0.0:9000/queryservice
    [java]
    [java] 13 août 2013 15:04:35
org.glassfish.grizzly.http.server.NetworkListener start
    [java] INFO: Started listener bound to [0.0.0.0:9000]
    [java] 13 août 2013 15:04:35
org.glassfish.grizzly.http.server.HttpServer start
    [java] INFO: [HttpServer] Started.
```

4.2.2 List of Running Processes

There are three processes that should be up and running. Here is the info that a “ps -aux” command should return:

```
...
root      8098  0.0 10.5 788956 217244 ?        Sl      2013   56:30
virtuoso-t -f
...

...

root      28381  0.7  1.7 1498172 35008 pts/1    Tl      18:11    0:00
/usr/bin/java -classpath /usr/share/ant/lib/ant-
launcher.jar:/usr/share/java/xmlParserAPIs.jar:/usr/share/ja
root      28394  1.4  2.8 1552412 58736 pts/1    Tl      18:11    0:01
/usr/lib/jvm/java-6-openjdk-amd64/jre/bin/java -classpath
/root/PSFrontEnd/bin:/root/PSFrontEnd/lib/cxf-2.5.
...

...

root      28432  1.8  1.6 1498172 34780 pts/1    Sl      18:12    0:00
/usr/bin/java -classpath /usr/share/ant/lib/ant-
launcher.jar:/usr/share/java/xmlParserAPIs.jar:/usr/share/ja
root      28445  3.9  2.5 1502560 53124 pts/1    Sl      18:12    0:02
/usr/lib/jvm/java-6-openjdk-amd64/jre/bin/java -classpath
/root/SemanticWrapper/bin:/root/SemanticWrapper/li
...
```

4.2.3 Network interfaces Up & Open

Here are the list of ports that should be open and in use. All of them will handle TCP traffic.

- port 8890
- port 1027
- port 9000

4.2.4 Databases

This enabler uses a Virtuoso database server. The testing procedure is described in the section "Testing the Virtuoso DBMS server" above.

4.3 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.

4.3.1 Resource availability

In order to have a healthy enabler, the minimum amount of available resources are:

- 2GB of RAM
- 5GB of hard disk

Bellow these thresholds the enabler is likely to experience problems or bad performance.

4.3.2 Remote Service Access

The PubSub Semantic Extension enabler is a client of the CAP Context Broker (called Context Consumer (CC), in the CAP terminology). However, The binding of the PubSub semantic extension to the CAP Context Broker is performed outside the enabler itself, as explained in the Installation and Administration guide, thus the parameters of the Context Broker is not hardwired coded nor stored by the enabler.

4.3.3 Resource consumption

In a normal working state the PubSub Semantic Extension enabler process uses around 300Mb to 500Mb of memory and shouldn't require more than 15% of the CPU.

4.3.4 I/O flows

The Semantic Extension uses the following flows:

- from the SPARQL clients to the Semantic Extension: TCP port 9000
- from the CAP Context Broker to the Semantic Wrapper: TCP port 1027

The throughput in these flows depends on the amount of connections between the CAP Context Broker (CB) and its Context Providers (CPs). However, we can assume that the throughput in this three different flows are equivalent.

5 CEP GE - IBM Proactive Technology Online Installation and Administration Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

5.1 CEP GE

The CEP GE is implemented by IBM Proactive Technology Online (a.k.a Proton). The Proton runtime engine detects patterns on incoming events and the Proton authoring tool is a web based user interface in which CEP applications, also known as Event Processing Networks, can be defined and deployed to an engine.

5.1.1 Prerequisites

IBM Proactive Technology Online is a standard web application. It requires:

1. Java SE 6 or later installed
2. Apache Tomcat 7 installed (Was tested on apache-tomcat-7.0.26. Currently the only tested web server.)

5.1.1.1 *Setup Apache Tomcat for management*

- On Linux make sure CATALINA_HOME is defined as an environment variable pointing to the Apache Tomcat directory (e.g. /opt/apache-tomcat-7.0.26)
- Configure manager access to application. Instructions are available here [Configuring Manager Application Access](#)

As a suggested reference, have the file under the Apache Tomcat directory **./conf/tomcat-users.xml** include the following:

```
<tomcat-users>
  <role rolename="manager-gui" />
  <role rolename="manager-status" />
  <role rolename="manager-script" />
  <role rolename="manager-jmx" />
  <user username="manager" password="manager" roles="manager-
gui,manager-status,manager-script,manager-jmx" />
  <role rolename="admin-gui" />
  <user username="admin" password="admin" roles="admin-gui" />
</tomcat-users>
```

```
</tomcat-users>
```

- Enable JMX access on Apache Tomcat. Instructions are available here [Enabling JMX Remote](#)

As a suggested reference on:

- Windows

Add the following to the file located in the Apache Tomcat directory **./bin/startup.bat**

```
set CATALINA_OPTS=-Dcom.sun.management.jmxremote -  
Dcom.sun.management.jmxremote.port=8686 -  
Dcom.sun.management.jmxremote.ssl=false -  
Dcom.sun.management.jmxremote.authenticate=false  
  
set JRE_HOME={java_install_dir}\Java\jre6
```

- Linux

Add the following to the file located in the Apache Tomcat directory **./bin/startup.sh**

```
CATALINA_OPTS="-Dcom.sun.management.jmxremote -  
Dcom.sun.management.jmxremote.port=8686 -  
Dcom.sun.management.jmxremote.ssl=false -  
Dcom.sun.management.jmxremote.authenticate=false"  
  
JRE_HOME={java_install_dir}/jre
```

Note: The actual **jmxremote.port** number maybe different. The Apache Tomcat default JMX port number is 8686. Remember this value as you will need to use it in [configuring the Proton administrator application](#).

5.1.2 Installation

The installation package contains four archived web applications and a sample folder.

Deploy the four war files (ProtonOnWebServerAdmin.war, ProtonOnWebServer.war, AuthoringTool.war, AuthoringToolWebServer.war) to the Apache Tomcat server. Instructions on how to deploy applications to an Apache Tomcat server can be found at [How To Deploy](#). As a suggested reference you may drop the four war files in the Apache Tomcat installation directory under **./webapps** while the server is running. They should be deployed automatically by the server soon after.

Note: you will need to perform the [configuration](#) instructions before IBM Proactive Technology Online will work as expected.

5.1.2.1 *Installing a new Proton instance*

1. Rename **ProtonOnWebServer.war** provided by the install package
2. Deploy the renamed war on to the Apache Tomcat server
3. Follow the configuration instructions for an engine instance [Configuring an engine instance](#)
4. You may need to restart the application

5.1.3 Configuration

5.1.3.1 *Configuring the administration application*

Proton administration application is responsible for the management of the definitions, event processing networks, repository (add, update, delete a definition) and the multiple Proton instances (update definition of an instance, retrieve an instance state and start\stop and instance). The prerequisite installation instructions of setting up Apache Tomcat for management is crucial for the appropriate functioning of the admin application. After successful deployment of the admin application the following must be configured in the **ProtonAdmin.properties** file located under the Apache Tomcat directory in **./webapps/ProtonOnWebServerAdmin**.

- Location of the definitions repository (make sure there are relevant credentials for read\write access)

```
definitions-repository={directory_of_choice}
```

- Apache Tomcat manager authentication details according to the suggested reference in [Setup Apache Tomcat for management](#)

```
manager-username=manager  
manager-password=manager
```

- Server port number of the Apache Tomcat server (default for Apache Tomcat is **8080**)

```
tomcat-server-port={port}
```

- JMX services port of the Apache Tomcat server (default for Apache Tomcat is **8686** and should be the same as for the **jmxremote.port** property set in [Setup Apache Tomcat for management](#))

```
tomcat-jmx-port={port}
```

5.1.3.2 *Configuring an engine instance*

There are two files for configuring an engine instance. Both files are located in the Apache Tomcat directory under **./webapps/{instance_name}** where **{instance_name}** is an identifier of a Proton instance, e.g. **ProtonOnWebServer**.

- The file called **Proton.properties** contains two port properties for the input and output adapters. Each engine instance should have a different value for these properties. The other properties should not be manually configured.

- In the file called logging.properties

5.1.3.3 **Configuring the authoring tool**

No configuration is required

5.1.3.4 **Configuring input and output adapters**

Instructions on how to define the input and output adapters to receive raw events and send derived events are given in the programmer guide [IBM Proactive Technology Online Programmer Guide](#). The sample application that is provided with the installation uses file input and output adapters that are already defined and are ready for a sanity check.

5.1.4 **Running**

Once the Prerequisites, Installation and Configurations instructions have been completed, IBM Proactive Technology Online should be up and running. In general, starting up the Apache Tomcat server and starting the applications (AuthoringTool, AuthoringToolWebServer, ProtonOnWebServerAdmin and Proton engine instances) constitutes a running product.

- Accessing the authoring tool is through the following link (after completing the host and port values) <http://{host}:{port}/AuthoringTool/Main.html>
- Administrating the product and pushing events to the engine instances is described in [Complex Event Processing Open RESTful API Specification \(PRELIMINARY\)](#)

5.2 **Sanity check procedures**

5.2.1 **End to End testing**

To verify that IBM Proactive Technology Online is running:

1. Access the Apache Tomcat administrator tool – <http://{host}:{port}/manager> and log in with user and password you configured in [Setup Apache Tomcat for management](#). Identify that all applications are installed and running (AuthoringTool, AuthoringToolWebServer, ProtonOnWebServerAdmin and all Proton engine instances, e.g. ProtonOnWebServer).
2. Access the authoring tool - <http://{host}:{port}/AuthoringTool/Main.html> (tested on Google Chrome)
3. [Run the sample](#)

Run the sample

1. Copy the sample directory included in the install package to the root directory of the Apache Tomcat installation

2. Stop and then restart the ProtonOnWebServer application (you may use the Apache Tomcat manager tool or the Admin REST APIs offered by the GE).
3. Observe the files that were generated in the sample directory. Expected results are as follows:
 - DoSAAttack_TrafficReport.txt -> a file representing a consumer for the **TrafficReport** events, the raw events.

```
Name=TrafficReport;Certainty=0.0;Chronon=null;Cost=0.0;Annotation=;DetectionTime=1342101349469;Duration=0.0;EventSource=;EventId=4d41820d-6147-4b64-82cb-3e66649cd748;volume=1000;ExpirationTime=null;OccurrenceTime=null;

Name=TrafficReport;Certainty=0.0;Chronon=null;Cost=0.0;Annotation=;DetectionTime=1342101349469;Duration=0.0;EventSource=;EventId=0ea035f1-6c17-4900-83b0-c7f7a3efb43e;volume=1600;ExpirationTime=null;OccurrenceTime=null;

Name=TrafficReport;Certainty=0.0;Chronon=null;Cost=0.0;Annotation=;DetectionTime=1342101349469;Duration=0.0;EventSource=;EventId=589c2533-4ed7-451f-8ca1-a67cf7b80909;volume=2500;ExpirationTime=null;OccurrenceTime=null;
```

- DoSAAttack_SteepDemand.txt --> a file representing a consumer for the **SteepDemand** derived events that are generated during the test execution

```
Name=SteepDemand;DetectionTime=1342360040630;Duration=0.0;EventSource=;EventId=bd18fadf-db97-48e1-a2ad-ce5bbcf40652;Certainty=0.0;Chronon=null;ExpirationTime=null;Cost=10.0;OccurrenceTime=1342360040630;Annotation=;
```

Note: the sample file (DoSAAttack.json, located in the sample folder) contains paths for the input event file and for the output files. If you run on linux, you should change the 3 relative paths in this file to ".\sample\" instead of "../sample/"

5.2.2 List of Running Processes

- Apache Tomcat server running as a Java process.

5.2.3 Network interfaces Up & Open

- Apache Tomcat server and JMX ports are used (default 8080 and 8686 respectively)
- Each Proton engine instance uses two ports, one for input and one for output adapters as configured in [Configuring an engine instance](#). The single instance called **ProtonOnWebServer** provided with the base installation has the following ports as default 3000, 3300.

5.2.4 Databases

No database is used in this release

5.3 Diagnosis Procedures

IBM Proactive Technology Online logs with Apache Tomcat logging. The log files are located in the Apache Tomcat directory **./logs**.

5.3.1 Resource availability

- The required RAM is dependent on the event processing patterns defined by the event processing network and by the size and number of events that need to be held on to for detecting the patterns. Fortunately, a basic box available off-the-shelf is sufficient for most of the applications.
- Usually the disk size required during run time is negligible, unless the application uses adapters of type "File" and the input files or the generated output files are very big.

5.3.2 Remote Service Access

Currently the CEP GE has no built-in integration with other GEs.

5.3.3 Resource consumption

The resource consumption is highly dependent on the defined CEP application and on the event streams that are processed. There are no typical numbers.

5.3.4 I/O flows

- Ports 8080 and 8686 for administrating IBM Proactive Technology Online and for working against the authoring tool.
- Input (e.g. 3000) and Output (e.g. 3300) ports configured for engine instances see [Configuring an engine instance](#) are used in receiving and notifying on events and integrating with other systems as producers and consumers of events. Most traffic will be observed on these ports.

6 BigData Analysis - Installation and Administration Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

6.1 Introduction

This document details how to install, configure and administrate the Big Data platform.

6.2 Platform installation

6.2.1 Requirements

6.2.1.1 *Operating system requirements*

Whichever 64 bits Linux distribution can be used while supporting OpenVZ and Puppet (see Dependencies section), but CentOS 6.4 is recommended.

6.2.1.2 *Resource requirements*

Cosmos platform requires the following 3 machines (minimum):

- The master node machine, supporting the APIs and all the management logic of the platform.
- The Infinity namenode machine, supporting the naming of all the files within the permanent storage of Cosmos.
- The Infinity datanode machine, supporting the real data of the files managed by the Infinity namenode machine.

In a production deployment, the Infinity datanode should be expanded to more machines, at least as many machines as the replication factor of HDFS (otherwise, such replication factor has no sense at all).

In addition, at least 2 slave machines are required, in order to host the cluster's nodes. The exact number of slave machines depends on your needs, the many slaves you deploy, the many clusters you will be able to create. With 2 slave machines you'll be able to serve exactly a unique user at the same time.

These machines may be physical (recommended) or virtualized on top on some hipervisor-based virtualization tool such as [VirtualBox](#) or [VMWare](#). Anyway, the following hardware is recommended:

- 32 MB RAM
- At least 10 GB HDD

When formatting the file system of the machines (except the one acting as master node), take into account that a specific partition for OpenVZ (/vz) is desirable (format it to ext4).

6.2.2 Dependencies

In addition to the common tools not present in a fresh installation of CentOS (wget, unzip, git, etc.), the installation basically depends on Puppet and OpenVZ. In addition, you will have to create your own local repos for Cosmos if you do not have access to public ones.

6.2.2.1 *Puppet*

In order to install Puppet in all the machines, just add the Puppet repository:

```
$ sudo rpm -ivh
http://yum.puppetlabs.com/el/6/products/i386/puppetlabs-release-6-7.noarch.rpm
```

and install the package:

```
$ sudo yum install puppet
```

6.2.2.2 *OpenVZ*

The OpenVZ kernel must be installed in all the machines except the master one.

In order to proceed, manually download and install vzkernel-firmware and vzkernel version 2.6.32-042stab079.6 (this version is **mandatory**; it is highly recommended not to install the OpenVZ repository in this step) from <https://openvz.org/Download/kernel>:

```
$ wget http://download.openvz.org/kernel/branches/rhel6-2.6.32/042stab079.6/vzkernel-firmware-2.6.32-042stab079.6.noarch.rpm
$ wget http://download.openvz.org/kernel/branches/rhel6-2.6.32/042stab079.6/vzkernel-2.6.32-042stab079.6.x86\_64.rpm
$ sudo rpm -Uvh vzkernel-firmware-2.6.32-042stab079.6.noarch.rpm
$ sudo rpm -Uvh vzkernel-2.6.32-042stab079.6.x86_64.rpm
```

Now is the time to install the OpenVZ repository:

```
$ wget -P /etc/yum.repos.d/ http://ftp.openvz.org/openvz.repo
$ rpm --import http://ftp.openvz.org/RPM-GPG-Key-OpenVZ
```

After installing, disable SELinux and reboot the machine in order the installation applies. **This step is recommended for the master node** as well.

```
$ sudo echo "SELINUX=disabled" > /etc/sysconfig/selinux
$ sudo shutdown -r now
```

The rest of the steps in a typical OpenVZ installation [1] are automatically performed by the Puppet scripts.

6.2.2.3 *Create your own Cosmos repos*

You will need to create a yum repo for (a modified version of) Ambari, the Hadoop deployment manager. In addition, other files must be downloaded and put into a well-known location in order the Puppet scripts can find them.

1. Create the following repo structure in a Linux box:

```
$ mkdir /path/to/repos/cosmos/java/  
$ mkdir /path/to/repos/cosmos/ambari/
```

2. Install, if not yet installed, the createrepo tool:

```
$ sudo yum install createrepo (RedHat/CentOS)  
$ sudo apt-get install createrepo (Debian/Ubuntu)
```

3. Java repo:

Download the jdk-6u31-linux-x64.bin and jce_policy-6.zip files from [Oracle](http://www.oracle.com/technetwork/java/javase-downloads-138443.html) and put them into /path/to/repos/cosmos/java.

4. Ambari repo:

Download the ambari-* files and the RPM-GPG-KEY-Jenkins file and put them in the above Ambari directory:

```
$ cd /path/to/repo/cosmos/ambari/  
$ wget https://forge.fi-ware.org/frs/download.php/1322/ambari-server-0.16.0-rpm.rpm  
$ wget https://forge.fi-ware.org/frs/download.php/1323/ambari-agent-0.16.0-rpm.rpm  
$ wget https://forge.fi-ware.org/frs/download.php/1324/ambari-log4j-0.16.0-rpm.rpm  
$ wget https://forge.fi-ware.org/frs/download.php/1326/RPM-GPG-KEY-Jenkins.txt
```

Create the Ambari repo

```
$ createrepo /path/to/repos/cosmos/ambari/
```

Check a repodata folder has been created (/path/to/repos/cosmos/ambari/repodata/).

6.2.3 Installation and configuration

Let's suppose the following machines configuration:

- Master node

- Hostname: cosmos-master
 - IP address: ip1
- Infinity namenode
 - Hostname: machine02
 - IP address: ip2
- Infinity datanode
 - Hostname: machine03
 - IP address: ip3
- Slave 1
 - Hostname: machine04
 - IP address: ip4
- Slave 2
 - Hostname: machine05
 - IP address: ip5
- Slave N
 - Hostname: machine(3+N)
 - IP address: ip(3+N)

All the machines are in the "my_domain" domain and in the "my_subnet/my_netmask" subnet. There is a gateway which IP address is "my_gateway".

It is highly recommended the hostname resolution is added to /etc/hosts in all the machines.

The installation is automated thanks to Puppet:

6.2.3.1 **Step 1: Cosmos platform installation software download**

Log into the machine acting as Master node, download [this](#) ZIP file into a temporal folder (COSMOS_TMP_PATH) and uncompress it:

```
$ mkdir [COSMOS_TMP_PATH]
$ cd [COSMOS_TMP_PATH]
$ wget https://forge.fi-ware.org/frs/download.php/1320/cosmos-
platform_2.10-0.16.0.zip
$ unzip cosmos-platform_2.10-0.16.0.zip
```

Once unzipped, you will find two subfolders, `[COSMOS_TMP_PATH]/puppet` and `[COSMOS_TMP_PATH]/rpms`. The first one contains all the necessary scripts to deploy the software, and it must be copied to all the machines (it is recommended to be copied once the configuration has been set up in the master, thus such configuration can be replicated in the slaves). The second subfolder contains the RPM packages necessary to install the Master node, and is not necessary to be copied to the Slave nodes.

6.2.3.2 **Step 2: platform environment creation**

Create the following folders in all the machines:

```
$ mkdir  
[COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/hieradata/<my_environment>  
  
$ mkdir -p  
[COSMOS_TMP_PATH]/puppet/modules/cosmos/files/environments/<my_environment>/certs
```

An environment is a specific Cosmos configuration, describing each one a different way on deploying the platform. You may have configured as many environments as you wish, but only one environment may be applied on the infrastructure. `<my_environment>` is just a name of your choice.

6.2.3.3 **Step 3: generating and installing the master node's certificate**

The platform requires a certificate for the master node, signed by a valid CA, is installed in order to be shown as an authentication proof. Thus, this certificate must be created by generating a Certificate Signing Request (CSR); do it once in the master node:

```
$ openssl req -newkey rsa:2048 -new -keyout newkey.pem -out  
newreq.pem
```

The above command will prompt for certain information; the most important information regarding the Cosmos platform is the name of the server (whichever hostname was chosen for the cosmos master node) where the certificate is going to be installed, and that the challenge password must be empty. Although the PEM pass phrase must be empty (otherwise, the httpd server will not be automatically started), it has to be filled in this step and removed later by performing:

```
$ openssl rsa -in newkey.pem -out newkey.pem
```

Reached this point, you may choose among two options for signing the certificate:

- Use a valid CA in the Internet. The content of the generated SCR (`newreq.pem` file) must be used within the CA in order to retrieve the final certificate, which will be typically called `certnew.cer`. The way each CA manages the CSR varies from one to another.
- Self-signing the certificate. In this case, you have to perform this command:

```
$ openssl req -new -x509 -key newkey.pem -out certnew.cer
```

In any case, once the certificate (`certnew.cer`), key (`newkey.pem`) and CSR (`newreq.pem`) have been got, rename the files according to this (do it in all the machines):

```
$ cp newkey.pem
[COSMOS_TMP_PATH]/puppet/modules/cosmos/files/environments/<my_environment>/certs/<cosmos-master-node>_key.pem

$ cp cernew.cer
[COSMOS_TMP_PATH]/puppet/modules/cosmos/files/environments/<my_environment>/certs/<cosmos-master-node>_cer.pem

$ cp newreq.pem
[COSMOS_TMP_PATH]/puppet/modules/cosmos/files/environments/<my_environment>/certs/<cosmos-master-node>_req.pem
```

6.2.3.4 **Step 4: CA's certificate installation**

The CA's certificate itself must be installed. Download it from the appropriate link (if you self-signed the master node's certificates, then such certificate is the CA's certificate as well) and do the following in the Cosmos master node:

Copy the CA's certificate (generic name `<ca_cert>.pem`) to the local certificates store and change directory to it:

```
$ mv <ca_cert>.pem /etc/pki/tls/certs
$ cd /etc/pki/tls/certs
```

Create a symbolic link for the CA's certificate. An 8-digit-number-based file will be created. It is very important the extension of this file is '.0':

```
$ ln -s <ca_cert>.pem `openssl x509 -hash -noout -in <ca_cert>.pem`.0
```

Verify the certificate has been successfully installed:

```
$ openssl verify -CApath /etc/pki/tls/certs <ca_cert>.pem
xxxxxxx.0: OK
```

You must see a 8-digit hash .0 file followed by "OK".

6.2.3.5 **Step 5: creating the containers identity**

In OpenVZ terminology, a container is an isolated Linux box within the host. It may be considered as a "virtual machine", although it is not.

For each slave, its SSH identity is stored in the default file `/etc/ssh/ssh_host_rsa_key(.pub)`. Something similar must be generated for the container (no passphrase must be used):

```
$ ssh-keygen -t rsa
```

```

Generating public/private rsa key pair.

Enter file in which to save the key (/root/.ssh/id_rsa):
/etc/ssh/ssh_ct_rsa_key

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /etc/ssh/id_rsa.
Your public key has been saved in /etc/ssh/id_rsa.pub.
The key fingerprint is:
91:6e:e3:c4:62:53:6b:2a:cb:fb:e3:8c:5f:bc:c9:c3 root@cosmos1.hi.inet

The key's randomart image is:
+--[ RSA 2048]-----+
|                      |
|      .               |
|      +               |
|      + o             |
|      + S             |
|      . X .           |
|      . ..+           |
|      . =.oEo          |
|      ===.+           |
+-----+

```

Now, copy all the slave and containers SSH private keys to the environment folder of all the machines:

```

$ cp /etc/ssh/ssh_host_rsa_key
[COSMOS_TMP_PATH]/puppet/modules/cosmos/files/environments/<my_environment>/machine0i_key

$ cp /etc/ssh/ssh_ct_rsa_key
[COSMOS_TMP_PATH]/puppet/modules/cosmos/files/environments/<my_environment>/machine0i_ct_key

```

6.2.3.6 **Step 6: Puppet configuration**

Proceed to configure the following Puppet files in all the machines (please observe there is not a machine01.yaml file since it is the master node, not a slave). This files must be created from the scratch

unless templated from a packaged environment (typically called "andromeda" within the `cosmos-platform` zip file).

Do it only once in the master node and then copy the `[COSMOS_TMP_PATH]/puppet` folder in all the slaves. Do not copy `[COSMOS_TMP_PATH]/rpms` since they are only needed in the master node.

Red parameters must be configured, the other ones may have the specified default values.

- `[COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/hieradata/<my_environment>/common.yaml`

```
#
[COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/hieradata/<my_environment>/common.yaml
# modified Ambari repo, should not be changed
ambari::params::repo_url: 'http://130.206.81.65/cosmos/ambari/'
# Hortonworks repo, should not be changed
ambari::params::hdp_stack_repo_url: 'http://public-repo-1.hortonworks.com/HDP/centos6/'
# do not change
ambari::params::cosmos_stack_repo_url:
'http://%{cosmos::params::master\_ip}:%{cosmos::params::master\_repo\_port}/'
# JDK binary, should not be changed
ambari::params::jdk_url: 'http://130.206.81.65/cosmos/java/jdk-6u31-linux-x64.bin'
# JCE binay, should not be changed
ambari::params::jce_url: 'http://130.206.81.65/cosmos/java/jce\_policy-6.zip'
# Hortonworks utils repo, should not be changed
ambari::params::hdp_utils_repo_url: 'http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.16/repos/centos6/'
# Hortonworks GPG key, should not be changed
ambari::params::hdp_utils_gpg_url:
'http://130.206.81.65/cosmos/ambari/RPM-GPG-KEY-Jenkins'
# put your region for timezone configuration
timezone::region: 'Europe'
# put your city for timezone configuration
```

```
timezone::locality: 'Madrid'

# EPEL repo, should not be changed
yum::params::yum_epel:
'http://dl.fedoraproject.org/pub/epel/6/x86\_64/'

# CentOS repo, should not be changed
yum::params::yum_centos:
'http://mirror.centos.org/centos/6/os/x86\_64/'

# CentOS updates repo, should not be changed
yum::params::yum_centos_updates:
'http://mirror.centos.org/centos/6/updates/x86\_64/'

# not used
yum::params::yum_redhat: ""

# not used
yum::params::yum_redhat_optional: ""

# Puppet repo, should not be changed
yum::params::yum_puppet:
'http://yum.puppetlabs.com/el/6/products/x86\_64/'

# Puppet dependencies repo, should not be changed
yum::params::yum_puppet_deps:
'http://yum.puppetlabs.com/el/6/dependencies/x86\_64/'

# Cosmos dependencies, should not be changed
cosmos::params::cosmos_repo_deps_url:
'http://130.206.81.65/cosmos/rpms/cosmos-deps/'

# OpenVZ templates, should not be changed
cosmos::openvz::images::base_image_url:
'http://130.206.81.65/cosmos/repos/ovz-templates'

# CentOS image to be used, do not change
cosmos::openvz::images::image_name: 'centos-6-cosmos.HDP.2.0.6-20140123-x86_64.tar.gz'

# put the subnet your machines are
cosmos::params::cosmos_subnet: 'my_subnet'

# put the mask for the above subnet
cosmos::params::cosmos_netmask: '255.255.0.0'
```

```
# do not change
cosmos::params::cosmos_ssl_cert_source:
"puppet:///modules/cosmos/environments/{environment}/certs/{hostname}
 Cer.pem"

# do not change
cosmos::params::cosmos_ssl_key_source:
"puppet:///modules/cosmos/environments/{environment}/certs/{hostname}
 Key.pem"

# put the gateway IP address for your subnet
cosmos::slave::gateway: 'my_gateway'

# should be the same than the above
cosmos::slave::ct_gateway: 'my_gateway'

# do not change
cosmos::slave::host_key_priv_file:
"puppet:///modules/cosmos/environments/{environment}/{hostname}_key"

# do not change
cosmos::slave::ct_key_priv_file:
"puppet:///modules/cosmos/environments/{environment}/{hostname}_ct_k
ey"

# put the domain your machines are
cosmos::params::domain: '.my_domain'

# put the master node IP address
cosmos::params::master_ip: 'ip1'

# for future usage, do not change
cosmos::params::master_repo_port: '8081'

# put the master node hostname
cosmos::params::master_hostname: 'cosmos-master'

# do not change
cosmos::params::pdihub_client_id: '20f0381a6698c19953d5'

# do not change
cosmos::params::pdihub_client_secret:
'b11b73f83e7dcf940fec3356d3d0604e3169ab98'

# do not change
```

```
cosmos::params::horizon_password: 'horizon!'

# configure properly with the CA provider (the one configured when
creating the master node's certificate)

cosmos::params::ssl_authority: 'CA_provider'

# FAQ-related information, configure properly with the URL where the
CA may be accessed; if using self-signed certificates, just put a link
to the certificate itself

cosmos::params::ssl_cert_location: 'CA_URL'

# FAQ-related information, configure properly with the support team
name

cosmos::params::ssl_support_name: 'support'

# FAQ-related information, configure properly with the support email

cosmos::params::ssl_support_email: 'support@your_organization'

# should not be changed

cosmos::params::ambari_mr_app_master_memory: 1024
cosmos::params::ambari_map_task_memory: 2048
cosmos::params::ambari_map_heap_memory: 1536
cosmos::params::ambari_reduce_task_memory: 3072
cosmos::params::ambari_reduce_heap_memory: 2304
cosmos::params::ambari_yarn_total_memory: 4096
cosmos::params::ambari_yarn_container_min_memory: 128
cosmos::params::ambari_yarn_virtual_physical_memory_ratio: 2.1

# put the network interface name

cosmos::openvz::network::host_iface: 'eth0'

# put the list of slave machines

slave_hosts:
  - 'machine02'
  - 'machine03'
  - ...
```

- [COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/hieradata/<my_environment>/machine02.yaml

- [COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/hieradata/<my_environment>/machine03.yaml
- [COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/hieradata/<my_environment>/machineN.yaml

```
#
[COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/hieradata/<my_environment>/machine0i.yaml
# do not change
cosmos::slave::rack: "rack01"
# put the slave IP address
cosmos::slave::ip: "ip_i"
# put the container IP address
cosmos::slave::ct_ip: "ip_ct_i"
# put the the hostname the container will have, typically
"my_environment-machine0i"
cosmos::slave::ct_hostname: "my_environment-machine0i"
# put the hardware profile, "hdfs-master" for the Infiity namenode,
"hdfs-slave" for the Inifinity datanode, "gl-compute" for the rest
cosmos::slave::hardware_profile: gl-compute
# put the slave hostname
cosmos::slave::name: "machine0i"
# put the RAM memory of the slave
cosmos::slave::ram: 32768
# put the SSH identity of the slave, i.e.
/etc/ssh/ssh_host_rsa_key.pub
cosmos::slave::host_key_pub: "ssh-rsa AAAAB3NzaC1...RHqPbcBQbL"
# put the SSH identity of the container, i.e.
/etc/ssh/ssh_ct_rsa_key.pub
cosmos::slave::ct_key_pub: "ssh-rsa AAAAB3NzaC1...zMdallH/Sz"
```

and

- [COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/nodes/<my_environment>.pp

```
# do not change
node 'cosmos-master' {
```

```

    include profiles::master
}

# put all the slave machines
node 'machine02' {
    include profiles::slave
}

node 'machine03' {
    include profiles::slave
}

...

```

Regarding the slaves, a regular expression may be used as well:

```

node /^machine(0[12]|[^0].)$/ {
    include profiles::slave
}

```

6.2.3.7 **Step 7: applying Puppet**

Start with the slaves, and apply Puppet in the master node at the end.

Slaves command, executed from [COSMOS_TMP_PATH]:

```

$ sudo puppet apply --debug --verbose \
    --modulepath
[COSMOS_TMP_PATH]/puppet/modules/:[COSMOS_TMP_PATH]/puppet/modules_thi
rd_party/ \
    --environment <my_environment> --hiera_config
[COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/hiera.yaml \
    --manifestdir [COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/
[COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/site.pp \
    > puppet.out 2> puppet.err

```

Please observe the connection with the slave is lost, and that it is only reachable from the master node (the IP address of the slave is now binded to the OpenVZ bridging interface).

Master node command (the same then above, but adding the *rpms* folder to the *modulepath* option), executed from [COSMOS_TMP_PATH]:

```
$ sudo puppet apply --debug --verbose \  
    --modulepath  
[COSMOS_TMP_PATH]/puppet/modules/:[COSMOS_TMP_PATH]/puppet/modules_thi  
rd_party/:[COSMOS_TMP_PATH]/rpms/ \  
    --environment <my_envionment> --hiera_config  
[COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/hiera.yaml \  
    --manifestdir [COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/  
[COSMOS_TMP_PATH]/puppet/modules/cosmos/manifests/site.pp \  
    > puppet.out 2> puppet.err
```

6.3 Cygnus installation

6.3.1 Requirements

There is no specific requirements in terms of CPU, memory or disk. Just in case, the Cygnus requirements are the same than the machine hosting the Orion Context Broker, due to Cygnus is usually installed and run in such machine.

6.3.2 Dependencies

Maven (and thus Java SDK, since Maven is a Java tool) is needed in order to install and run the injector.

In order to install Java SDK (not JRE), just type (CentOS machines):

```
$ yum install java-1.6.0-openjdk-devel
```

Remember to export the JAVA_HOME environment variable.

```
$ export JAVA_HOME=...
```

In order to do it permanently, edit /root/.bash_profile (root user) or /etc/profile (other users).

Maven is installed by downloading it from <http://maven.apache.org/download.cgi>. Install it in a folder of your choice (represented by APACHE_MAVEN_HOME):

```
$ wget http://www.eu.apache.org/dist/maven/maven-3/3.2.1/binaries/apache-maven-3.2.1-bin.tar.gz  
$ tar xzvf apache-maven-3.2.1-bin.tar.gz  
$ mv apache-maven-3.2.1-bin APACHE_MAVEN_HOME
```

6.3.3 Installation and configuration

6.3.3.1 *Step 1: Cygnus installation*

Apache Flume can be easily installed by downloading its latests version from <http://flume.apache.org/download.html>. Move the untared directory to a folder of your choice (represented by `APACHE_FLUME_HOME`):

```
$ wget http://www.eu.apache.org/dist/flume/1.4.0/apache-flume-1.4.0-bin.tar.gz
$ tar xvzf apache-flume-1.4.0-bin.tar.gz
$ mv apache-flume-1.4.0-bin APACHE_FLUME_HOME
```

Then, the developed classes must be packaged in a Java jar file which must be added to the `APACHE_FLUME_HOME/lib` directory:

```
$ git clone https://github.com/telefonicaid/fiware-connectors.git
$ git checkout release/0.1
$ cd fiware-connectors/flume
$ APACHE_MAVEN_HOME/bin/mvn package
$ cp target/cosmos-injector-1.0-SNAPSHOT.jar APACHE_FLUME_HOME/lib
```

Please observe the cosmos-injector code has been built using the Flume provided versions of `httpcomponents-core` and `httpcomponents-client` (4.2.1). These are not the newest versions of such packages, but trying to build the cosmos-injector with such newest libraries has shown incompatibilities with Flume's ones. cosmos-injector configuration

6.3.3.2 *Step 2: Cygnus configuration*

The typical configuration when using the HTTP source, the OrionRestHandler, the MemoryChannel and the OrionHDFSink is shown below:

```
# APACHE_FLUME_HOME/conf/cosmos-injector.conf

orionagent.sources = http-source
orionagent.sinks = hdfs-sink
orionagent.channels = notifications

# Flume source, must not be changed
orionagent.sources.http-source.type =
org.apache.flume.source.http.HTTPSource
```



```
# channel name where to write the notification events
orionagent.sources.http-source.channels = notifications

# listening port the Flume source will use for receiving incoming
notifications

orionagent.sources.http-source.port = 5050

# Flume handler that will parse the notifications, must not be
changed

orionagent.sources.http-source.handler =
es.tid.fiware.orionconnectors.cosmosinjector.OrionRestHandler

# regular expression for the orion version the notifications will
have in their headers

orionagent.sources.http-source.handler.orion_version = 0\.10\.*

# URL target
orionagent.sources.http-source.handler.notification_target = /notify


# channel name from where to read notification events
orionagent.sinks.hdfs-sink.channel = notifications

# Flume sink that will process and persist in HDFS the notification
events, must not be changed

orionagent.sinks.hdfs-sink.type =
es.tid.fiware.orionconnectors.cosmosinjector.OrionHDFSSink

# IP address of the Cosmos deployment where the notification events
will be persisted

orionagent.sinks.hdfs-sink.cosmos_host = x.y.z.w

# port of the Cosmos service listening for persistence operations;
14000 for httpfs, 50070 for webhdfs and free choice for inifinty

orionagent.sinks.hdfs-sink.cosmos_port = 14000

# username allowed to write in HDFS (/user/myusername)
orionagent.sinks.hdfs-sink.cosmos_username = myusername

# dataset where to persist the data
(/user/myusername/path/to/my/dataset)

orionagent.sinks.hdfs-sink.cosmos_dataset = path/to/my/dataset

# HDFS backend type (webhdfs, httpfs or infinity)
```

```

orionagent.sinks.hdfs-sink.hdfs_api = httpfs

# channel name

orionagent.channels.notifications.type = memory

# capacity of the channel

orionagent.channels.notifications.capacity = 1000

# amount of bytes that can be sent per transaction

orionagent.channels.notifications.transactionCapacity = 100

```

6.3.3.3 *Step 3: log4j configuration*

The injector uses the log4j facilities added by Flume for logging purposes. You can maintain the default `APACHE_FLUME_HOME/conf/log4j.properties` file, where a console and a file appender are defined (in addition, the console is used by default), or customize it by adding new appenders. Typically, you will have several instances of the cosmos-injector running; they will be listening on different TCP ports for incoming `notifyContextRequest` and you'll probably want to have different log files for them. E.g., if you have two Flume processes listening on TCP/1028 and TCP/1029 ports, then you can add the following lines to the `log4j.properties` file:

```

log4j.appender.cosmosinjector1028=org.apache.log4j.RollingFileAppender
log4j.appender.cosmosinjector1028.MaxFileSize=100MB
log4j.appender.cosmosinjector1028.MaxBackupIndex=10
log4j.appender.cosmosinjector1028.File=${flume.log.dir}/cosmos-
injector.1028.log

log4j.appender.cosmosinjector1028.layout=org.apache.log4j.PatternLayout
log4j.appender.cosmosinjector1028.layout.ConversionPattern=%d{dd MMM
yyyy HH:mm:ss,SSS} %-5p [%t] (%C.%M:%L) %x - %m%n

log4j.appender.cosmosinjector1029=org.apache.log4j.RollingFileAppender
log4j.appender.cosmosinjector1029.MaxFileSize=100MB
log4j.appender.cosmosinjector1029.MaxBackupIndex=10
log4j.appender.cosmosinjector1029.File=${flume.log.dir}/cosmos-
injector.1029.log

```

```
log4j.appender.cosmosinjector1029.layout=org.apache.log4j.PatternLayout
log4j.appender.cosmosinjector1029.layout.ConversionPattern=%d{dd MMM
yyyy HH:mm:ss,SSS} %-5p [%t] (%C.%M:%L) %x - %m%n
```

Once the log4j has been properly configured, you only have to add to the Flume command line the following parameter, which overwrites the default configuration (flume.root.logger=INFO,LOGFILE):

```
-Dflume.root.logger=<loggin_level>,cosmos-injector.<TCP_port>.log
```

6.3.3.4 **Step 4: running**

In foreground (with logging):

```
APACHE_FLUME_HOME/bin/flume-ng agent --conf APACHE_FLUME_HOME/conf -f
APACHE_FLUME_HOME/conf/cosmos-injector.conf -n orionagent -
Dflume.root.logger=INFO,console
```

In background:

```
nohup APACHE_FLUME_HOME/bin/flume-ng agent --conf
APACHE_FLUME_HOME/conf -f APACHE_FLUME_HOME/conf/cosmos-injector.conf
-n orionagent -Dflume.root.logger=INFO,LOGFILE &
```

Remember you can change the logging level and the logging appender by changing the -Dflume.root.logger parameter.

6.4 Sanity check procedures

6.4.1 End to end testing

The e2e tests must be done per logical component of the platform and per public service.

6.4.1.1 **Master Node**

Log into the Master Node web page (https://<master_node>), go to the documentation page (https://<master_node>/doc.html) and try the Swagger-like API. For instance, you can get the information about you user:

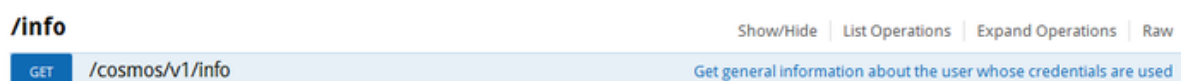


Figure 1 - /info API

A response like the following one should be received:

```
{
  "profileId": 36,
  "handle": "xxxxxx",
  "individualQuota": 6,
  "group": {
    "name": "No Group",
    "guaranteedQuota": 0
  },
  "clusters": {
    "owned": [],
    "accessible": []
  },
  "resources": {
    "groupConsumption": 12,
    "individualConsumption": 0,
    "available": 4,
    "availableForGroup": 4,
    "availableForUser": 4
  }
}
```

You can also try to create and destroy a computing cluster:

/cluster		Show/Hide	List Operations	Expand Operations	Raw
GET	/cosmos/v1/cluster				List clusters
POST	/cosmos/v1/cluster				Create a new cluster
POST	/cosmos/v1/cluster/{id}/add_user				Add users to an existing cluster
GET	/cosmos/v1/cluster/{id}				Get cluster machines
POST	/cosmos/v1/cluster/{id}/remove_user				Remove a user from an existing cluster
POST	/cosmos/v1/cluster/{id}/terminate				Terminate cluster

Figure 2 - /cluster API

A response like the following one should be received when trying to create it:

```
{
  "id": "42821b68953f44dca44bfff6f561a4b9",
  "href":
"https://cosmos.hi.inet/cosmos/v1/cluster/42821b68953f44dca44bfff6f561a4b9",
  "name": "test",
  "state": "provisioning",
  "stateDescription": "Cluster is acquiring and configuring resources",
  "creationDate": "2014-05-19T14:51:41+02:00"
}
```

As can be seen, the cluster starts to provision; you can check its status by its id. Once provisioned, the response should be:

```
{
  "href":
"https://cosmos.hi.inet/cosmos/v1/cluster/42821b68953f44dca44bfff6f561a4b9",
  "id": "42821b68953f44dca44bfff6f561a4b9",
  "name": "test",
  "size": 2,
  "state": "running",
  "stateDescription": "Cluster is ready",
  "services": [
    "HDFS",
    "MAPREDUCE2",
    "YARN",
    "ZOOKEEPER"
  ],
  "master": {
    "hostname": "xxxxx",
    "ipAddress": "xxxxx"
  },
}
```

```

"slaves": [
  {
    "hostname": "xxxxx",
    "ipAddress": "xxxxx"
  },
  {
    "hostname": "xxxxx",
    "ipAddress": "xxxxx"
  }
],
"users": [
  {
    "username": "xxxxx",
    "sshPublicKey": "ssh-rsa
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
    "isSudoer": false
  }
]
}

```

Finally, terminate it; the following response should be received:

```

{
  "message": "Terminating cluster"
}

```

6.4.1.2 *Infinity (storage cluster)*

Infinity speaks WebHDFS, the RESTful API for HDFS management. In order to test the status of the persistent storage of the GE, try listing the user space of a valid HDFS user:

```

$ curl -i
"http://<infinity_namenode>:57000/webhdfs/v1/user/<hdfs_user>?op=LISTS
TATUS&user.name=<hdfs_user>"

```

You can also try to put some data (2-step operation), and then read it. First of all, connect to the Infinity namenode in order to create the new file name:

```
$ curl -i -X PUT
"http://<infinity_namenode>:57000/webhdfs/v1/user/<hdfs_user>/<new_file_name>?op=CREATE&user.name=<hdfs_user>"
```

A redirection to the Infinity datanode that will store the data is received:

```
$ curl -i -X PUT -T <local_data_file> --header "content-type:
application/octet-stream"
"http://<infinity_datanode>:57000/webhdfs/v1/user/<hdfs_user>/<new_file_name>?op=CREATE&user.name=<hdfs_user>&data=true"
```

Now it can be listed or read:

```
$ curl -i
"http://<infinity_namenode>:57000/webhdfs/v1/user/<hdfs_user>/<new_file_name>?op=LISTSTATUS&user.name=<hdfs_user>"

$ curl -i -L
"http://<infinity_namenode>:57000/webhdfs/v1/user/<hdfs_user>/<new_file_name>?op=OPEN&user.name=<hdfs_user>"
```

Finally, delete it:

```
$ curl -i -X DELETE
"http://<infinity_namenode>:57000/webhdfs/v1/user/<hdfs_user>/<new_file_name>?op=DELETE&recursive=true&user.name=<hdfs_user>"
```

6.4.1.3 *Cygnus*

The Flume-based event processor can be tested by sending it a simple notification:

```
$ curl $1 -v -s -S --header 'Content-Type: application/xml' --header
'Accept: application/xml' --header 'User-Agent: orion/0.10.0' -d @-
<<EOF

<notifyContextRequest>

  <subscriptionId>51c0ac9ed714fb3b37d7d5a8</subscriptionId>

  <originator>localhost</originator>

  <contextResponseList>
    <contextElementResponse>
      <contextElement>
        <entityId type="Room" isPattern="false">
          <id>Room1</id>
        </entityId>
        <contextAttributeList>
```

```

        <contextAttribute>
            <name>temperature</name>
            <type>centigrade</type>
            <contextValue>26.5</contextValue>
        </contextAttribute>
    </contextAttributeList>
</contextElement>

<statusCode>
    200
    <reasonPhrase>OK</reasonPhrase>
</statusCode>
</contextElementResponse>
</contextResponseList>
</notifyContextRequest>
EOF

```

If everything goes well, the `curl` command must return something like this:

```

* About to connect() to localhost port 5050 (#0)
*   Trying ::1... connected
* Connected to localhost (::1) port 5050 (#0)
> POST /notify HTTP/1.1
> Host: localhost:5050
> Content-Type: application/xml
> Accept: application/xml
> User-Agent: orion/0.10.0
> Content-Length: 560
>
< HTTP/1.1 200 OK
< Transfer-Encoding: chunked
< Server: Jetty(6.1.26)
<

```



```
* Connection #0 to host localhost left intact
* Closing connection #0
```

In addition, you can check a `cygnus-myuser-mydataset-Room1-Room.txt` file has been created in the `/user/myuser/mydata/` HDFS folder, being the content of such file:

```
{"ts":"1400486021","iso8601date":"2014-05-19T09:53:41.158","entityId":"Room1","entityType":"Room","attrName":"temperature","attrType":"centigrade","attrValue":"26.5"}
```

6.4.2 List of running processes

Following there is a list of processes (both from Cosmos and Cygnus) that must be running if the GE has been correctly deployed.

The `ps` command can be used in order to check if the process is running or not:

```
$ ps -ef | grep <keyword> | grep -v grep | cut -c 1-120
```

You can also check them by looking for the corresponding opened ports:

```
$ netstat -na | grep <port> | grep -v grep
```

6.4.2.1 **Master Node**

- Ambari Server (Java process)
 - 8080: internal use, accessed by the GUI
 - 8440: accessed by the Ambari agents
- Cosmos API (Scala process)
 - 9000: API and web, internal use
- httpd (Apache process)
 - 80: httpd redirects to SSL (443)
 - 443: httpd redirects to Cosmos Admin running at 9000
 - 8081: internal access to the local repository
 - 8000: internal secure access to the local repository
- mysqld
 - 3306: default port
- postgres
 - 5432: default port

6.4.2.2 ***Infinity Namenode***

- HDFS NameNode (Java process)
 - 8020: Namenode inter-process communications
 - 50070 WebHDFS port for external communications
- Ambari Agent (Python process)
 - 8670: ping port

6.4.2.3 ***Infinity Datanode***

- HDFS DataNode (Java process)
 - 8010: Datanode inter-process communications
 - 50010: HDFS port for local communications
 - 50075: Namenode WebHDFS port for external communications
- Zookeeper (Java process)
 - 38767: Zookeeper
 - 2181: Zookeeper
- Ambari Agent (Python process)
 - 8670: ping port

6.4.2.4 ***Slave nodes***

- Zookeeper (Java process)
 - 38767: Zookeeper
 - 2181: Zookeeper
- Hadoop Yarn (Java process)
 - 8141: Yarn resource manager
 - 8050: Yarn resource manager
 - 8088: Yarn resource manager
 - 45454: Yarn node manager
 - 8025: Yarn node manager
 - 13562: Yarn node manager
 - 8030: Yarn node manager

- 8040: Yarn node manager
 - 8042: Yarn node manager
- HDFS NameNode (Java process)
 - 8020: Namenode inter-process communications
 - 50010: HDFS port for local communications
 - 50070: Namenode WebHDFS port for external communications
- HDFS DataNode (Java process)
 - 8010: Datanode inter-process communications
 - 50075: Datanode WebHDFS port for external communications
- Hadoop JobTracker (Java process)
 - 50030: JobTracker
- Hadoop TaskTracker (Java process)
 - 50060: TaskTracker
- Ambari Agent (Python process)
 - 8670: ping port

6.4.2.5 *Cygnus*

- Flume (Java process)

6.4.3 Network interfaces up and open

The master node does not any special network interface apart from the management/service interface (typically, eth0).

Nevertheless, due OpenVZ is installed in the slaves, 3 new interfaces must appear:

```
$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:50:56:B5:00:26
          inet6 addr: fe80::250:56ff:feb5:26/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2161156 errors:0 dropped:0 overruns:0 frame:0
          TX packets:656661 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
```

```

RX bytes:1990403376 (1.8 GiB)  TX bytes:45343724 (43.2 MiB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

venet0  Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
        inet6 addr: fe80::1/128 Scope:Link
        UP BROADCAST POINTOPOINT RUNNING NOARP  MTU:1500  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

veth101.0 Link encap:Ethernet  HWaddr FE:FF:FF:FF:FF:FF
        inet6 addr: fe80::fcff:ffff:feff:ffff/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:22 errors:0 dropped:0 overruns:0 frame:0
        TX packets:27745 errors:0 dropped:2 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:832 (832.0 b)  TX bytes:1779790 (1.6 MiB)

vzbr0   Link encap:Ethernet  HWaddr 00:50:56:B5:00:26
        inet addr:10.95.171.92  Bcast:10.95.171.127
Mask:255.255.255.192

```

```

inet6 addr: fe80::250:56ff:feb5:26/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:2111362 errors:0 dropped:0 overruns:0 frame:0
TX packets:628570 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:1887070544 (1.7 GiB)  TX bytes:43363129 (41.3 MiB)

```

These new interfaces are shown only if the slaves have been restarted once OpenVZ has been installed. If so, the slave starts with the OpenVZ kernel (version 2.6.32-042stab079.6); this can be checked in the slaves by typing this command:

```

$ uname -r
2.6.32-042stab079.6

```

The OpenVZ container may be accessed as well:

```

$ sudo vzctl enter 101
entered into CT 101
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:18:51:CF:FD:6F
          inet addr:10.95.171.41  Bcast:10.95.255.255
Mask:255.255.0.0
          inet6 addr: fe80::218:51ff:fecf:fd6f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:20 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1108 (1.0 KiB)  TX bytes:552 (552.0 b)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

```

```
collisions:0 txqueuelen:0
RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
```

6.4.4 Databases

The platform uses a MySQL database where to store certain metainformation. This can be checked by typing this command on the master node:

```
$ mysql cosmos
```

The mysql prompt will be opened, and the "cosmos" database will be selected. The following tables should be shown:

```
mysql> show tables;
+-----+
| Tables_in_cosmos |
+-----+
| cluster           |
| cluster_master    |
| cluster_slave     |
| cluster_state     |
| cluster_user      |
| play_evolution    |
| public_key        |
| schema_migrations |
| user              |
| user_capability   |
| user_group        |
+-----+
11 rows in set (0.00 sec)

mysql>
```

6.5 Diagnosis procedures

6.5.1 Resource availability

Please, refer to the [Resource Requirements](#) section in this document.

6.5.2 Remote Service Access

Please, refer to the [End to End Testing](#) section in this document.

6.5.3 Resource consumption

The following outputs are got by typing this command in a shell:

```
$ top -p <pid1>,<pid2>,...
```

MySQL:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5626	root	20	0	105m	24	20	S	0.0	0.0	0:00.00	
mysqld_safe											
5784	mysql	20	0	715m	4864	1660	S	0.0	1.0	67:12.42	mysqld

Cosmos API and Ambari Server:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
6738	root	20	0	2134m	133m	1412	S	2.7	29.1	101:49.40	java
(cosmos-api)											
6723	root	20	0	2879m	228m	3436	S	0.7	49.8	136:01.25	java
(ambari-server)											

httpd:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5156	apache	20	0	148m	84	76	S	0.0	0.0	0:00.00	httpd
5157	apache	20	0	148m	1740	632	S	0.0	0.4	0:01.02	httpd
5158	apache	20	0	148m	1060	332	S	0.0	0.2	0:00.02	httpd
5159	apache	20	0	148m	1060	332	S	0.0	0.2	0:00.00	httpd
5160	apache	20	0	148m	264	148	S	0.0	0.1	0:00.10	httpd
5161	apache	20	0	148m	84	76	S	0.0	0.0	0:00.00	httpd
5162	apache	20	0	148m	84	76	S	0.0	0.0	0:00.00	httpd

5163	apache	20	0	148m	84	76 S	0.0	0.0	0:00.00	httpd
27583	root	20	0	148m	164	128 S	0.0	0.0	0:22.66	httpd

Cygnus:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
15032	myuser	20	0	2426m	123m	15m	S	0.0	1.2	73:33.50	java

6.5.4 I/O flows

There will be a flow per each exposed service. Please, refer to the [List of running processes](#) section in this document.

7 Compressed Domain Video Analysis - Installation and Administration Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

7.1 Prerequisites

7.1.1 System

Codoan is available for Windows (at least Windows 7) and Debian-based Linux distributions, e.g., Ubuntu (at least Ubuntu 13.10). The system architecture has to be x86_64-compatible and must support multi-threading techniques. For running Codoan on desktop based systems at least a CPU with 2 cores (4 cores preferred) and physical RAM of at least 2 GB should be available (4 GB preferred).

7.1.2 Network

Codoan provides a RESTful API and therefore requires port 80 to be open for input and output operations. The media is received as stream for which RTSP and RTP/RTCP is used. RTSP requires port 554 to be open and RTP/RTCP uses random ports ≥ 49152 .

7.2 Installation

The RESTful web service and the core library of Codoan are provided within a single executable file (Windows: **codoan.exe**, Linux: **codoan**). Additionally, a daemon (**codoand**) is provided for Debian-based Linux systems that automatically starts Codoan when the system starts or Codoan has been terminated for some reason. Both will be installed, configured and started when executing the provided Linux shell script **install.sh** as follows.

1. Open a terminal as root
2. Create an empty directory (e.g., **codoan**) and unzip the Codoan package into this directory
3. Make all files in this directory accessible by root:

```
chmod -R 700 codoan/
```

4. Change to this directory:

```
cd codoan/
```

5. Run the shell script:

```
./install.sh
```

If Codoan should be uninstalled for some reason, the Linux shell script **uninstall.sh** can be used as follows.

1. Open a terminal as root

2. Change to the previously created directory:

```
cd codoan/
```

3. Run the shell script:

```
./uninstall.sh
```

7.3 Configuration

Each instance within Codoan can be configured separately. A single instance receives and processes a single RTP stream. The available configuration method is *configureInstance* (see [\[1\]](#)). It is accessible via the RESTful API.

The standard procedure to set up and start an instance is as follows: *createInstance* -> *configureInstance* -> *addSink* -> *startInstance*

7.4 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.

7.4.1 End to End Testing

To check that Codoan is up and running, request its version by sending the following HTTP request.

```
GET //server/codoan/version HTTP/1.1
Accept: application/xml
```

Where *server* indicates the actual URI of Codoan.

The response should be similar to the following.

```
HTTP/1.1 200 OK
Content-Length: 184
Content-Type: application/xml
Server: codoan REST server

<?xml version="1.0" encoding="UTF-8"?>
```

```
<Codoan>

  <Version>2.0.0</Version>

  <Copyright>(c) 2010-2014 Imaging and Computer Vision, Siemens
Corporate Technolog</Copyright>

</Codoan>
```

7.4.2 List of Running Processes

The following processes should be running when Codoan has been started.

- Windows
 - **codoan.exe**: Executable of Codoan implementation
- Linux (Debian-based)
 - **codoan**: Executable of Codoan implementation
 - **codoand**: Daemon that automatically starts Codoan on startup

On Debian-based Linux systems, the following command line can be used to verify that the required processes are running.

```
ps -ef | grep -e codoan -e PID
```

The expected output is similar to the following.

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	905	1	0	Dec04	?	00:00:01	/usr/local/bin/codoan/codoand
root	907	905	0	Dec04	?	00:07:50	codoan
root	11620	11539	0	14:01	pts/0	00:00:00	grep --color=auto -e codoan -e PID

7.4.3 Network Interfaces Up & Open

- 80/HTTP: RESTful API provided by Mongoose web server
- 554/RTSP: Establishing sessions for RTP reception
- 49152-65535/RTP-RTCP: Receiving RTP streams (random port pair for each session)

7.4.4 Databases

Codoan does not use any databases.

7.5 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.

7.5.1 Resource Availability

Although 4 GB RAM are recommended, the system should not have less than 2 GB to work properly. That is because of some internal data structures that are stored temporarily during processing. The hard disk size is less critical since Codoan just writes some small log files to disk.

7.5.2 Remote Service Access

In the current release, Codoan has no built-in integration with other GEs.

7.5.3 Resource Consumption

The resource consumption depends on the number and the characteristics of processed streams. No typical numbers can be stated here. However, less than 200 MB of free memory and more than 85% of CPU load over more than 10 seconds may lead to an abnormal behavior of the GE implementation (i.e., *codoan*).

7.5.4 I/O Flows

The main I/O flows are the input RTP video streams and respective RTCP control streams (on random ports between 49152 and 65535). Further I/O flows are for configuration over the RESTful API (HTTP on port 80) and for stream setup (RTSP on port 554).

7.6 References

[1] [Compressed Domain Video Analysis Open RESTful API Specification](#)

8 Unstructured Data Analysis - Installation and Administration Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

8.1 Introduction

This guide covers the steps needed to install and configure the Unstructured Data Analysis GE.

8.2 System Requirements

In order to deploy the Unstructured Data Analysis GE the following software must be previously installed:

- JavaTM Platform, Standard Edition Development Kit (JDKTM) 6 [\[1\]](#).
- Maven 2.1.1 [\[2\]](#)
- Apache Tomcat 6.0.32 [\[3\]](#)
- MySQL Community Server GA 5.1.63 [\[4\]](#)
- Apache Rome Fetcher 1.0 [\[5\]](#)
- Apache Nutch 2.2.1 [\[6\]](#)
- Apache Solr 3.6.0 [\[7\]](#)
- Apache HBase 0.94.6 [\[8\]](#)
- Apache Flume 1.4.0 [\[9\]](#)

Online installation documentation is available for all required software in case it is needed.

8.3 Instalation guidelines

This guide defines the procedure to install the Unstructured Data Analysis. For the sake of simplicity, all the commands and procedures included in this guide are oriented to a Linux server, but, as the Unstructured Data Analysis is a JEE application, it can be easily installed on a windows server.

8.3.1 Configuring the infrastructure

In order to deploy a working instance of the GE a suitable HBase datastore, Solr Index and MySQL schema should be needed. In this section we will learn how to deploy a new datastore in Hbase, how to create a new DB schema using MySQL and how to create a new Data Index using Apache Solr.

8.3.1.1 *HBase Datastore Backend*

Instead of segment files, UDA GE uses HBase as its backend datastore.

Support for multiple backends is achieved using GORA, an ORM framework (originally written for Nutch) that works against Column databases. So changing backends would (probably, haven't looked at the GORA code yet) mean adding the appropriate GORA implementation JAR into Nutch's classpath.

Currently, there is a working HBase backend, and adequate documentation on how to set it up.

So this guideline is basically an attempt to figure out what Nutch does to the HBase datastore as each of its subcommands are run.

The first step is to download Nutch 2.0 and GORA sources, and build them. The only things to remember is to set the GORA backend in `conf/nutch-site.xml` after generating the nutch runtime.

Two other changes are to set the `http.agent.name` and `http.robots.agents` in `nutch-default.xml` (so nutch actually does the crawl), and the `hbase.rootdir` in `hbase-default.xml` to something other than `/tmp` (to prevent data loss across system restarts).

First, we have to start up HBase so Nutch can write to it. Part of the Nutch/GORA integration instructions was to install HBase, so now we can start up a local instance, and then login to the HBase shell.

UDA first time inject URLs from "project sources" into a seed file called `seed.txt`. First time a single table called "webpage" being created in HBase, with the following structure. UDA used `list` to list the tables, and `scan` to list the contents of the table. For ease of understanding, It was reformatted the output manually into a JSON structure. Each leaf level column (cell in HBase-speak) consists of a (key, timestamp, value) triplet, so we could have written the first leaf more compactly as `{f1 : "\x00'\x80\x00"}`.

It might help to refer to the `conf/gora-hbase-mapping.xml` file in your Nutch runtime as you read this. If you haven't set up Nutch 2.0 locally, then this information is also available in the [GORA_HBase wiki page](#).

```
webpage : {
  key : "com.blogspot.sujitpal:http/",
  f : {
    fi : {
      timestamp : 1293676557658,
      value : "\x00'\x8D\x00"
    },
    ts : {
      timestamp : 1293676557658,
      value : "\x00\x00\x01-5!\x9D\xE5"
    }
  }
}
```

```

},
mk : {
  _injmrk_ : {
    timestamp : 1293676557658,
    value : "y"
  }
},
mtdt : {
  _csh_ : {
    timestamp : 1293676557658,
    value : "x80\x00\x00"
  }
},
s : {
  s : {
    timestamp : 1293676557658,
    value : "x80\x00\x00"
  }
}

```

```

}

```

Now, UDA internally run the generate step, which generates the fetchlist:

This creates an additional column "mk:_gnmrk_" containing the batch id, in the webpage table for the record keyed by the seed URL.

```

webpage : {

```

```

  key : "com.blogspot.sujitpal:http/",
  f : {
    fi : {
      timestamp : 1293676557658,
      value : "\x00'\x8D\x00"
    },
    ts : {

```

```

        timestamp : 1293676557658,
        value : "\x00\x00\x01-5!\x9D\xE5"
    }
},
mk : {
    _injmrk_ : {
        timestamp : 1293676557658,
        value : "y"
    },
    _gnmrk_ : {
        timestamp=1293732629430,
        value : "1293732622-2092819984"
    }
},
mtdt : {
    _csh_ : {
        timestamp : 1293676557658,
        value : "x80\x00\x00"
    }
},
s : {
    s : {
        timestamp : 1293676557658,
        value : "x80\x00\x00"
    }
}
}

```

```

}

```

Next UDA run a fetch with the batch id returned by the generate command.

This creates some more columns as shown below. As you can see, it creates additional columns under the "f" column family, most notably the raw page content in the "f:cnt" column and a new "h" column family with page header information. It also creates a batch id marker in the "mk" column family.

webpage : {

```
key : "com.blogspot.sujitpal:http/",
f : {
  bas : {
    timestamp : 1293732801833,
    value : "http://sujitpal.blogspot.com/"
  },
  cnt : {
    timestamp : 1293732801833,
    value : "DOCTYPE html PUBLIC "-//W3C//DTD X...rest of page
content"
  },
  fi : {
    timestamp : 1293676557658,
    value : "\\x00'\\x8D\\x00"
  },
  prot : {
    timestamp : 1293732801833,
    value : "\\x02\\x00\\x00"
  },
  st : {
    timestamp : 1293732801833,
    value : "\\x00\\x00\\x00\\x02"
  },
  ts : {
    timestamp : 1293676557658,
    value : "\\x00\\x00\\x01-5!\\x9D\\xE5"
  }
}
```

```
typ : {
  timestamp : 1293732801833,
  value : "application/xhtml+xml"
},
h : {
  Cache-Control : {
    timestamp : 1293732801833,
    value : "private"
  },
  Content-Type : {
    timestamp : 1293732801833,
    value : "text/html; charset=UTF-8"
  },
  Date : {
    timestamp : 1293732801833,
    value : "Thu, 30 Dec 2010 18:13:21 GMT"
  },
  ETag : {
    timestamp : 1293732801833,
    value : "40bdf8b9-8c0a-477e-9ee4-b19995601dde"
  },
  Expires : {
    timestamp : 1293732801833,
    value : "Thu, 30 Dec 2010 18:13:21 GMT"
  },
  Last-Modified : {
    timestamp : 1293732801833,
    value : "Thu, 30 Dec 2010 15:01:20 GMT"
  },
}
```

```
Server : {
  timestamp : 1293732801833,
  value : "GSE"
},
Set-Cookie : {
  timestamp : 1293732801833,
  value : "blogger_TID=130c0c57a66d0704;HttpOnly"
},
X-Content-Type-Options : {
  timestamp : 1293732801833,
  value : "nosniff"
},
X-XSS-Protection : {
  timestamp : 1293732801833,
  value : "1; mode=block"
}
},
mk : {
  _injmrk_ : {
    timestamp : 1293676557658,
    value : "y"
  },
  _gnmrk_ : {
    timestamp=1293732629430,
    value : "1293732622-2092819984"
  },
  _ftcmrk_ : {
    timestamp : 1293732801833,
    value : "1293732622-2092819984"
  }
}
```

```

},
mtdt : {
  _csh_ : {
    timestamp : 1293676557658,
    value : "x80\x00\x00"
  }
},
s : {
  s : {
    timestamp : 1293676557658,
    value : "x80\x00\x00"
  }
}

```

```

}

```

Finally UDA parse the fetched content. This extracts the links and parses the text content out of the HTML.

This results in more columns written out to the webpage table. At this point it parses out the links from the page and stores them in the "ol" (outlinks) column family, and the "p" column family, which contains the parsed content for the page.

```

webpage : {

```

```

  key : "com.blogspot.sujitpal:http/",
  f : {
    bas : {
      timestamp : 1293732801833,
      value : "http://sujitpal.blogspot.com/"
    },
    cnt : {
      timestamp : 1293732801833,
      value : "DOCTYPE html PUBLIC "-//W3C//DTD X...rest of page content"
    },
  },

```

```
fi : {  
  timestamp : 1293676557658,  
  value : "\x00'\x8D\x00"  
},  
prot : {  
  timestamp : 1293732801833,  
  value : "x02\x00\x00"  
},  
st : {  
  timestamp : 1293732801833,  
  value : "x00\x00\x00\x02"  
},  
ts : {  
  timestamp : 1293676557658,  
  value : "\x00\x00\x01-5!\x9D\xE5"  
},  
typ : {  
  timestamp : 1293732801833,  
  value : "application/xhtml+xml"  
},  
},  
h : {  
  Cache-Control : {  
    timestamp : 1293732801833,  
    value : "private"  
  },  
  Content-Type : {  
    timestamp : 1293732801833,  
    value : "text/html; charset=UTF-8"  
  },  
  Date : {
```

```
    timestamp : 1293732801833,
    value : "Thu, 30 Dec 2010 18:13:21 GMT"
  },
  ETag : {
    timestamp : 1293732801833,
    value : 40bdf8b9-8c0a-477e-9ee4-b19995601dde"
  },
  Expires : {
    timestamp : 1293732801833,
    value : "Thu, 30 Dec 2010 18:13:21 GMT"
  },
  Last-Modified : {
    timestamp : 1293732801833,
    value : "Thu, 30 Dec 2010 15:01:20 GMT"
  },
  Server : {
    timestamp : 1293732801833,
    value : "GSE"
  },
  Set-Cookie : {
    timestamp : 1293732801833,
    value : "blogger_TID=130c0c57a66d0704;HttpOnly"
  },
  X-Content-Type-Options : {
    timestamp : 1293732801833,
    value : "nosniff"
  },
  X-XSS-Protection : {
    timestamp : 1293732801833,
    value : "1; mode=block"
```

```
}  
,  
mk : {  
  _injmrk_ : {  
    timestamp : 1293676557658,  
    value : "y"  
  },  
  _gnmrk_ : {  
    timestamp=1293732629430,  
    value : "1293732622-2092819984"  
  },  
  _ftcmrk_ : {  
    timestamp : 1293732801833,  
    value : "1293732622-2092819984"  
  },  
  __prsmrk__ : {  
    timestamp : 1293732957501,  
    value : "1293732622-2092819984"  
  }  
,  
mtdt : {  
  _csh_ : {  
    timestamp : 1293676557658,  
    value : "x80\x00\x00"  
  }  
,  
s : {  
  s : {  
    timestamp : 1293676557658,  
    value : "x80\x00\x00"
```

```

    }
  }
  ol : {
    http://pagead2.googlesyndication.com/pagead/show\_ads.js : {
      timestamp : 1293732957501,
      value : ""
    },
    http://sujitpal.blogspot.com/ : {
      timestamp : 1293732957501,
      value : "Home"
    },
    http/
column=ol:http://sujitpal.blogspot.com/2005\_03\_01\_archive.html : {
      timestamp : 1293732957501,
      value : "March"
    },
    // ... (more outlinks below) ...
  },
  p : {
    c : {
      timestamp : 1293732957501,
      value : "Salmon Run skip to main ... (rest of parsed content)"
    },
    sig : {
      timestamp : 1293732957501,
      value="cW\xA5\xB7\xDD\xD3\xBF`\x80oYR8\x1F\ x80\x16"
    },
    st : {
      timestamp : 1293732957501,
      value : "\x02\x00\x00"
    }
  }
}

```



```
    },  
    t : {  
      timestamp : 1293732957501,  
      value : "Salmon Run"  
    },  
    s : {  
      timestamp : 1293732629430,  
      value : "?\x80\x00\x00"  
    }  
  }  
}
```

```
}
```

UDA run the `updatedb` command to add the outlinks discovered during the parse to the list of URLs to be fetched.

8.3.1.2 *Creating a MySQL DB schema*

After you've installed MySQL, you need to set mysql root password. To do so:

Enter the next command in a terminal:

```
mysql -u root
```

Now it should open the mysql console. And type the following line:

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('yourpassword');
```

To exit from the mysql console enter `exit`.

Now you should create the database with the root user. To do so:

Open mysql from terminal:

```
mysql -u root -p
```

Enter the password created before.

Enter the following line:

```
CREATE DATABASE fiware-uda;
```

If you enter `SHOW DATABASES;` you should see it in the list. If so, you have a database ready to use.

8.3.1.3 *Setting up Solr*

It is really easy to setup and basically the only thing requiring special attention is the custom schema to be used (see Solr wiki for more Details about available schema configuration options). Unpack the archive and go to the example directory of extracted package.

I edited the example schema (solr/conf/schema.xml) and added the fields required by Nutch in it's stock configuration:

<fields>

```
<field name="url" type="string" indexed="true" stored="true"/>
<field name="content" type="text" indexed="true" stored="true"/>
<field name="segment" type="string" indexed="false" stored="true"/>
<field name="digest" type="string" indexed="false" stored="true"/>
<field name="host" type="string" indexed="true" stored="false"/>
<field name="site" type="string" indexed="true" stored="false"/>
<field name="anchor" type="string" indexed="true" stored="false"
multiValued="true"/>
<field name="title" type="text" indexed="true" stored="true"/>
<field name="tstamp" type="slong" indexed="false" stored="true"/>
<field name="text" type="text" indexed="true" stored="false"
multiValued="true"/>
</fields>
<uniqueKey>url</uniqueKey>
<defaultSearchField>text</defaultSearchField>
<solrQueryParser defaultOperator="AND"/>
<copyField source="anchor" dest="text"/>
<copyField source="title" dest="text"/>
<copyField source="content" dest="text"/>
```

After setting up the schema just start the Solr server with command: `java -jar start.jar`

8.3.1.4 *Setting up Nutch*

Before starting the crawling process you need to first configure Nutch. If you are not familiar with the way nutch operates it is recommended to first follow the tutorial in Nutch web site.

Basically the steps required are (make sure you use correct filenames - replace '_' with '-')

1. Set up conf/regex-urlfilter.txt 2. Set up conf/nutch-site.xml 3. Generate a list of seed urls into folder urls (UDA framework provides this seed file after fetching with ROME) 4. Grab this simple script that will help you along in your crawling task.

8.3.2 Deploying the GE

This section describes the steps needed to deploy a working version of the Unstructured Data Analysis GE. To do so, we would need to download the binaries from the FI-WARE private SVN (or contact GE owner to get a configured binarie), configure the GE and deploy the resulting war file in our JEE container.

8.3.2.1 *Getting the software*

Unstructured Data Analysis source code can be downloaded from the FI-WARE SVN (with the proper user and password) by using the following command:

```
svn checkout --username <YOUR_USER_NAME> https://forge.fi-ware.eu/scmrepos/svn/data/trunk/UnstructuredDataAnalysis .
```

By doing so, a set of Maven projects should be downloaded into your current directory:

- UDA-service
- UDA-manegement
- UDA-Framework

8.3.2.2 *Configuring the software*

Once the Maven projects are downloaded, we would need to modify the configuration files to connect the GE with a proper data source. To do so we would need to modify one file:

/UnstructuredDataAnalysis/src/main/resources/META-INF/persistence.xml: Here we would need to configure the database access for UDA statistics DB

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
  version="1.0">
```

```

    <persistence-unit name="es.atos.research.fware.uda.beans">
        <class>es.atos.research.fware.uda.beans.Project</class>
        <class>es.atos.research.fware.uda.beans.Source</class>
        <properties>
            <property name="hibernate.archive.autodetection"
value="class, hbm"/>
            <property name="hibernate.connection.driver_class"
value="com.mysql.jdbc.Driver"/>
            <!-- <property name="hibernate.connection.password"
value="reverse"/> -->
            <property name="hibernate.connection.password" value=""/>
            <property name="hibernate.connection.url"
value="jdbc:mysql://localhost/fware-uda"/>
            <property name="hibernate.connection.username"
value="root"/>
            <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect"/>
            <property name="hibernate.c3p0.min_size" value="5"/>
            <property name="hibernate.c3p0.max_size" value="20"/>
            <property name="hibernate.c3p0.timeout" value="300"/>
            <property name="hibernate.c3p0.max_statements"
value="50"/>
            <property name="hibernate.c3p0.idle_test_period"
value="3000"/>
            <!-- <property name="hibernate.hbm2ddl.auto"
value="create"/> -->
        </properties>
    </persistence-unit>
</persistence>

```

8.3.2.3 *Deploying the software*

Once built, you would need to deploy the `uda-service.war` to a JEE container. To do so, you should copy the war file to the deployments directory e.g.: `$ cd <directory uda-rest-service>/target $ cp uda-service.war $TOMCAT_HOME/webapps`

8.4 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.

8.4.1 End to End testing

To verify access to the access to the Unstructured Data Manager API, just issue the following GET request to the Location GE Host using a web browser :

```
http://[UDA GE server ip]:8080/uda-  
service/uda/[PROJECT_NAME]/sources/[SOURCE_NAME]
```

As a result a xml document containing the configuration of a source analyzed in a specific project should be returned.

8.4.2 List of Running Processes

- 1 Tomcat instance, plus the UDA Rest, Solr processes started in Tomcat
- 1 HBase service instance
- 1 MySQL RDBMS
- 1 Flume service instance
- 1 Nutch service instance
- 1 Rome Fetcher service instance

8.4.3 Network interfaces Up & Open

- TCP:8080 (Tomcat + UDA Rest Service API)

8.4.4 Databases

Unstructured Data Analysis GE relies in a MySQL database:

fiware-uda

This database store all the data related with project and resources management. In order to test this instance, the next statements must be executed:

```
mysql -u [DB_USER] -h [MSQL_SERVER] -D fiware-uda  
>select * from project
```

8.5 Diagnosis Procedures

This section will provide information to execute effective diagnosis of potential problems related with the GE

8.5.1 Resource availability

The main critical requirements are the data storage capabilities. This GE continuously gathers and stores unstructured data from different sources. For this reason, a good on-server configuration requirement must be composed at least of 12GB of RAM and 500GB of hard disk

8.5.2 Remote Service Access

The component provides a REST API to allow the remote acces. This API is described in the User Guide.

8.5.3 Resource consumption

The resource consumption depends heavily on the amount of sources to be monitored. This GE required relevant storage capabilities for store the raw and processed data gathered. These data will increase in a significant rate across the time.

8.5.4 I/O flows

As was mentioned before, the GE provide a REST API which is only used by external user (i.e. Use Cases)

9 Metadata Preprocessing - Installation and Administration Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

9.1 Prerequisites

9.1.1 System

9.1.1.1 **Windows**

The MetadataProcessor requires Windows 7. The system architecture has to be 64-bit compatible and must support multi-threading techniques. For running the MetadataProcessor on desktop based systems, at least a CPU with 4 cores and physical RAM of at least 2 GB should be available (4 GB preferred). The DotNET libraries must be installed in Version 4.0 (complete).

9.1.1.2 **Linux**

The MetadataPreprocessor requires Ubuntu Linux 13.04 (64 bit) or later. A CPU with 4 cores and at least 2 GB of physical RAM are recommended.

9.1.2 Network

The MetadataProcessor provides a RESTful API and therefore requires port 8080 to be open for input and output operations. The metadata is received as a stream, for which RTSP and RTP/RTCP is used. RTSP requires port 554 to be open and RTP/RTCP uses random ports ≥ 1024 . While the port for the REST API can be changed via the configuration file, it is not recommended to do so.

9.2 Installation

The Metadata Processing GE is implemented in C# and therefore requires .net, or mono on Ubuntu, respectively.

9.2.1 Windows

The RESTful web server and the core library of the MetadataProcessor is provided as a single executable file (mdpp.exe), a configuration file, and several DLL files, which should all be kept in the same folder as the EXE file. In this case, no explicit installation is required.

Setting up Windows Security: Please do open a command window in superuser mode. Execute the following command:

```
C:\WINDOWS\system32>netsh http add urlacl url=http://+:8080/  
user=<domain\user>
```

Please replace <domain\user> with your domain and username.

9.2.2 Linux

Please create an empty directory and unzip the MetadataPreprocessor package in that directory. Run the following commands in that directory:

```
chmod ugo+x install.sh mdppd.sh mdppd.py  
sudo ./install.sh
```

This will install and start the service “mdppd” in linux.

9.3 Configuration

Before starting mdpp.exe, please edit the config file “mdpp.conf” to reflect your settings. Please make sure to use the same HTTP port as used for setting up Windows Security.

Each instance within the MetadataProcessor can be configured separately. A single instance receives and processes a single RTP stream. The available configuration method is *configureInstance* (see [\[1\]](#)). It is accessible via the RESTful API.

The standard procedure to set up an instance is as follows: *createInstance* -> *configureInstance* -> *addSink* -> *startInstance*. Please be aware that a valid source URI must be set via *configureInstance* before using *startInstance*.

9.4 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.

9.4.1 End to End Testing

To check that the MetadataProcessor is up and running, its version should be requested by sending the following HTTP request.

```
GET //server/mdp/version HTTP/1.1  
Accept: application/xml
```

server indicates the actual URI of the MetadataProcessor (*mdp*). The URI can be found in the configuration file.

9.4.2 List of Running Processes

All modules of the MetadataProcessor (e.g., transformation module, RTP stack, web server, ...) are included in the executable file. Therefore, the MetadataProcessor will run in a single process (mdpp.exe/mdp), however, consisting of multiple threads.

9.4.3 Network Interfaces Up & Open

- 8080/HTTP: RESTful API provided by a .NET web server
- 554/RTSP: Establishing sessions for RTP reception
- 1024-65535/RTP-RTCP: Receiving RTP streams (random port pair for each session, i.e., inbound RTP stream)
- 1024-65535/RTP-RTCP: Sending RTP streams (random port pair for each session, i.e., outbound RTP stream)

9.4.4 Databases

In its current version, the MetadataProcessor does not use any databases.

9.5 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.

9.5.1 Resource Availability

Although 4 GB RAM are recommended, the system should not have less than 2 GB to work properly. That is because of some internal data structures that are stored temporarily during processing. The hard disk size is less critical since the MetadataProcessor just writes a rolling log file with configurable size to the disk. The log file should be used to trace errors in case of unexpected behavior of the MetadataProcessor.

9.5.2 Remote Service Access

The MetadataProcessor currently has no built-in integration with other GEs.

9.5.3 Resource Consumption

The resource consumption depends on the number and the characteristics of processed streams and the complexity of the transformations. No typical numbers can be stated here. However, less than 200 MB

of free memory and more than 85% of CPU load over more than 10 seconds may lead to an abnormal behavior of the GE implementation (i.e., the MetadataProcessor).

9.5.4 I/O Flows

The main I/O flows are the input and output RTP metadata streams and respective RTCP control streams (on random ports between 1024 and 65535 as required by the RTP/RTCP specification). Further I/O flows are for configuration over the RESTful API (HTTP on port 80) and for stream setup (RTSP on port 554).

9.6 References

[1] [Metadata Preprocessing GE RESTful API Specification](#)

10 LOCS - Installation and Administration Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

10.1 Introduction

This guide covers the steps needed to install and configure the Location GE server platform.

LOCS implements the FI-WARE GE Open Specifications associated to the Location GE available at: [FIWARE.ArchitectureDescription.Data.Location](#)

Whenever the term "Location GE" is used, you may assume that we are indeed referring to LOCS that implements the Location GE Open Specifications or an instance of LOCS.

The Location GE is provided as tar gzip archives. First of all, Operating System and external COTS installation prerequisites is described. Then, installation/initial configuration of Location GE software is described.

Protocol and localization functional configuration is defined in editable text properties files. Location system configuration data (network cell definition, privacy policy and end user access control, etc..) is maintained internally in a MySQL database).

10.2 Installation and Preparation

This chapter describes the host server COTS prerequisites before installing the Location GE Software. In general, standard installation of these COTS is sufficient. Specific settings is indicated here for each COTS installation if needed.

10.2.1 Resource requirements

In order to operate effectively the Location GE platform requires the following minimum recommended resource specifications :

Resource	Requirement
CPU	2 cores
Physical RAM	2GB
Disk Space	80GB

10.2.2 Operating System and COTS Installation

10.2.2.1 *Operating System*

Linux operating system is required and must be installed before. Location GE can run using Ubuntu / RedHat. Ubuntu 10.04 or higher is recommended. RedHat 5.3 or higher is recommended.

The **chkconfig** package is needed for installing locationGE as a service at server reboot. If not present on Ubuntu this package can be obtained with the command *apt-get install chkconfig*. This package is only need for starting Location Server as a service. Depending on target OS system version, alternative system settings are necessary.

10.2.2.2 *Java JRE*

Standard Java JRE 6.0 is required. At least version 1.6.0_31 is recommended. On Ubuntu, Java JRE can be obtained running *apt-get openjdk-6-jre*.

10.2.2.3 *MySQL Installation*

MySQL Community edition is required. Both client and server rpms must be installed. At least version 5.1.49 is recommended. On Ubuntu, MySQL can be obtained using *apt-get install mysql-server*.

10.2.2.4 *MySQL Preparation and Specific Settings*

The Location GE software uses the mysql account **locs**, created at installation. The mysql root account is necessary at installation only (see herebelow).

10.2.3 Location GE Software Installation

/etc/hosts file shall be completed if necessary with IP address of target server.

It is recommended to create a specific user account for administrating the location GE (for example locationGE), under which installation/start/stop procedures are run.

The top installation directory is not enforced. The proposed one is */opt/fiware/locationGE*. User (admin) account running the Location Server GE must have read/write access to this directory contents (including sub-directories). Get the Location GE distribution archive (**Standalone-Fiware-LOCS-
<version>.tar.gz**) and unzip the archive in the chosen installation directory.

10.2.3.1 *Environment settings*

Before running any scripts, it is mandatory to define the **\$LOCS_HOME** system variable with value equal to the top directory of the installation distribution layout. For example define in the **.bashrc** of the Location GE admin account.

```
export LOCS_HOME=/opt/fiware/locationGE
```

10.2.3.2 *Location GE service installation*

Optionally, it is possible to install the location GE as as standard init.d service. This option allows automatic start-up of service on location GE server reboot.

Nota :

locsd service script shall be copied on /etc/init.d/. InstallLocsServices.sh script is based on checkconfig usage and could be adapted for target system OS (using for example update-rc.d command). The value of \$LOCS_HOME variable shall be edited in the script **locsd** to point to choosen target installation location is not default installation location is used.

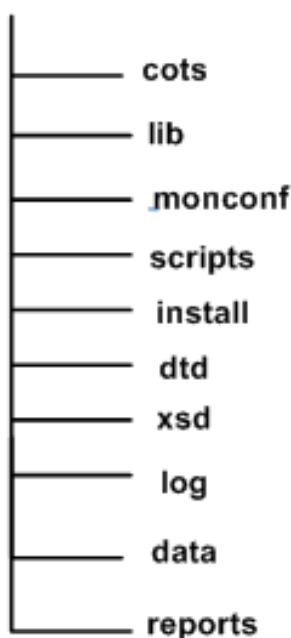
To perform such configuration, run the following command as **root** user :

```
cd $LOCS_HOME/install  
  
chmod +x *.sh  
  
./InstallLocsService.sh
```

10.2.3.3 *Deployment description*

One unzipped, the Location GE software directory organization is the following :

\$LOCS_HOME



Overview of directories content is given here below :

- **cots** : contains the java third party librairies used by the Location GE software
- **lib** : contains the Location GE java software libraries
- **monconf** : contains the Location GE standalone software configuration
- **scripts** : contains the Location GE standalone management scripts (start, shutdown)

- **install** : contains the Location GE standalone post-installation materials (database set-up)
- **dtd** : contains the XML dtd schemas used in Location GE agent APIs.
- **xsd** : contains the XML xsd schemas used in Location GE agent APIs.
- **log** : contains log file outputted at execution time
- **data** : contains Location GE test data
- **reports** : contains output reports generated by tests execution

10.2.3.4 *Post-installation initialization procedure*

Database Initialization :

Database schema contents and minimum insertion of test data shall be performed using the following command (mysql root account password passed at command line) :

```
cd $LOCS_HOME/install  
chmod +x *.sh  
./Install.sh <mysql root account password>
```

At any time, this command can be used to reset the test database contents.

10.2.4 End User Handset Simulator Installation

An handset SUPL V2 simulator needs to be also installed in order to exercise SUPL scenarios workflows between Location GE Server and a end-user handset.

The top installation directory is not enforced. The proposed one is */opt/fiware/handsetSimulator*. User (admin) account running the Location Server GE must have read/write access to this directory contents (including sub-directories). Get the simulator distribution archive (**TestTool-Fiware-<version>.tar.gz**) and unzip the archive in the chosen installation directory.

To start the handset simulator, use the following command

```
cd /opt/fiware/handsetSimulator  
./TestTool.sh
```

To stop the handset simulator, just interrupt the script with CTRL-C.

To verify the current status of the Handset Simulator, the following GET request can be issued to the Location GE instance :

```
GET http://<server root>:8111/testtool/scenario/status HTTP/1.1  
Host: example.com
```

Response :

The response indicates the current simulation status and current running status.

```
HTTP/1.1 200 OK
Content-Length: nnnn
Date: Thu, 02 Jun 2011 02:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<scenarioStatus>
  <selectedScenario>LocationGE-LocationQuery</selectedScenario>
  <running>true</running>
</scenarioStatus>
```

10.2.5 SUPL V2 Enabler Android Application Installation

An Android application is provided to simulate SUPL V2 protocol session to the Location GE server as no compatible mobile handsets exist yet.

Minimum Android System requirements is 4.O.x (Ice Scream Sandwich). The Android application **SuplV2Enabler.apk** is not available on the Play Store Market. It shall be installed from SD card for example (installation from SD Card shall be authorized from Development system Android parameters).

- Click on the SuplV2Enabler.apk installation program to install it.
- Answer to YES to requested access rights.

A Settings Screen is available through standard application Menu. There is no need to change the default settings in scope of Fiware Testing, as only GPS mobile terminal are targeted in scope of Fiware.

By default, mobile geographic positions follow-up is saved in a KML export file on SD card on a per location session basis. It can be retrieved for analysis.

10.3 Starting Location GE

Manual start procedure

To be used if locationGE is not installed as as service.

Open a terminal on Location GE host with Location GE admin account, and run the **StartUp.sh** script in the `$LOCS_HOME/scripts` directory (for background execution run **StartUp.sh > \$LOCS_HOME/log/StartUp.log &**).

Start of locationGE service

If locationGE is optionally installed as a service, the following procedure is valid also :

Open a terminal on Location GE host with Location GE admin account, and run the **service locsd start** command.

Location GE run-time application is composed of 6 java agent processes. Sub-version of each software components is outputted at start-up, then each started agent is listed.

For NET-Initiated scenario involving SMS sending from SUPL Agent to simulated handset, a SMS gateway simulator is also needed.

1. To manually start provided SMS gateway simulator, run the **SmscSimulator.sh** script in the \$LOCS_HOME/scripts directory.
2. to include automatic start of SMS gateway simulator in locationGE start-up, set the following deployment settings (default settings) :

In **LightAgent_app.properties** configuration file in \$LOCS_HOME/monconf directory, set the property **LightAgent.startSmscSimulator** to true.

10.4 Stopping Location GE

Manual stop procedure

There is two ways of stopping manually the LOCS :

- Type **CTRL-C** (interrupt signal) in the terminal in which the deployment manager is started (in interactive session)
- Run the script Shutdown.sh in the \$LOCS_HOME/scripts directory (to be used in case of reconnection to server)

In interactive session, the shutdown procedure lists the agent/monitor currently stopping.

Stopping locationGE service

If locationGE is optionally installed as a service, the following procedure is valid also :

Open a terminal on Location GE host with Location GE admin account, and run the **service locsd stop** command.

10.5 Logging

Each Location GE agent/monitor produces an individual log file in the \$LOCS_HOME/log directory.

- DBMonitor.log : manage DB access and global agent/monitor monitoring.
- MLPAgent.log : contains front-end processing of Terminal Location REST request.
- LrxMonitor.log : contains log of agent provisioning locationGE with live GPS data
- SUPLAgent.log : contains log of SUPL exchanges with simulated end-user terminal
- SmscMonitor.log : contains log of SMS gateway interface monitor
- TestSmsc.log : contains log of simulated network SMS gateway

10.6 REST Interface configuration

Terminal Location API REST Interface can be configured using the

`$LOCS_HOME/monconf/MLPAgent_app.properties`.

HTTP connection port can be configured using the property `http_rest.port`. Default value at installation is 3128.

10.7 MLP Interface configuration

MLP Interface can be configured using the `$LOCS_HOME/monconf/MLPAgent_app.properties`.

HTTP connection port can be configured using the property `http.port`. Default value at installation is 3129.

10.8 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.

10.8.1 End to End testing

To verify access to the Terminal Location API, just issue the following GET request to the Location GE Host using a web browser :

<http://LOCATION GE SERVER IP:3128/location/v1/queries/location>

The location GE shall answer in the browser with a `serviceException` indicating a SYNTAX ERROR.

10.8.2 List of Running Processes

- 7 java processes (one per agent + deployment launcher (see details herebelow))
- `mysqld`
- optionally if launched, one java process for SMS simulator and handset simulator.

Location Platform Java agent processes refers to main executed class **`com.lbs.locs.testsagents.xxx`**.

Deployment Launcher agent process refers to main executed class

`com.lbs.locs.deployment.DefaultDeploymentManager`.

SMS gateway simulator refers to main executed class **`com.lbs.locs.simu.smsc.TestSmsc`**.

Handset Simulator refers to main executed class **`com.locs.testtool.TestTool`**.

10.8.3 Network interfaces Up & Open

- HTTP:3128 (depending on Terminal Location API port defined)

- HTTP:3129 (depending on MLP API port defined)
- TCP:3306 (MySQL)

10.8.4 Databases

MySQL database monitoring status can be obtained running the \$LOCS_HOME/scripts/HostMonitoring.sh script. if Location GE Database status is good, the script returns "OK", else it returns "KO: Invalid MySQL Status".

10.9 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.

Starting point for diagnosis procedure shall check :

- contents of log files for "ERROR" messages (See [Logging](#))
- running processes (See [List of Running Processes](#))

10.9.1 Resource availability

Each Location Platform agent outputs log file in \$LOCS_HOME/log directory. Although all these logs files are designed to be rolling circular files, 1GB of free disk space is needed for that log area.

See also [resource requirements](#) for information regarding the required minimum configuration for this GE.

10.9.2 Remote Service Access

N/A

10.9.3 Resource consumption

The resources used by the platform depend on the amount of location request received per second par the Location GE server. In normal condition for the Fiware Location Platform typical deployment, CPU usage shall not exceed 85%, for 20 location requests per second.

10.9.4 I/O flows

Except for internal health monitoring, there is no "background" activity in the Location Platform.

I/O flows occurs only on location requests through HTTP network interfaces listed in [Network interfaces Up & Open](#).

11 Query Broker - Installation and Administration Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

11.1 Introduction

This guide covers the steps needed to install and configure the Media-enhanced QueryBroker GE.

11.2 System Requirements

This section covers the base level requirements needed to install and use the Media-enhanced QueryBroker GE.

11.2.1 Prerequisites

The Media-enhanced QueryBroker is a J2SE (Java 2 Platform, Standard Edition) application. The REST interface is realized as a wrapper and provided as WAR file (or Web application ARchive) together with the QueryBroker core. The Media-enhanced QueryBroker has been built and tested against a Windows 7 SP1 distribution.

It is recommended to deploy it on a Apache Tomcat 7. For this you need:

1. [Java JRE 7](#) installed
2. [Apache Tomcat 7](#) installed (currently the only tested web server)

Please download and install the recommended software according to the accompanied installation guidelines.

11.2.2 Deploying the Media-enhanced QueryBroker

The Media-enhanced QueryBroker can be deployed in Tomcat by one of the following approaches:

- Copy the WAR file into directory `$CATALINA_BASE/webapps/`. When Tomcat is started, it will automatically expand the web application archive file into its unpacked form, and execute the application that way. NOTE - If you use this approach, and wish to update your application later, you must both replace the web application archive file AND delete the expanded directory that Tomcat created, and then restart Tomcat, in order to reflect your changes.
- Copy the unpacked directory hierarchy into a subdirectory in directory `$CATALINA_BASE/webapps/`. Tomcat will assign a context path to the Media-enhanced QueryBroker based on the subdirectory name you choose. Be sure to restart Tomcat after installing or updating your application.
- Use the Tomcat "Manager" web application to deploy and undeploy web applications. Tomcat includes a web application, deployed by default on context path `/manager`, that allows you to deploy and undeploy applications on a running Tomcat server without restarting it. See the

[administrator documentation](#) for more information on using the Manager web application.

Remark: The default tomcat7 install for the module "manager" contains a maximum file size in 50MB or 52428800 Byte. As the QueryBrokerServer war-file exceeds this size you have to increase *max-file-size* and *max-request-size* in \webapps\manager\WEB-INF\web.xml (on Windows) or in /usr/share/tomcat7-admin/manager/WEB-INF/web.xml (on Ubuntu).

11.2.3 Resource requirements

In order to operate effectively, the Media-enhanced QueryBroker requires an adequately physical RAM, as the queries are processed in-memory. For running the GE on desktop based systems at least a CPU with 2-4 cores and physical RAM of about 2-4 GB should be available.

11.3 Configuration

In order to be able to register and access data repositories according "data base connectors", namely MPQF interpreters need to be implemented. The following section describes in general the actions which are necessary to connect a data store to the media-enhanced Query Broker. Afterwards a quite general adapter for accessing SQL-based relational data stores is depicted.

11.3.1 Implementation of a data repository adapter (in general)

In order to integrate a new adapter (connector), usually the following steps have to be conducted :

1. Construct a MPQF service description:

In order to register a new retrieval service at the QueryBroker, a valid MPQF service description must be produced. In the following two example service description XML files are depicted, describing a [LIRE \(Lucene Image REtrieval\)](#) and the [flickr](#) retrieval service.

```
<?xml version="1.0" encoding="UTF-8"?>

<mpqf:MpegQuery mpqfID="001" xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpqf:schema:2008
mpqf_semantic_enhancement.xsd">

    <mpqf:Management>

        <mpqf:Input>

            <mpqf:DesiredCapability>

                <mpqf:SupportedExampleMediaTypes>image/jpeg</mpqf:SupportedExampleMediaTypes>

                <mpqf:SupportedResultMediaTypes>image/jpeg</mpqf:SupportedResultMediaTypes>
```

```

                                <mpqf:SupportedQueryTypes
href="urn:mpeg:mpqf:2008:CS:full:100.3.6.1" />
                                </mpqf:DesiredCapability>

                                <mpqf:ServiceID>de.uop.dimis.air.adapters.LireAdapter</mpqf:Ser
viceID>

                                </mpqf:Input>

                                </mpqf:Management>
</mpqf:MpegQuery>
<?xml version="1.0" encoding="UTF-8"?>
<mpqf:MpegQuery mpqfID="001" xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpqf:schema:2008
mpqf_semantic_enhancement.xsd">
    <mpqf:Management>
        <mpqf:Input>
            <mpqf:DesiredCapability>

                <mpqf:SupportedResultMediaTypes>image/jpeg</mpqf:SupportedResul
tMediaTypes>

                <mpqf:SupportedQueryTypes
href="urn:mpeg:mpqf:2008:CS:full:100.3.6.2" />
                </mpqf:DesiredCapability>

                <mpqf:ServiceID>de.uop.dimis.air.adapters.FlickrAdapter</mpqf:S
erviceID>

                </mpqf:Input>

                </mpqf:Management>
</mpqf:MpegQuery>

```

A complete description of the used elements can be found in the MPQF standardization document (ISO_IEC_FDIS_15938-12:2008) along with the needed classification scheme. The *ServiceID* points to the Java class of the interpreter implementation (see next point). The actual registering is carried out by submitting the above capability description to the Query Broker. The REST interface provides two ways to do [that](#).

2. Implement the Interface 'Service':

Besides the capability description of a service also a connector to it must be realized. This is done by implementing the interface "Service" of the package "de.uop.dimis.air.backend" accordingly. The system always sends a valid MPQF query to the retrieval service. If the original query consists of more than one query type, this initial query will be split into several sub-queries (one for each query type). This ensures, that every retrieval service only gets the query type for processing, which is defined in the corresponding service description file. Therefore, a retrieval service must translate the information encapsulated in the incoming query into the underlying language/API calls. At the end of processing, the retrieved data must be encapsulated into a valid MPQF response.

In the following an example source code of a QueryByDescription-Service for the service "ExampleService" is given:

```
// ExampleService.java

public class ExampleService implements Service {

    @Override

    public MpegType execute(MpegQueryType distributedQuery) {

        // get the query conditions

        BooleanExpressionType conditions =

distributedQuery.getQuery().getInput().getQueryCondition();

        // ... do your program logic with the query conditions ...

        // create a result container for the computed results
        ObjectFactory mpqfObjFac = new ObjectFactory();
        MpegQueryType result = mpqfObjFac.createMpegQueryType();
        Query qry = mpqfObjFac.createMpegQueryTypeQuery();
        result.setQuery(qry);
        OutputQueryType oqt = mpqfObjFac.createOutputQueryType();
        qry.setOutput(oqt);
        List<ResultItemType> resultItems = oqt.getResultItem();

        // for each result of the service endpoint create a result
        item and add it
    }
}
```

```

        // to the results list

        de.uop.dimis.air.internalObjects.jpsearch.ObjectFactory
jpsearchObjFac =

            new
de.uop.dimis.air.internalObjects.jpsearch.ObjectFactory();

        for(...) {

            ResultItemType resultItem =
mpqfObjFac.createResultItemType();

            resultItmes.add(resultItem);

            Description description =
mpqfObjFac.createResultItemTypeDescription();

            resultItem.getDescription().add(description);

            JPSearchCoreType coreType =
jpsearchObjFac.createJPSearchCoreType();

            description.getContent().add(coreType);

            // set the result properties in the jpsearch coreType
object. (e.g.
            // identifier)
            coreType.setIdentifier("...");

            // set the origin to identify from which service endpoint
the
            // result item was generated
            resultItem.setOriginID("MedicoExecuteDICOM");

        }

        return result;
    }
}

```

Important:

A *serviceID* is the full qualified class name (e.g., de.uop.dimis.test.Service) of the implemented service. The generated class must be located in the classpath of the QueryBroker (by adding a new according class to the Eclipse project QueryBroker-Adapters, which will be provided on request).

11.3.2 Implementation of an adapter for a SQL data base

As most data repositories are today of relational type a quite generic adapter for such data stores has been implemented, which will be available with the end of release 2. Due to the many different variants of SQL capable DB implementations individual adaptations are required. Currently versions for [Postgres](#), [MySQL](#), [MSSQL](#) and [SQLite](#) are included.

The SQL adapter for the Media-enhanced Query Broker aims to provide a solution which translates between the XML-oriented MPEG Query Format and the SQL-based relational database world. It enables the query capability and result aggregation of a SQL database in a distributed environment, controlled by the Query Broker and accessed via standard-compliant MPQF syntax.

11.3.2.1 *Architecture*

The SQL-adapter can operate in one of two general modes:

- normal mode, where a specific MPQF input structure is expected and automatically converted to an SQL query; and
- direct mode, where one can supply the SQL query directly, which bypasses the generator and is executed in its original form.

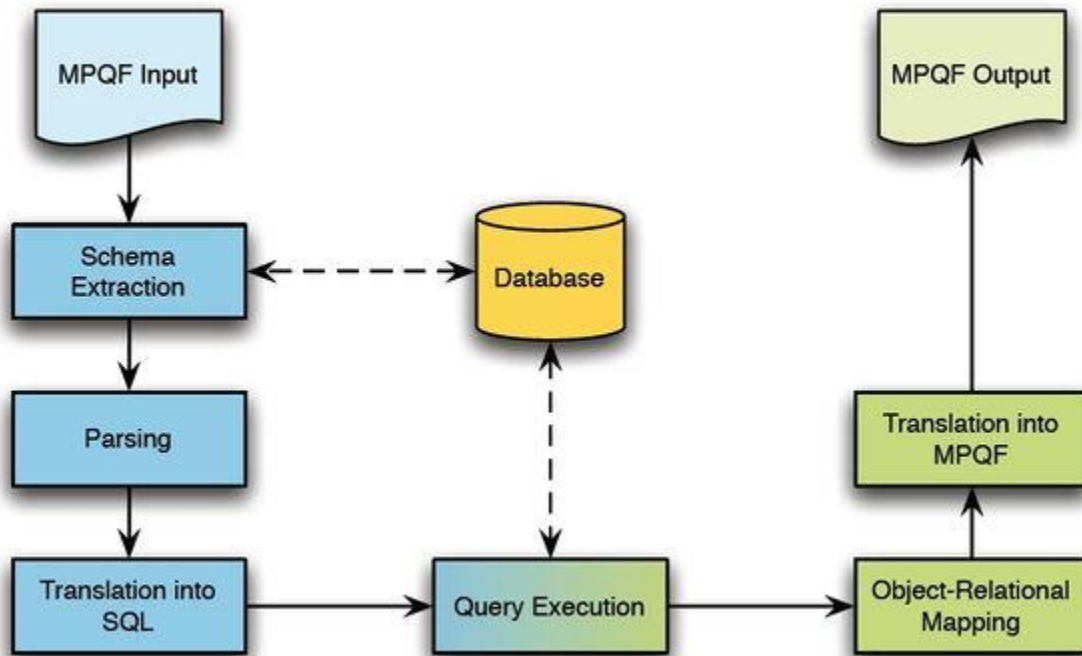
The normal mode is the preferred way of using the adapter, but does not yet support all SQL language features. If some unsupported feature is to be used, one can leverage the direct mode, where (almost) every query string supported by JDBC and the database can be used.

Although a relational structure can be mapped unambiguously to an XML structure the following conventions are made for the sake of convenience:

- The primary keys (PK) of the tables need to be a single column (no composite key!)
- The primary key of a table must not coincide with a foreign key of the same table (PK(A) != FK(A))
- The SELECT/FROM-part shell only contain column names or an asterisk (*) followed by a table name ("AS" or other constructs are not supported)

11.3.2.1.1 *Normal mode*

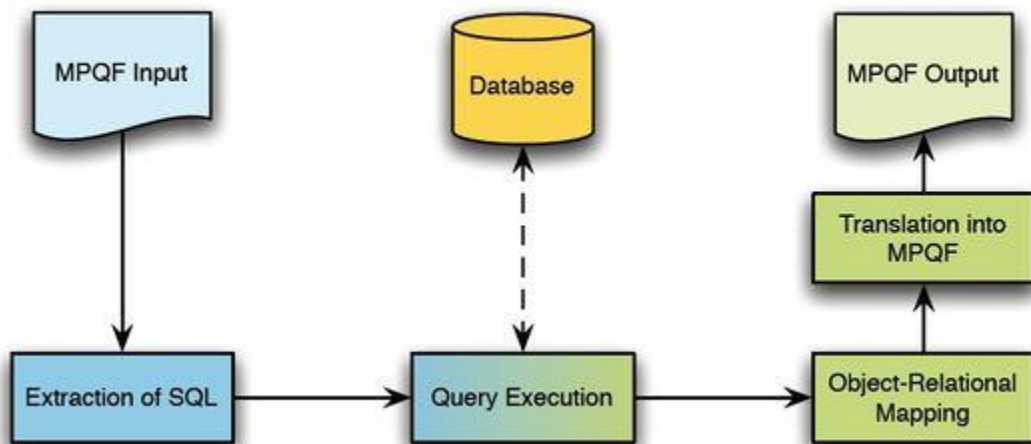
The following figure shows the conceptual flow of the application in normal mode. Parts dealing with the input are marked blue, parts handling the results are marked green and the database and its schema are marked yellow. An MPQF input query from the QueryBroker is processed by extracting the schema for the tables from the database and saving it in an internal format. Afterwards, a parser is used to extract the relevant information from the XML structure of the query and both parts are used to create an equivalent and correct SQL query. This query is then sent to the database via JDBC and the results (in relational form) are – again with the help of the schema extracted earlier – mapped back into XML, which is returned to the QueryBroker.



Architecture of the SQL adapter in normal mode

11.3.2.1.2 *Direct mode*

The following figure shows the conceptual flow of the application in direct mode. The steps for conversion of the input from MPQF to SQL are absent as they are not needed. Instead, the SQL query is passed directly as a QueryByFreeText element to the adapter, is extracted and sent to the database. The result mapping steps are the same as in normal mode.



Architecture of the SQL adapter in direct mode

11.3.2.2 *Input format*

The input format for all queries must be valid MPQF according to the standard specification. The query structure differs between both modes.

11.3.2.2.1 *Normal mode*

The desired query itself is formulated in two locations – the columns that should be projected and the tables from which those column originate are defined in the *<OutputDescription>* part, while the selections which limit the range of results are to be found in the *<QueryCondition>* part.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MpegQuery>
  <Query>
    <Input>
      <OutputDescription>
        <ReqField>/Table1</ReqField>
        <ReqField>/Table1/Table2/Column1</ReqField>
        <ReqField>/Table1/Table2/Column2</ReqField>
      </OutputDescription>
      <QueryCondition>
        <Condition type="OR">
          <Condition type="Equal">
            <ArithmeticField>/Table1/Column1</ArithmeticField>
            <LongValue>100</LongValue>
          </Condition>
          <Condition type="Equal">
            <StringField>/Table2/Column3</StringField>
            <StringValue>some value</StringValue>
          </Condition>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

```

    </Input>
  </Query>
</MpegQuery>

```

This query would translate to the following SQL query, assuming they are related through their primary and foreign key columns:

```

SELECT t1.*, t2.Column1, t2.Column2
FROM Table1 AS t1, Table2 AS t2
WHERE t1.PrimaryKey = t2.ForeignKey
      AND (t1.Column1 = 100 OR t2.Column1 = 'some value');

```

The MPQF elements inside the *<OutputDescription>*, namely the *<ReqField>* elements, are first parsed, then checked against the database schema extracted earlier, and finally assigned to the *SELECT* statement. In case of joins the foreign key and primary key are added automatically if the query contains such selections across multiple tables.

The allowed format is as follows:

```

/Table
  select whole single table
/Table/Column
  select specific column in single table
/Table1/Table2/
  select second table while joining both tables
/Table1/Table2/Column
  select specific column in second table while joining both tables

```

Please regard the following limitations when running joins in the normal mode:

- only the columns of the second table can be listed in the *<ReqField>* elements (cf.

/Table1/Table2/ColumnOfTable2)

- the foreign key (FK) of *"Table2"* references the primary key (PK) of *"Table1"*

Aggregations and sorting are not yet implemented and are therefore ignored.

The query condition part is traversed in a similar way, although it is usually much bigger and ordered like a tree. Each *<Condition>* element is visited recursively until a *<ComparisonExpression>* element is reached, which is the leaf of this subtree and holds the actual condition (arithmetic comparisons or string comparisons). The results are collected, preserving the original order of the tree.

Currently supported elements are:

- OR
- AND
- Equal (=)
- NotEqual (/=)
- LessThan (<)
- LessThanEqual (<=)
- GreaterThan (>)
- GreaterThanEqual (>=)

Currently supported types are numeric values (<ArithmeticField> followed by <LongValue>) and string values (<StringField> followed by <StringValue>).

11.3.2.2.2 Direct mode

When using direct input mode, the complete SQL query has to be wrapped inside a FreeText object.

When running joins, the primary key of the first table listed in the "FROM"-clause is used for identifying the result tuples and therefore must be unique for these. The reason is, that the employed library [DbUtils](#) for submitting the SQL query requires a unique attribute as it uses a hashmap when returning the result tuples.

The query from the previous example would be formulated like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QueryCondition>
        <Condition xsi:type="QueryByFreeText">
          <FreeText>
            SELECT t1.*, t2.Column1, t2.Column2
            FROM Table2 AS t2, Table1 AS t1,
            WHERE t1.PrimaryKey = t2.ForeignKey
            AND (t1.Column1 = 100 OR t2.Column1 = 'some value');
          </FreeText>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

```

    </QueryCondition>
  </Input>
</Query>
</MpegQuery>

```

11.3.2.3 Query execution and result aggregation

After the query string has been created or has been extracted from the query (depending on operation mode), it is sent to the database. The [Apache Commons DB framework](#) in connection with the [C3P0 connection pool](#) take care of the creation and execution of the query via [JDBC](#). The actual query results are then stored as a map containing another map. The outer map has a column with unique values (the primary key of one of the selected tables) as key and the inner map as value; while the inner map has the column names of the result set as keys and the actual cell contents as values.

Type conversion is done automatically from SQL to Java types, utilizing the information gained from the JDBC driver. After an internal conversion into JAXB elements the result tuples are output in according XML format.

11.3.2.4 Output format

The schema of those JAXB objects consists of a result relation, where a *Row* represents a single row in the result set from the database and a *Column* represents a single column. To identify the original tables and columns where the results come from (in case of joins and selections from multiple tables), the *tableName* and *columnName* attributes are used. All result items are stored in the MPQF Output part of the result object, which in turn will be aggregated with the others in the QueryBroker. Each result item holds exactly one row of the result set from the database. The following snippet shows an example of a single result item.

```

:
<Output>
  <ResultItem recordNumber="1" originID="SQL" confidence="1.0">
    <Description fromREF="SQL">
      <Row id="1">
        <Column tableName="Table1" columnName="Column1"
xsi:type="xsd:long">
          100
        </Column>
        :
        <Column tableName="Table2" columnName="Column1"
xsi:type="xsd:string">

```

```

        some value
    </Column>

    <Column tableName="Table2" columnName="Column2"
xsi:type="xsd:string">
        another value
    </Column>
</Row>
</Description>
</ResultItem>
:
</Output>
:
```

Because of the use of DbUtils and the aggregation inside the QueryBroker, it is necessary to supply at least one column with a primary key for the whole query. This key is represented as the id attribute in the result element's Row.

11.3.2.5 **Configuration of a SQL-adapter**

11.3.2.5.1 *Database connection information*

To specify the values for database connectivity (hostname, port, database name, connector, type, driver, username, password) for the adapter, a configuration file must be supplied inside the WEB-INF/classes folder of the deployed web app. The name of this textfile must be *{FullyQualifiedNameOfAdapter}.properties*, normally this would be:

```
de.uop.dimis.air.adapters.SQLAdapter.properties
```

Here is an example for a configuration textfile:

```
db.hostname=localhost
db.port=5432
db.database=querybroker
db.username=postgres
db.schema=public
db.password=geheim
db.connector=jdbc
db.type=postgresql
```

```
db.driver=org.postgresql.Driver
```

Remark: This principle works for other adapters too, i.e. if it is necessary to provide configuration data for an own adapter an according properties file can be created and located inside the WEB-INF/classes folder.

11.3.2.5.2 *Multiple databases*

To use more than a single database (or to use a single database with different user/-password credentials), the adapter code has to be copied (technically, it is sufficient to only copy the *de.uop.dimis.air.adapters.SQLAdapter* class file) and renamed (e.g., to *de.uop.dimis.air.adapters.NewSQLAdapter*), because the internal list of registered adapters does not allow for two different adapters with the same fully qualifying name.

As configuration information is read from a file with the same name as the adapter, the config file for the new adapter must also be located in the WEB-INF/classes folder and must be named *{FullyQualifiedNameOfNewlyCreatedAdapter}.properties*, in this example:

```
de.uop.dimis.air.adapters.NewSQLAdapter.properties
```

11.3.2.5.3 *Registering the service with the QueryBroker*

Depending on which mode is to be used, the register process is slightly different. In the following examples, *serverRoot* specifies the Tomcat host (e.g., <http://localhost:8080/>), while *serviceID* specifies the fully qualified name of the adapter package (e. g., *de.uop.dimis.air.adapters.SQLAdapter*). The HTTP request mode is in both cases POST.

Normal mode

To register the service, a valid CapabilityDescription XML file has to be provided. It must contain all preferred QueryCondition operators and could look like this example:

```
<?xml version="1.0" encoding="UTF-8"?>

<mpqf:MpegQuery mpqfID="" xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpqf:schema:2008
mpqf_semantic_enhancement.xsd">

  <mpqf:Management>
    <mpqf:Input>
      <mpqf:DesiredCapability>
        <!-- GreaterThan: 100.3.2.5 (Overview in Standard Annex B.2) -
->

        <mpqf:SupportedQueryTypes
href="urn:mpeg:mpqf:2008:CS:full:100.3.2.5" />

        <!-- GreaterThanEqual: 100.3.2.6 (Overview in Standard Annex
B.2) -->
```



```

    <mpqf:SupportedQueryTypes
href="urn:mpeg:mpqf:2008:CS:full:100.3.2.6" />
    <!-- LessThan: 100.3.2.7 (Overview in Standard Annex B.2) -->
    <mpqf:SupportedQueryTypes
href="urn:mpeg:mpqf:2008:CS:full:100.3.2.7" />
    <!-- LessThanEqual: 100.3.2.8 (Overview in Standard Annex B.2)
-->
    <mpqf:SupportedQueryTypes
href="urn:mpeg:mpqf:2008:CS:full:100.3.2.8" />
    <!-- Equal: 100.3.2.9 (Overview in Standard Annex B.2) -->
    <mpqf:SupportedQueryTypes
href="urn:mpeg:mpqf:2008:CS:full:100.3.2.9" />
    <!-- NotEqual: 100.3.2.10 (Overview in Standard Annex B.2) -->
    <mpqf:SupportedQueryTypes
href="urn:mpeg:mpqf:2008:CS:full:100.3.2.10"/>
    </mpqf:DesiredCapability>

<mpqf:ServiceID>de.uop.dimis.air.adapters.SQLAdapter</mpqf:ServiceID>
    </mpqf:Input>
    </mpqf:Management>
</mpqf:MpegQuery>

```

To register with the QueryBroker, this XML string has to be included in the body of the HTTP request in serialized form, while the command itself specifies the keyword *CapabilityDescription* to denote the use of the information in the body.

```
{serverRoot}/QueryBrokerServer/services/{serviceID}/CapabilityDescription
```

Direct mode

To register the service in direct mode, just the following command has to be used (the body does not need to contain any information):

```
{serverRoot}/QueryBrokerServer/services/{serviceID}/QueryByFreeText
```

11.3.3 Implementation of a GIS adapter

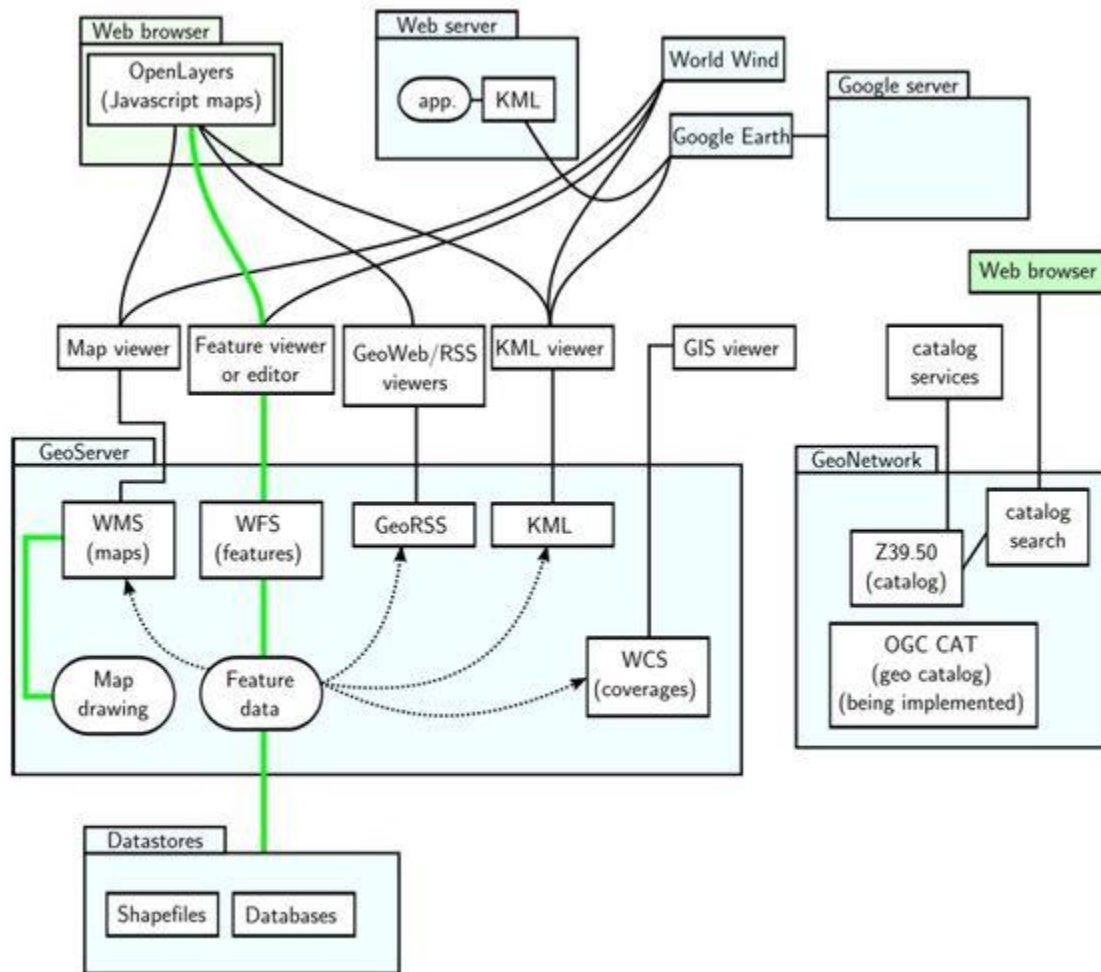
A large number of different web services exist which offer access to geographical information systems (GIS), like the popular map services Google Maps or OpenStreetMap or more specialized data sources

which have cartographic information about weather, forests, mountains, traffic, residential areas and so on. It would be beneficial to combine some of these services into a single result, a map with all specified content overlaid over each other. As there are many different protocols and file formats in the GIS world, it is sometimes not easy to combine them. While the possibility of creating a complete GIS server (which would then combine them) exists, it is an impractical and heavyweight workaround for simpler retrieval tasks.

But the QueryBroker GE multimedia retrieval middleware allows to fill this gap by providing access and combination of these different services in a lightweight manner. In order to demonstrate this two exemplary adapters have been realized.

11.3.3.1 **Background**

Most of the public available services follow the Open Geospatial Consortium's (OGC) standards, namely the Web Map Service (WMS) and the Web Feature Service (WFS). A WMS creates an image file consisting of several queried layers and spanning a bounding box around a given coordinate point and returns it to the client, while a WFS does not render such a map image, but returns the textual information which contains all the information of this map region in XML format for further processing by other programs. The following figure illustrates the architecture of those services. Unfortunately, popular mapping services and data like OpenStreetMap (OSM) or Google Maps do not or only partially follow this standard. Therefore, conversion is needed in these cases, which is done inside the adapter.



Open GIS standards overview

11.3.3.2 Architectural considerations

Because of the modular structure of the QueryBroker, each web service can have its own adapter which knows how to translate the query into the specific commands and how to convert the result information back, if necessary.

11.3.3.2.1 Exemplary adapters

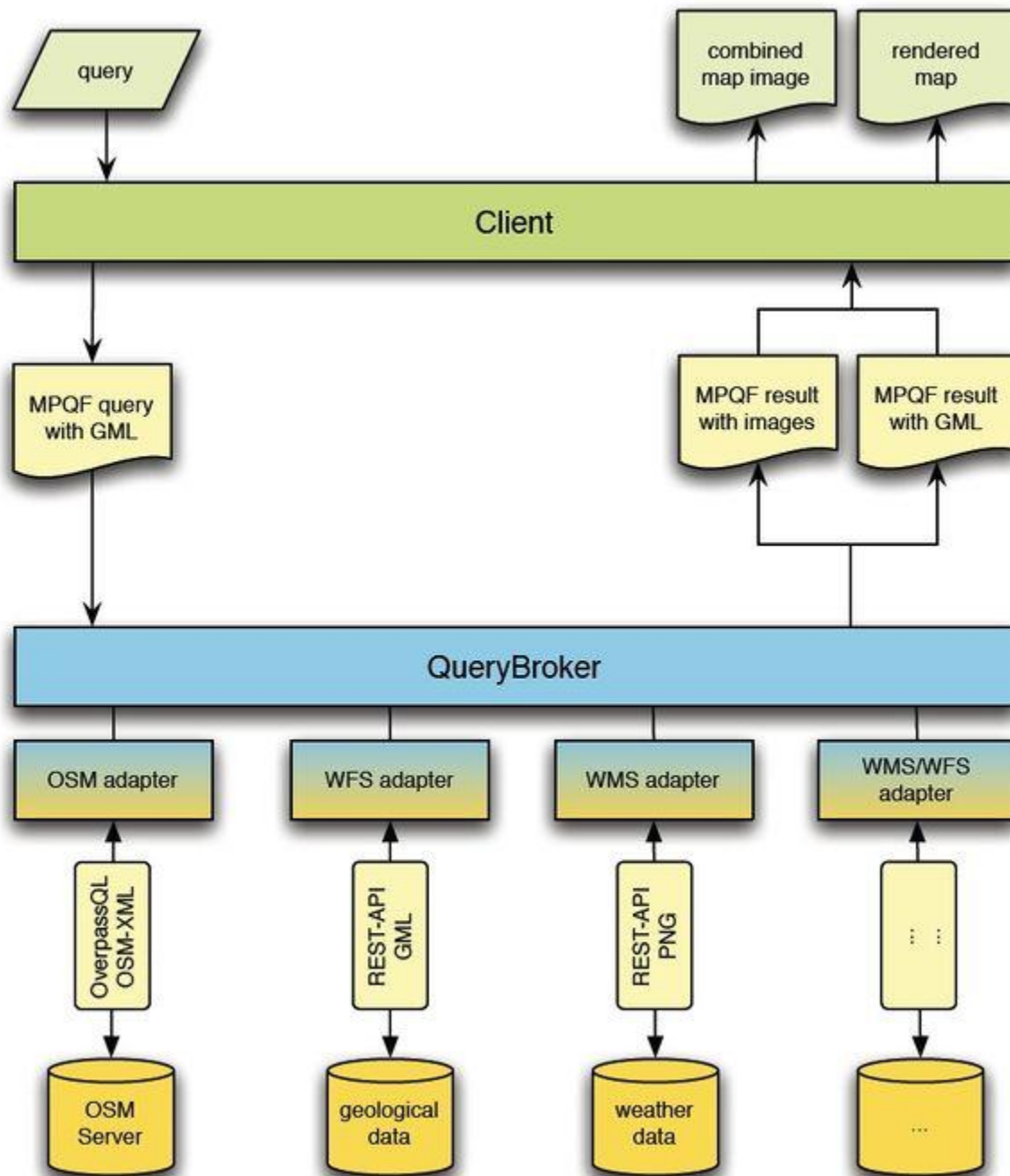
In order to showcase the access of geospatial data sources three example adapters are developed:

- OSM adapter** - The OpenStreetMap adapter communicates with OpenStreetMap (or more specifically, one of their public servers which serve the data). It's XML output can be used mainly as the basis background map because of the broad and deep coverage of the world and also because of the better license in comparison to Google.

- **WFS adapter** - For adding additional layers of information and points of interest, a generic WFS (Web Feature Service) adapter would be good. Such an adapter could be used with all services that adhere to the OGC standard, which makes it quite versatile.
- **WMS adapter** - As the availability of functioning WFS servers is rather sparse, the much more common Web Map Service shall be accessed by an adapter that also conforms to the appropriate OGC standard. Instead of XML, images will be returned, which - although not as flexible as XML - allow for easier use inside the QueryBroker and direct visualization inside the client application.

For a graphical overview of the complete system, please see the figure below. Blue parts represent the Controller, yellow parts the Model and data itself, while green parts represent the View or user interface (MVC model).

Although it looks like the three adapters shown are different, they share certain aspects, like the processing and conversion of the input data. By extending an abstract superclass, those parts can be reused, while the service-specific tasks must be implemented separately. Each adapter class should implement those query capabilities (in their respective special format, like OverpassQL for OSM) and also handle the result conversion back to GML (from their respective result formats, like OSM-XML) in the WFS and OSM cases or simply the returning of result images in the WMS case.



QueryBroker GIS overview

11.3.3.2.2 Supported input

The input must be in GML format and at least contain a coordinate reference system and a region of interest via a bounding box with coordinates according to the GML format specifications.

For reference, see an example query below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```

<mpqf:MpegQuery xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns:wfs="http://www.opengis.net/wfs"

  xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://www.opengis.net/wfs WFS-basic.xsd"
mpqfID="gis_oklahoma">
  <mpqf:Query>
    <mpqf:Input immediateResponse="true">
      <mpqf:OutputDescription maxItemCount="32"/>
      <mpqf:QueryCondition>
        <mpqf:TargetMediaType>text/xml</mpqf:TargetMediaType>
        <mpqf:Condition xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:
          type="mpqf:QueryByDescription" matchType="similar"
preferenceValue="1.0">
          <mpqf:DescriptionResource resourceID="resource_23">
            <mpqf:AnyDescription>
              <wfs:FeatureCollection>
                <gml:boundedBy>
                  <gml:Envelope srsName="urn:ogc:def:crs:EPSG:4326">
                    <gml:coordinates decimal="." cs="," ts=" ">
                      -100.0,25.0 -70.0,40.0
                    </gml:coordinates>
                  </gml:Envelope>
                </gml:boundedBy>
                <gml:featureMember></gml:featureMember>
              </wfs:FeatureCollection>
            </mpqf:AnyDescription>
          </mpqf:DescriptionResource>
        </mpqf:Condition>
      </mpqf:QueryCondition>
    </mpqf:Input>
  </mpqf:Query>
</mpqf:MpegQuery>

```

```
</mpqf:Query>
</mpqf:MpegQuery>
```

This query applies to all registered *QueryByDescription* adapters which return *text/xml* results. The content of the bounding box with lower left corner of $x=-100$ and $y=-25$ and upper right corner of $x=-70$ and $y=40$ in the coordinate reference system *EPSG:4326* should be returned. Note that *srsName* only specifies the query coordinate reference system (CRS), not the result's CRS, which is encoded in the result itself. For a detailed list of available coordinate reference systems and their coverages, see [Spatial Reference](#).

The only difference in querying of WMS adapters would be *<mpqf:TargetMediaType>image/png</mpqf:TargetMediaType>* (and of course a correctly registered WMS-capable adapter).

11.3.3.2.3 Supported output

Each adapter supports output in either image or XML, depending on his class. In a scenario where several different adapters are active, the choice of the client querying must be expressed through the *TargetMediaType* element in the query request, indicating the MIME type the client expects. If *image/jpeg*, *image/gif* or *image/png* is used, a image (with alpha channel transparency if available by the server) is returned from those adapters that are of the WMS type. In a similar way all the adapters that output XML will return their results, if *text/xml* is used.

In both cases, Geography Markup Language (GML) in the format of the WFS schema is used to store the information inside the MPQF request and response. It is also used to aggregate the different services, in case textual representation is required. If image representation is required, the combination of images is left up to the client, because this would require destructive merging of image files inside the QueryBroker itself (this would not be possible with images without alpha channel, and would require a specific order, which cannot be guessed by the QueryBroker, only by the client).

An exemplary output is illustrated below:

```
<mpqf:Output>
  <mpqf:ResultItem recordNumber="1" originID="SQL" confidence="1.0">
    <?xml version="1.0" encoding="UTF-8"?>
    <wfs:FeatureCollection xmlns:ogi="http://ogi.state.ok.us"
xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:wfs="http://www.opengis.net/wfs"
xmlns:gml="http://www.opengis.net/gml">
      <gml:boundedBy>
        <gml:Box
srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
          <gml:coord>
            <gml:X>-104.20301</gml:X>
```

```

        <gml:Y>30.297490000000003</gml:Y>
    </gml:coord>
    <gml:coord>
        <gml:X>-93.79799</gml:X>
        <gml:Y>40.70251</gml:Y>
    </gml:coord>
</gml:Box>
</gml:boundedBy>
<gml:featureMember>
    <ogi:quad100 fid="quad100.5">
        <gml:boundedBy>
            <gml:Box
srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
                <gml:coord>
                    <gml:X>-100.00041404</gml:X>
                    <gml:Y>37.00002376</gml:Y>
                </gml:coord>
                <gml:coord>
                    <gml:X>-99.00037809</gml:X>
                    <gml:Y>37.50002229</gml:Y>
                </gml:coord>
            </gml:Box>
        </gml:boundedBy>
        <ogi:the_geom>
            <gml:MultiPolygon
srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
                <gml:polygonMember>
                    <gml:Polygon>
                        <gml:outerBoundaryIs>
                            <gml:LinearRing>
                                <gml:coordinates>

```



```

-99.00038033,37.50001233
:
</gml:coordinates>
</gml:LinearRing>
</gml:outerBoundaryIs>
</gml:Polygon>
</gml:polygonMember>
</gml:MultiPolygon>
</ogi:the_geom>
</ogi:quad100>
</gml:featureMember>
<gml:featureMember>
  <ogi:quad100 fid="quad100.6"/>
</gml:featureMember>
:
</wfs:FeatureCollection>
</mpqf:ResultItem>
</mpqf:Output>

```

11.3.3.3 **Implementation comments**

11.3.3.3.1 *Requirements on data resources*

As sufficient source data is needed to test and validate the adapters, several issues shall be regarded:

- Because of the two different desired output formats (images and XML text files), services which support both WMS and WFS are preferred.
- The data should be free of charge and without limits in reasonable use ranges.
- No license should restrict the use of the data and the access should be possible without an API key.
- The covered area should be large enough (not only a single city, necessary to measure performance).
- There should be enough different services available (like weather, satellite images, sensor points, landmarks, . . .) from different web services.

11.3.3.3.2 *Selecting map services*

Therefore, OSM is chosen for supplying the base map, while data from the U.S. government is chosen for additional information, as there exist many different services on different servers, all adhering to the same OGC standards (WFS and WMS).

Please note that this choice is done for practical reasons like availability, documentation etc. and does not necessarily show the best or most useful possible use of combined information. It merely acts as a proof-of-concept implementation, and additional services can be implemented with considerably less effort (or none in case of standards-compliant WMS/WFS services).

While implementing the adapters, it quickly became clear that the assumptions made about the servers had to be adjusted to the prevailing situation – some servers were slow, others were down most of the time or for some periods of time. From the seven servers classified as "*working*" by [GeoServer](#), five were not available at time of this writing. Therefore, the initial considerations of overlapping boundaries were taken less into consideration, as this largely depends on proper servers and not so much on the program code itself.

Instead the following four WFS servers were chosen, which have proven to be somewhat reliable:

- [Oklahoma State GIS](#)
- [Institute of Massachusetts GIS](#),
- [Delaware Geologic Information Resource Server](#)
- [New Zealand GIS](#)

For WMS servers the situation is better, as the OpenStreetMap data makes it easy to combine any place of the world with a second service. Additionally the servers are relatively fast, because the result size is fixed for an arbitrary number of confined features. Therefore, the following servers have been chosen:

- [OSM-WMS Uni Heidelberg](#)
- [CubeWerkx MapServer Demo](#)

11.3.3.4 **Usage & Configuration**

11.3.3.4.1 *Adding a new adapter*

To add a new adapter to the QueryBroker, the procedure is similar to adapters of other types.

1. Create a class named *NewAdapter.java* within package *de.uop.dimis.air.adapters* inside of *QueryBroker-Adapters/src/main/java*
2. Create a configuration file named *de.uop.dimis.air.adapters.NewAdapter.properties.default* inside of *QueryBroker-Adapters/src/main/resources*
3. Edit the configuration file according to next section
4. Optionally it is also possible to change the configuration later without recompiling the whole project by adding a new configuration file to the WEB-INF/classes directory of the deployed web

app. This file must follow the same structure as the original and must be named *de.uop.dimis.air.adapters.NewAdapter.properties*

11.3.3.4.2 *Configuring an adapter*

Following is an example of a configuration file for a WFS server. Note that not required configuration parameters like *image size* are empty, but have nevertheless to be included. The same principle applies to WMS servers, which need only the first four settings.

```
gis.url=http://ogi.state.ok.us/geoserver/  
gis.version=1.0.0  
image.width=  
image.height=  
wfs.xml.namespace=http://ogi.state.ok.us  
wfs.xml.prefix=ogi  
wfs.epsg=EPSG:4326
```

The semantics of the parameters are:

- **gis.url:** Required by all services. The URL to the web server, up to and not including the service-specific part. For example, *http://ogi.state.ok.us/geoserver/wfs?request=GetCapabilities* would amount to the URL in the example snippet.
- **gis.version:** Required by all services. The version of the service queried.
- **image.width, image.height:** Required by WMS services only. The dimensions in pixel which the result image should have.
- **wfs.xml.namespace:** Required by WFS services only. The XML namespace of the service. It can normally be found in the GetCapabilities document.
- **wfs.xml.prefix:** Required by WFS services only. The XML prefix that the result XML should have for parts that are specific to the service.
- **wfs.epsg:** Required by all services. Although it is possible to extract this info directly from the servers, the information is hard to parse in some cases, which might lead to unexpected errors. If it is known beforehand, it can be supplied here.

11.3.3.4.3 *Querying an adapter*

After the configuration has been done and the QueryBroker has been deployed, it can be queried in the usual way via the REST API (see [above](#) for an example query).

11.3.3.5 *Supported versions*

Although most servers support many different versions of their services, specific versions have been chosen to reduce complexity in the implementation. It would of course pose no problem to add those versions by implementing a version check and modifying the code that differs. Additional libraries with the OGC schemas in different versions would also have to be added.

11.3.3.5.1 *Supported service versions*

Currently supported versions include:

- **WMS 1.1.1:** Fully supported.
- **WFS 1.0.0:** Because of a [bug](#) in GeoTools 1.1.0 is not possible, so 1.0.0 is chosen. Partly supported, see [next section](#).
- **OSM API v6:** Experimental support, see [next section](#).

11.3.3.5.2 *Unsupported features*

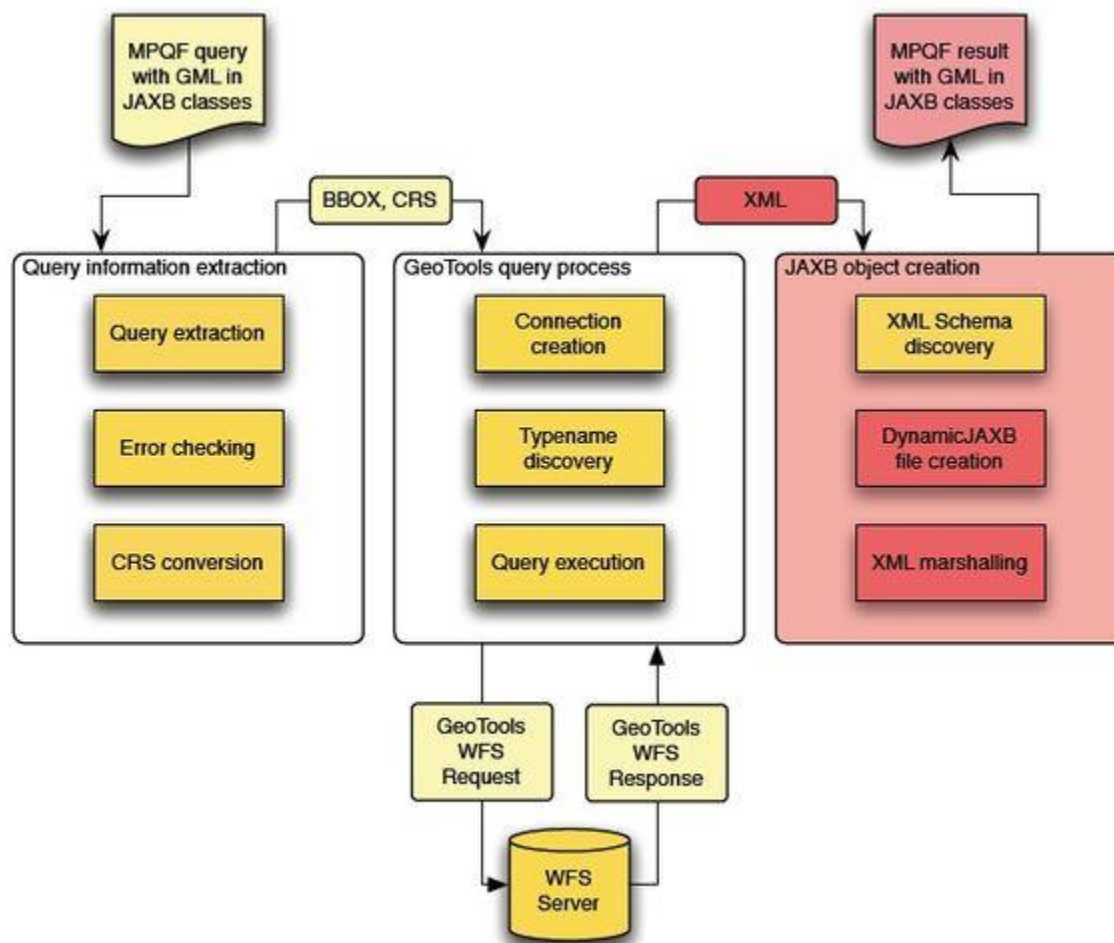
The following smaller features are not yet implemented:

- Automatic size calculation of result images with WMS servers; instead fixed values in the configuration file are used.
- Due to a mission configuration option for setting transparency inside the request object, it is assumed that transparency is wanted as long as the server supports it and it is possible with the file format (for example, PNG). Additionally, it is easy for a client to add a layer of background of any color, but difficult to remove it from an opaque image.
- Z-order when returning layers from WMS servers. The client should make several single requests if the order of the layers is important (this feature is very hard to be implemented in the QueryBroker due to architectural reasons)
- Conversion between different CRS does not work with all reference system strings. It is advised to include the correct EPSG system in the query and in the adapter configuration file, if possible in the simple form EPSG:codeNumber, for example EPSG:4326.
- Only one layer can be requested for WFS adapters. It would require several single connections inside the adapter for more layers, so there is not much difference to requesting these layers directly inside the original query.

11.3.3.6 *Implementation obstacles of WFS and OSM-XML*

11.3.3.6.1 *Overview*

The basic idea of the adapter is a three-step process: first the input object is read and all necessary information is extracted, which yields just the textual information of Bounding Box and CRS. Extending it with parameters from the adapter configuration file allows the creation of a GeoTools datastore, which communicates with the WFS server, executes the query and receives the results. Finally, the XML data received need to be dynamically converted into JAXB objects to be passed back to the QueryBroker (because each query holds its own XML schema).



Process flow of WFS adapter and obstacles

Aside from the minor problems already mentioned before, there is a quite substantial problem:

"Getting the results of the WFS query back into JAXB for use with the QueryBroker."

The above figure visualizes the inner work flow of the WFS adapter in detail. The red parts indicate the zones of problems which could not yet be solved. They will now be discussed in more depth.

11.3.3.6.2 Challenges in detail

A query is always received as a JAXB object with a certain structure, as detailed [above](#). The necessary Java classes for this object are already known beforehand, as all queries share the same syntax and therefore it is no problem to use the compiled classes of the official [OGC WFS and GML standards](#). In contrast to this, on the other side of the adapter there exists not one, but several possible schemas - each for one query - because each XML response has its own schema, which is in most cases a combination of [XLink](#), [GML v2.1.2](#) and a schema specific to the domain of the server with elements that extend the *AbstractFeatureType* of GML. Each WFS server offers several *typeName*s (these are the layers; categories of grouped features, like "only schools" or "geological data from a certain year"), and each of those has its own XML schema. This is also the reason why GeoTools does [not support JAXB](#) at all.

11.3.3.6.3 *Dynamic JAXB*

It is infeasible to try to compile all these schemas by hand and add them to the QueryBroker before issuing the first query. Therefore [DynamicJAXB](#) from the [EclipseLink MOXy](#) project was chosen, because it allows to create a JAXB context and classes on the fly in memory, as long as a XSD definition file is supplied. The URL to this XSD can be found via GeoTools, so it seemed possible to do it this way. Unfortunately it does not work the way it should – there was a [bug](#) which should have been fixed, but still exists. As DynamicJAXB is a relatively new technology, it may be possible that this will be changed in the future. Several people have already been looking for a [solution to this problem](#) in 2013, but without any success.

11.3.3.6.4 *Some notes on OSM-XML*

The XML format used by OpenStreetMap shares some of the problems with the WFS area. On the plus side, the schema is much simpler than the many intertwined OGC schema and – because of the single source where it is used – does not need to be compiled new for every server or every layer that is accessed. This simplifies the usage and allows, with help of a [XSLT transform script](#) the conversion into GML. Unfortunately, this script is incomplete (missing ways) and also outdated since the last revision change to version 0.6; it was adapted by two different users in different [ways](#), without proper documentation or specification of implemented, working and/or missing features.

Because of these problems and also the nonexistent support of OpenStreetMap queries in Geotools (direct HTTP access via REST was used) it was considered to be of lower priority than the more standards-compliant WFS adapter, which would also demonstrate better the ability of the QueryBroker to access several different services instead of the same.

Therefore the adapter is currently in an experimental state, sharing the problematic use of JAXB with the WFS adapter and facing the difficulty of converting the end results into a more useful format. While it would be no problem for the QueryBroker to collect OSM-XML and GML/WFS and present them to the client (considering correct JAXB classes would already exist), the combination of those different formats would be impossible without knowledge of the logic and meaning of the inner content.

To address this general problem, there would be two ways of action: the first possibility would place the responsibility in the hands of the adapters to return only those classes and formats that the QueryBroker could understand and aggregate/optimize. But as the problems with the different WFS variations have shown, this task would be quite complicated. Each adapter would need to inform the query broker about the semantics of his format so that rules about merging similar content can be applied in the aggregation process.

The second possibility would be to create container classes that contain XML streams of the actual results and also additional metadata/information, all supplied by the adapters.

This would of course be a bit of a workaround, as it complicates the general situation even further, without solving the problem of different output structures completely. Also the creation of these classes would require careful planning to include most of the important features and would get complicated easily.

11.3.3.6.5 *Conclusions & best practices*

Summarizing these experiences results in the fact that the task of converting the very dynamic and flexible XML output from GeoTools to the rigid and well-defined JAXB classes can pose serious challenges. Several ideas to circumvent this should be briefly discussed:

Considering the structure of the QueryBroker, which heavily relies on JAXB, it would be very difficult to change this all to normal XML, which is accessed via ordinary parsers. This way there would be no problem accessing and using the XML results, but the ease of use in internal handling of JAXB objects would be replaced by tedious parsing, which does all in all not sound very reasonable.

An alternative could be to implement a simple JAXB container class that holds generic XML input as a bytestream, similar to how images are contained. This way the general structure would be intact and the result XML could be used. The downside is that the inner content would be "dumb" and only unearthed at the client side – no optimization or aggregation of any kind could take place inside the QueryBroker. Despite this disadvantage, the solution seems to balance both sides rather evenly.

The optimal solution would of course be the JAXB route via dynamic creation and with suitable binding files. The creation of those files would pose a serious challenge and a considerable time investment, but if the DynamicJAXB bugs are fixed and if it is possible to construct according consistent classes, the resulting solution would be most elegant and flexible for nearly all WFS servers without further modifications of source code or XSDs.

The above-mentioned problems hindered a successful implementation and provision of an generic WFS adapter; nevertheless the working components (except DynamicJAXB) can be provided upon request.

11.4 Sanity check procedures

Some basic tests to quickly evaluate whether the QueryBroker is up and running.

To ease carrying out the sanity checks for the QB, a short [bash-shell script](#) is provided using cURL running the respective ReST commands (you may need to modify `{serverRoot}`).

11.4.1 End to End testing

To verify the access to the QueryBroker, just issue the following GET request to the QueryBroker Host using a web browser to report the version information of it:

```
http://{serverRoot}/QueryBrokerServer/version/
```

You shall get a response looking like `"Release 3.2 (QB: 2.6.0, QB-A: 0.8.0, QB-S: 1.0.3)"`. It states the release version accompanied by info on the underlying modules.

For further sanity checks the two pseudo services `"de.uop.dimis.air.adapters.DemoQuerybyDescription"` and `"de.uop.dimis.air.adapters.DemoQuerybyMedia"` have been implemented which return fix

responses. The first one accepts QuerybyDescription and the second one QuerybyMedia queries. Using these the following tests can be done:

1. Register 'de.uop.dimis.air.adapters.DemoQuerybyDescription' by

```
POST
http://{serverRoot}/QueryBrokerServer/services/de.uop.dimis.air.adapters.DemoQuerybyDescription/QueryByDescription/
```

Response:

```
Service "de.uop.dimis.air.adapters.DemoQuerybyDescription" registered for capability QueryByDescription ("urn:mpeg:mpqf:2008:CS:full:100.3.6.2").
```

2. Register 'de.uop.dimis.air.adapters.DemoQuerybyMedia' by

```
POST
http://{serverRoot}/QueryBrokerServer/services/de.uop.dimis.air.adapters.DemoQuerybyMedia/QueryByMedia/
```

Response:

```
Service "de.uop.dimis.air.adapters.DemoQuerybyMedia" registered for capability QueryByMedia ("urn:mpeg:mpqf:2008:CS:full:100.3.6.1").
```

3. Submit a 'QuerybyMedia' request

```
POST http://{serverRoot}/QueryBrokerServer/query

with body:

<?xml version="1.0" encoding="UTF-8"?>
<mpqf:MpegQuery mpqfID="ID_e2794b9"
xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpqf:schema:2008
mpqf_semantic_enhancement.xsd">
  <mpqf:Query>
    <mpqf:Input immediateResponse="true">
      <mpqf:QueryCondition>
```



```

        <mpqf:Condition xsi:type="mpqf:QueryByMedia"
matchType="similar">

            <mpqf:MediaResource resourceID="res01">

                <mpqf:MediaResource>

<mpqf:MediaUri>http://any.uri.com</mpqf:MediaUri>

                </mpqf:MediaResource>

            </mpqf:MediaResource>

        </mpqf:Condition>

    </mpqf:QueryCondition>

</mpqf:Input>

</mpqf:Query>

</mpqf:MpegQuery>

```

Response:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<mpegQueryType mpqfID="c587f583-0007-4367-95e3-49cc4cfc36"
xmlns:ns2="JPSearch:schema:coremetadata"
xmlns="urn:mpeg:mpqf:schema:2008"
xmlns:ns3="urn:medico:dicom:schema:2011">

    <Query>

        <Input immediateResponse="true">

            <QueryCondition>

                <Condition xsi:type="QueryByMedia" matchType="similar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

                    <MediaResource resourceID="res01">

                        <MediaResource>

                            <MediaUri>http://any.uri.com</MediaUri>

                        </MediaResource>

                    </MediaResource>

                </Condition>

```

```

        </QueryCondition>
    </Input>
    <Output>
        <ResultItem originID="QueryByMedia">
            <Description>
                <ns2:jpSearchCoreType>
                    <ns2:Identifier>ResultItem3</ns2:Identifier>
                </ns2:jpSearchCoreType>
            </Description>
        </ResultItem>
        <ResultItem originID="QueryByMedia">
            <Description>
                <ns2:jpSearchCoreType>
                    <ns2:Identifier>ResultItem2</ns2:Identifier>
                </ns2:jpSearchCoreType>
            </Description>
        </ResultItem>
        <SystemMessage>
            <Status>
                <Code>1</Code>
                <Description>Query was successful</Description>
            </Status>
        </SystemMessage>
    </Output>
</Query>
</mpegQueryType>

```

4. Submit a 'QuerybyDescription' request

```
POST http://{serverRoot}/QueryBrokerServer/query
```

with body:

```
<?xml version="1.0" encoding="UTF-8"?>

<mpqf:MpegQuery mpqfID="001" xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpqf:schema:2008
mpqf_semantic_enhancement.xsd">

    <mpqf:Query>
        <mpqf:Input>
            <mpqf:QueryCondition>
                <mpqf:Condition xsi:type="mpqf:QueryByDescription"
matchType="exact">
                    <mpqf:DescriptionResource resourceID="desc001">
                        <mpqf:AnyDescription
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004 M7v2schema.xsd">
                            <mpeg7:Mpeg7>
                                <mpeg7:DescriptionUnit
xsi:type="mpeg7:CreationInformationType">
                                    <mpeg7:Creation>
                                        <mpeg7:Title>Miracle Query
Format</mpeg7:Title>
                                    </mpeg7:Creation>
                                </mpeg7:DescriptionUnit>
                            </mpeg7:Mpeg7>
                        </mpqf:AnyDescription>
                    </mpqf:DescriptionResource>
                </mpqf:Condition>
            </mpqf:QueryCondition>
        </mpqf:Input>
    </mpqf:Query>
</mpqf:MpegQuery>
```

Response:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mpegQueryType mpqfID="f823eae0-163a-4e99-9e7d-2c162e854803"
xmlns:ns2="JPSearch:schema:coremetadata"
xmlns="urn:mpeg:mpqf:schema:2008"
xmlns:ns3="urn:medico:dicom:schema:2011">

  <Query>
    <Input>
      <QueryCondition>
        <Condition xsi:type="QueryByDescription"
matchType="exact" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
          <DescriptionResource resourceID="desc001">
            <AnyDescription>
              <mpeg7:Mpeg7
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xmlns:mpqf="urn:mpeg:mpqf:schema:2008">
                <mpeg7:DescriptionUnit
xsi:type="mpeg7:CreationInformationType">
                  <mpeg7:Creation>
                    <mpeg7:Title>Miracle Query
Format</mpeg7:Title>
                  </mpeg7:Creation>
                </mpeg7:DescriptionUnit>
              </mpeg7:Mpeg7>
            </AnyDescription>
          </DescriptionResource>
        </Condition>
      </QueryCondition>
    </Input>
    <Output>

```

```
<ResultItem originID="QueryByDescription">
  <Description>
    <ns2:jpSearchCoreType>
      <ns2:Identifier>ResultItem1</ns2:Identifier>
    </ns2:jpSearchCoreType>
  </Description>
</ResultItem>
<ResultItem originID="QueryByDescription">
  <Description>
    <ns2:jpSearchCoreType>
      <ns2:Identifier>ResultItem2</ns2:Identifier>
    </ns2:jpSearchCoreType>
  </Description>
</ResultItem>
<SystemMessage>
  <Status>
    <Code>1</Code>
    <Description>Query was successful</Description>
  </Status>
</SystemMessage>
</Output>
</Query>
</mpegQueryType>
```

5. Submit a 'QuerybyMedia' OR 'QuerybyDescription' request

```
POST http://{serverRoot}/QueryBrokerServer/query
```

with body:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<mpqf:MpegQuery mpqfID="101" xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpqf:schema:2008
mpqf_semantic_enhancement.xsd">
    <mpqf:Query>
        <mpqf:Input>
            <mpqf:QueryCondition>
                <mpqf:Condition xsi:type="mpqf:OR">
                    <mpqf:Condition xsi:type="mpqf:QueryByMedia">
                        <mpqf:MediaResource resourceID="ID_5001">
                            <mpqf:MediaResource>
                                <mpqf:MediaUri>http://tolle.uri/1</mpqf:MediaUri>
                                </mpqf:MediaResource>
                            </mpqf:MediaResource>
                        </mpqf:Condition>
                        <mpqf:Condition
xsi:type="mpqf:QueryByDescription">
                            <mpqf:DescriptionResource
resourceID="desc001">
                                <mpqf:AnyDescription
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004 M7v2schema.xsd">
                                    <mpeg7:Mpeg7>
                                        <mpeg7:DescriptionUnit
xsi:type="mpeg7:CreationInformationType">
                                            <mpeg7:Creation>
                                                <mpeg7:Title>Miracle Query
Format</mpeg7:Title>
                                            </mpeg7:Creation>
                                        </mpeg7:DescriptionUnit>
                                    </mpeg7:Mpeg7>
                                </mpqf:AnyDescription>
                            </mpqf:DescriptionResource>

```

```

        </mpqf:Condition>
    </mpqf:Condition>
</mpqf:QueryCondition>
</mpqf:Input>
</mpqf:Query>
</mpqf:MpegQuery>

```

which will return the union 'ResultItem1', 'ResultItem2', and 'ResultItem3' (xml output omitted).

6. Submit a 'QuerybyMedia' AND 'QuerybyDescription' request

```

POST http://{serverRoot}/QueryBrokerServer/query

with body:

<?xml version="1.0" encoding="UTF-8"?>
<mpqf:MpegQuery mpqfID="101" xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpqf:schema:2008
mpqf_semantic_enhancement.xsd">
    <mpqf:Query>
        <mpqf:Input>
            <mpqf:QueryCondition>
                <mpqf:Condition xsi:type="mpqf:AND">
                    <mpqf:Condition xsi:type="mpqf:QueryByMedia">
                        <mpqf:MediaResource resourceID="ID_5001">
                            <mpqf:MediaResource>

<mpqf:MediaUri>http://tolle.uri/1</mpqf:MediaUri>

                            </mpqf:MediaResource>
                        </mpqf:MediaResource>
                    </mpqf:Condition>

```

```

        <mpqf:Condition
xsi:type="mpqf:QueryByDescription">

            <mpqf:DescriptionResource
resourceID="desc001">

                <mpqf:AnyDescription
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004 M7v2schema.xsd">

                    <mpeg7:Mpeg7>

                        <mpeg7:DescriptionUnit
xsi:type="mpeg7:CreationInformationType">

                            <mpeg7:Creation>

                                <mpeg7:Title>Miracle Query
Format</mpeg7:Title>

                                    </mpeg7:Creation>

                                </mpeg7:DescriptionUnit>

                            </mpeg7:Mpeg7>

                        </mpqf:AnyDescription>

                    </mpqf:DescriptionResource>

                </mpqf:Condition>

            </mpqf:Condition>

        </mpqf:QueryCondition>

    </mpqf:Input>

</mpqf:Query>
</mpqf:MpegQuery>

```

which will return just 'ResultItem2' (xml output omitted)

11.4.2 List of Running Processes

- 1 Application Server / Servlet-Container (e.g. Tomcat) instance, plus the QueryBroker processes started in Tomcat

11.4.3 Network interfaces Up & Open

- TCP:8080 (if Tomcat is employed)

- Further depend on the connection types to the data repositories to be registered at the QueryBroker

11.4.4 Databases

- The GE itself has no database. However, it connects to external data repositories to be registered by the user in order to perform the queries.

11.5 Diagnosis Procedures

The Media-enhanced QueryBroker reports logical errors concerning the submitted query via the REST response. Otherwise it logs with Apache Tomcat logging anyway. The log files are located in the Apache Tomcat directory `./logs`.

11.5.1 Resource availability

For running the QueryBroker GE on desktop based systems at least a CPU with 2-4 cores and physical RAM of about 2G-4GB should be available.

- The required RAM is dependent on the size of the return query results, as the queries are processed in-memory; especially if you are requesting large images and did not limit the response number a memory shortage may happen.
- For optimal throughput not more than 2 simultaneous queries per CPU core shall be processed.

11.5.2 Remote Service Access

Currently the QueryBroker GE has no built-in integration with other GEs.

The configuration of probable data repository connectors requires the provision of according class files in the Class Path and their management is conducted via REST commands.

11.5.3 Resource consumption

The resources used by the GE depend on the amount of data being processed, especially on the size of the returned results as these are processed in memory. There are no typical numbers. However, less than 200 MB of free memory and more than 85% of CPU load over more than 10 seconds may lead to an abnormal behavior of the GE implementation (i.e., the Media-enhanced QueryBroker).

11.5.4 I/O flows

The main input/output uses the port tomcat runs on (typically: 80 or 8080); the use of other ports depends on data resources to be registered at the QB.

12 Semantic Application Support - Installation and Administration Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

12.1 Introduction

This guide covers the steps needed to install and configure the Semantic Application Support GE. Also, the steps needed to install and configure the related GUI, based on WebProtégé. WebProtégé is a web application that runs in a servlet container, such as Tomcat. WebProtégé comes as a war file that can be easily deployed in a servlet container.

12.2 System Requirements

In order to deploy the Semantic Application Support GE the following software must be previously installed:

- JavaTM Platform, Standard Edition Development Kit (JDKTM) 6[\[1\]](#).
- Maven 2.1.1 [\[2\]](#)
- JBoss AS 7.1.0 Final [\[3\]](#)
- Apache Tomcat 6.0.32 [\[4\]](#)
- Sesame 2.6.6 [\[5\]](#)
- OWLIM SE 5.1 [\[6\]](#)
- MySQL Community Server GA 5.1.63 [\[7\]](#)

Besides the above software, is needed for the GUI:

- mongoDB [\[8\]](#) - used to store different WebProtege configuration data
- and a data directory where WebProtégé will store all its data.

Online installation documentation is available for all required software in case it is needed.

12.3 Instalation guidelines

This guide defines the procedure to install the Semantic Web Application Support. For the sake of simplicity, all the commands and procedures included in this guide are oriented to a Linux server, but, as the Semantic Web Application Support is a JEE application, it can be easily installed on a windows server.

12.3.1 Configuring the infrastructure

In order to deploy a working instance of the GE a suitable JEE datasource and Sesame plus OWLIM knowledge base should be needed. In this section we will learn how to deploy a new datasource using MySQL and JBoss AS and how to create a new knowledge base using Sesame console plus OWLIM.

12.3.1.1 *Creating a Datasource*

The first step towards creating a new MySQL datasource is to install the MySQL JDBC driver. If you have already done it in your JEE container, you can just skip this step. In JBoss AS, we should need to:

- Create a folder in \$JBOSS_HOME/modules/com/mysql/main
- Copy the MySQL 5.1 java connector jar file into the created folder.
- Create a module.xml file into the created folder containing:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
~ JBoss, Home of Professional Open Source.
~ Copyright 2010, Red Hat, Inc., and individual contributors
~ as indicated by the @author tags. See the copyright.txt file in
the
~ distribution for a full listing of individual contributors.
~
~ This is free software; you can redistribute it and/or modify it
~ under the terms of the GNU Lesser General Public License as
~ published by the Free Software Foundation; either version 2.1 of
~ the License, or (at your option) any later version.
~
~ This software is distributed in the hope that it will be useful,
~ but WITHOUT ANY WARRANTY; without even the implied warranty of
~ MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
~ Lesser General Public License for more details.
~
~ You should have received a copy of the GNU Lesser General Public
```

```
~ License along with this software; if not, write to the Free
~ Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
~ 02110-1301 USA, or see the FSF site: http://www.fsf.org.
-->

<module xmlns="urn:jboss:module:1.0" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.15.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
  </dependencies>
</module>
```

Once the connector is installed, we would need to add the datasource definition itself into our JEE container. The reference documentation for JBoss datasource deployment descriptor can be found in [\[9\]](#). To do so, we would need to modify the file `$JBOSS_HOME/standalone/configuration/standalone.xml`, including the following description under the `<datasources>` tag:

```
<datasource jta="false" jndi-name="java:jboss/datasources/ExampleDS"
pool-name="ExampleDS" enabled="true" use-ccm="false">
  <connection-url>CONNECTION_URL</connection-url>
  <driver-class>com.mysql.jdbc.Driver</driver-class>
  <driver>mysql</driver>
  <security>
    <user-name>USER_NAME</user-name>
    <password>PASS</password>
  </security>
  <validation>
    <validate-on-match>false</validate-on-match>
    <background-validation>false</background-validation>
```

```

        <background-validation-millis>0</background-validation-millis>
    </validation>

    <statement>

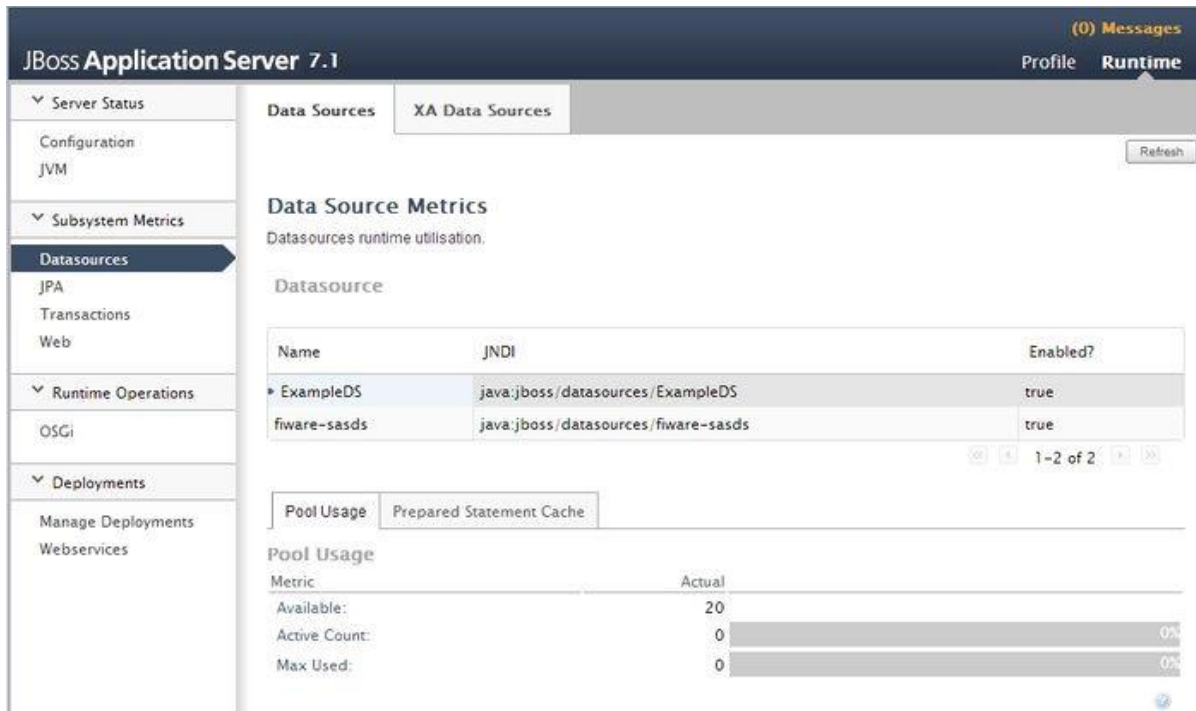
        <prepared-statement-cache-size>0</prepared-statement-cache-
size>

        <share-prepared-statements>false</share-prepared-statements>
    </statement>
</datasource>

```

As the GE would try to create the needed tables and sequences, the user provided for the data source connection should have the proper permissions to create, drop, insert or alter tables into the data base.

Once defined, you should be able to see the created datasource into your JEE management tool e.g.:



The screenshot shows the JBoss Application Server 7.1 Runtime console. The left sidebar has a navigation menu with 'Databases' selected. The main area is titled 'Data Source Metrics' and shows 'Datasources runtime utilisation'. Below this is a table of data sources:

Name	JNDI	Enabled?
ExampleDS	java:jboss/datasources/ExampleDS	true
fiware-sasds	java:jboss/datasources/fiware-sasds	true

Below the table is a 'Pool Usage' section with a table showing metrics for 'ExampleDS' and 'fiware-sasds'.

Metric	Actual	Max	Usage
Available:	20		
Active Count:	0	0	0%
Max Used:	0	0	0%

12.3.1.2 Creating a Knowledge Base

In order to create the knowledge repository, the sesame console would be used. To do so, we would run:

```

$ cd SESAME_DIR/bin
$ ./console.sh

```

The following output should be shown in the console:

```
14:02:08.976 [main] DEBUG info.aduna.platform.PlatformFactory -  
os.name = linux  
  
14:02:08.981 [main] DEBUG info.aduna.platform.PlatformFactory -  
Detected Posix platform  
  
Connected to default data directory  
  
Commands end with '.' at the end of a line  
Type 'help.' for help  
>
```

Then, we would need to connect to the registry. To do so:

```
connect http://klab.development.atosresearch.eu:8280/openrdf-sesame.  
Disconnecting from default data directory  
Connected to http://klab.development.atosresearch.eu:8280/openrdf-  
sesame
```

Now we can create a new OWLIM-SE repository. To do so:

```
> create owlim-se.  
Please specify values for the following variables:  
Repository ID [owlim-se-test]: owlim-se-test  
Repository title [OWLIM-SE test repository]:  
Storage folder [storage]:  
License file (leave blank for evaluation):  
Rule-set [owl-horst-optimized]: owl2-rl  
Base URL [http://example.org/owlim#]:  
Imported RDF files('; ' delimited):  
Default namespaces for imports('; ' delimited):  
Entity index size [200000]:
```

```
Total cache memory [80m]:
Main index memory [80m]:
Use predicate indices [false]:
Predicate index memory [0]:
Full-text search memory [0]:
Full-text search indexing policy [never]:
Full-text search literals only [true]:
Use PCSOT index [false]:
Use PTSOC index [false]:
Cache literal language tags [false]:
Repository created
>
```

In this case, we have created a knowledge base called `owlim-se-test` that uses `owl2-rl` rule set. Now we should need to configure our GE binaries to work with the database and knowledge base created.

12.3.2 Deploying the GE

This section describes the steps needed to deploy a working version of the Semantic Web Application Support GE. To do so, we would need to download the binaries from the FI-WARE private SVN (or contact GE owner to get a configured binarie), configure the GE and deploy the resulting war file in our JEE container.

12.3.2.1 *Getting the software*

Semantic Web Application Support source code can be downloaded from the FI-WARE SVN (with the proper user and password) by using the following command:

```
svn checkout --username <YOUR_USER_NAME> https://forge.fi-ware.eu/scmrepos/svn/data/trunk/SemanticApplicationSupport .
```

By doing so, a set of Maven projects should be downloaded into your current directory:

Ontology Registry

- `ontology-registry-rest-service`, that contains the wrapper that allows REST access style to the GE functionality.
- `ontology-registry-service-ejb`, that contains the JEE application implementing the ontology registry business logic.

- ontology-registry-test, that contains the software needed to test a clean installation of the GE.

Java Semantic Repository Connection (JSRC)

- JSRC, that contains the Java interfaces to allow the interaction with semantic repositories.
- Sesame2610, that contains a implementation of JSRC for Sesame 2.6.10

Semantic Workspaces Management

- SemanticWorkspaces, that contains the JEE application implementing the Semantic Workspaces Management
- semantic-workspaces-service, that contains the wrapper that allows REST access style for semantic workspaces management
- semantic-workspaces-test, that contains the wrapper that allows REST access style for semantic workspaces management

12.3.2.2 *Configuring the software*

Once the Maven projects are downloaded, we would need to modify the configuration files to connect the GE with a proper data source and knowledge base. To do so we would need to modify two files:

- /ontology-registry-service-ejb/src/main/resources/META-INF/persistence.xml: Here we would need to configure the GE to use the data source created in section "Configuring the infrastructure" (or any other suitable database).

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
    <persistence-unit name="ontology-registry-service-ejb">
        <jta-data-source>DATA_SOURCE_NAME (e.g.
java:jboss/datasources/ExampleDS)</jta-data-source>

        <class>eu.atosresearch.ontologyregistry.jpa.OntologyJPA</class>
        <properties>
            <property name="hibernate.hbm2ddl.auto"
value="update"/>
```



```

        </properties>

    </persistence-unit>

</persistence>

```

- `/SemanticWorkspaces/src/main/resources/META-INF/persistence.xml`: Here we would need to configure the database access for Semantic Workspaces Management

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence
    xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
    version="1.0">
    <persistence-unit name="eu.atosresearch.semanticworkspaces.beans">

        <class>eu.atosresearch.semanticworkspaces.beans.Workspace</class>

        <class>eu.atosresearch.semanticworkspaces.beans.OntologyRegistry</class>

        <class>eu.atosresearch.semanticworkspaces.beans.OntologyInstance</class>

        <properties>
            <property name="hibernate.archive.autodetection"
value="class, hbm"/>
            <property name="hibernate.connection.driver_class"
value="com.mysql.jdbc.Driver"/>
            <property name="hibernate.connection.password"
value="[DATABASE_PASSWORD]"/>
            <property name="hibernate.connection.url"
value="jdbc:mysql://[MYSQL_SERVER]/workspacemanagement"/>

```

```

        <property name="hibernate.connection.username"
value="[DATABASE_USER]" />

        <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect" />

        <property name="hibernate.c3p0.min_size" value="5" />
        <property name="hibernate.c3p0.max_size" value="20" />
        <property name="hibernate.c3p0.timeout" value="300" />
        <property name="hibernate.c3p0.max_statements"
value="50" />
        <property name="hibernate.c3p0.idle_test_period"
value="3000" />
        <property name="hibernate.hbm2ddl.auto" value="update" />
    </properties>
</persistence-unit>
</persistence>

```

- `/ontology-registry-service-ejb/src/main/java/eu/atosresearch/ontologyregistry/services/registryConfig.properties`: Here we would need to configure the GE to connect to the knowledge base created in section "Configuring the infrastructure" (or any other sesame plus owlim knowledge base). We would also need to configure the registry base that will be used to generate the urls for ontologies loaded into the registry.

```

OntologyRegistry.registryBase=REGISTRY_BASE (e.g.
http://localhost:8080/ontology-registry-
service/webresources/ontology-registry/ontologies)

OntologyRegistry.repositoryId=KNOWLEDGE_BASE_NAME (e.g. ontology-
registry)

OntologyRegistry.sesamePass=KNOWLEDGE_BASE_PASSWORD (e.g.
registryPass)

OntologyRegistry.sesameServer=KNOWLEDGE_BASE_SESAME_SERVER (e.g.
http://klab.development.atosresearch.eu:8280/openrdf-sesame)

OntologyRegistry.sesameUser=KNOWLEDGE_BASE_USER (e.g. sesameadmin)

```

- `/semantic-workspaces-service/src/main/java/config.properties`: We would also need to configure the Semantic Workspaces Management to allow manage repositories. The follow configuration uses a Sesame configuration

```
workspaces.manager.connectionstring=SESAME_ENDPOINT (Sesame endpoint
which will be used to store the workspaces e.g.
http://localhost:8080/openrdf-sesame)

workspaces.manager.type=eu.atosresearch.jsrc.sesame2610.Sesame2610Driver

workspaces.manager.user=KNOWLEDGE_BASE_MANAGER_USER (e.g. sesame-
admin)

workspaces.manager.password=KNOWLEDGE_BASE_MANAGER_PASSWORD (e.g.
sesamepassword)

workspaces.ontologyregistry.url=DEFAULT_ONTOLOGY_REGISTRY (Default
Ontology Registry that will be used to retrieve ontologies e.g.
http://localhost:8180/ontology-registry-service/webresources/ontology-
registry)

workspaces.basecontext=BASE_CONTEXT (Base URI that will be used to
create context e.g. http://localhost/semantic-workspaces/)
```

After modifying the files, we would need to build the project in order to generate the proper binaries to install. To do so, we should run the following commands:

```
$ cd <directory containing ontology-registry-service-ejb pom.xml file>
$ mvn clean install

$ cd <directory containing ontology-registry-rest-service pom.xml
file>
$ mvn clean install

$ cd <directory containing JSRC pom.xml file>
$ mvn clean install

$ cd <directory containing Sesame2610 pom.xml file>
$ mvn clean install

$ cd <directory containing SemanticWorkspaces pom.xml file>
$ mvn clean install

$ cd <directory containing semantic-workspaces-service pom.xml file>
$ mvn clean install
```

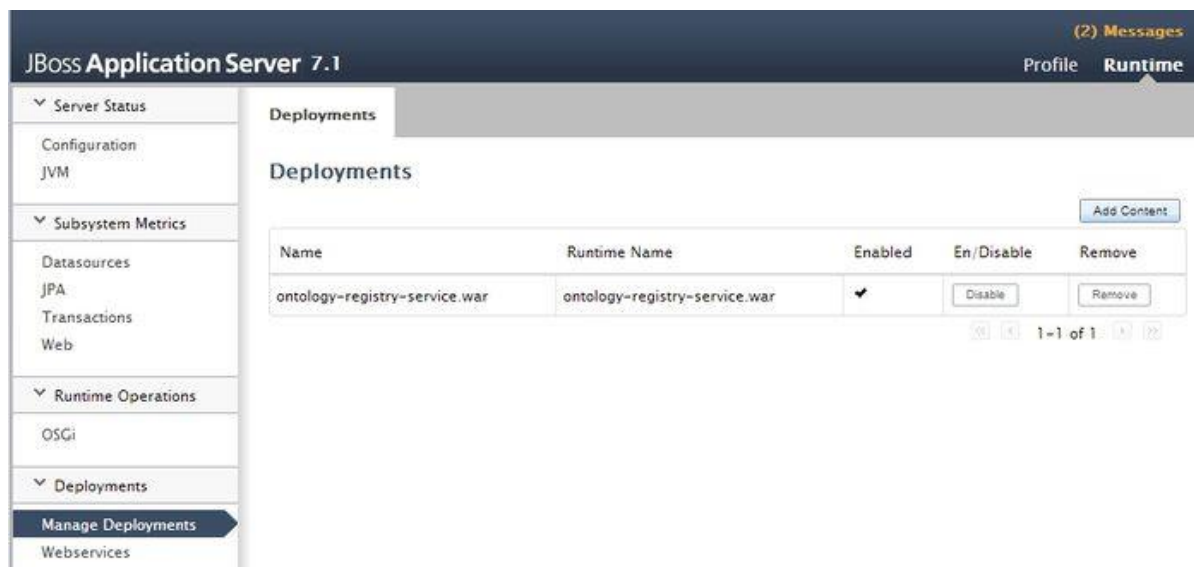
Two war files (ontology-registry-service.war and semantic-workspaces-service.war) to be deployed will be generated under /ontology-registry-rest-service/target and /semantic-workspaces-service/target

12.3.2.3 *Deploying the software*

Once built, you would need to deploy the `ontology-registry-service.war` to a JEE container. To do so, you should copy the war file to the deployments directory e.g.:

```
$ cd <directory ontology-registry-rest-service>/target
$ cp ontology-registry-service.war $JBOSS_HOME/standalone/deployments
$ cd <directory semantic-workspaces-service>/target
$ cp semantic-workspaces-service.war $TOMCAT_HOME/webapps/
```

You can also use any other deployment tool provided by your JEE container e.g.:



12.3.3 GUI

12.3.3.1 *Installation*

12.3.3.1.1 *Install mongoDB*

WebProtégé uses mongoDB[\[10\]](#) to store various configuration. We may use it in the future to store also other types of data. WebProtégé will not start, if mongoDB is not installed.

Follow the mongoDB installation instructions[\[11\]](#) for your operating system. The installation is easy, and usually requires no or very little configuration.

As part of the installation process, you may be required to create a folder where mongoDB will keep its data (e.g., on Windows, default location is `c:\data\db`, Linux: `/data/db`), and give access to the mongoddb

user to that folder. The mongoDB website has easy-to-follow, step-by-step installation instructions[\[12\]](#) for all operating systems.

Ideally, mongoDB should run as a service. If that is not possible, you need to make sure to start it manually, as explained in the installation website.

For your information, the mongoDB default host is localhost and the default port is 27017.

12.3.3.1.2 *Create a data directory*

WebProtégé stores all of its data (ontologies, project configurations, change history, etc.) in a data directory. Please create a data directory outside of your webapp folder (so that updates will be easy later). This directory needs to exist prior to starting WebProtégé, and the user that runs tomcat (or your servlet container), needs to be able to write to this folder.

For example, on Linux or OSX, you may create a data directory /data/webprotege, by typing in a console:

```
mkdir /data/webprotege  
chown tomcat /data/webprotege
```

Note. In the chown command, if tomcat is running under a different user name, just replace tomcat with the correct user name. You may need to use sudo in front of the commands.

For example, on Windows, you can create a directory C:\data\webprotege by typing in a Command Prompt:

```
md data  
md data\webprotege
```

You can use any directory on your filesystem, as long as the tomcat user can write to it.

12.3.3.2 **Deploying WebProtégé**

Deployment is easy, just copy the webprotege.war into the webapps folder of your servlet container. If using tomcat, copy webprotege.war into tomcat_install_dir/webapps/.

There are two configuration properties that WebProtege needs in order to run:

```
data.directory - this is the data directory where WebProtege keeps  
all of its data and you created in the previous step  
  
application.host - this is the URL under which WebProtege is  
deployed, without the http part. For example:  
application.host=webprotege.stanford.edu
```

There are three ways to configure these two required properties for WebProtege (just pick the one that is easier for you).

- (1) **Set them as Java arguments** using the '-D' option when starting your servlet container. For example, on Linux and OSX, if you are using tomcat, you can edit the catalina.sh and add this line (adapt to your own paths):

```
JAVA_OPTS="$JAVA_OPTS -Dwebprotege.data.directory=/data/webprotege -Dwebprotege.application.host=webprotege.stanford.edu"
```

Please note that the property names are prefixed with webprotege. On Windows systems, edit the catalina.bat.

- or -

- (2) **Set them as environment variables.** Check your operating system documentation on how to set environment variables. For example,

```
webprotege.data.directory=/data/webprotege  
webprotege.application.host=webprotege.stanford.edu
```

- or -

- (3) **Set them in the webprotege.properties** file that is available under webprotege webapps folder. This file can be used to store other properties as well. Documentation on all the properties is available here [\[13\]](#). If the webprotege.properties file does not exist in your expanded webprotege folder in tomcat/webapps, just create one, or copy the one under etc/webprotege.properties.template and rename it to webprotege.properties.

For example, your webprotege.properties could look like (Linux example):

```
data.directory=/data/webprotege  
application.host=webprotege.stanford.edu
```

Please note that the property names are not prefixed by 'webprotege.', as in the first two cases.

For Windows systems, the data directory path needs to use the Windows separator, so it will look like:

```
data.directory=c:\\data\\webprotege
```

Please, refer the WebProtégé Administrator's Guide webpage [\[14\]](#) for more detailed information about installation.

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. 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.

12.4.1 End to End testing

To verify access to the access to the Semantic Application Support Ontology Manager API, just issue the following GET request to the Location GE Host using a web browser :

```
http://<SAS GE server ip>:8180/ontology-registry-  
service/webresources/ontology-registry/mgm/list
```

As a result a xml document containing a single empty <ontologies/> tag should be returned. Also, you can verify access to the access to the Semantic Application Support - Semantic Workspaces Manager API, just issue the following GET request to the Location GE Host using a web browser :

```
http://<SAS GE server ip>:8080/semantic-workspaces-  
service/rest/workspaces/mgm/list
```

As a result a xml document containing a single empty <workspaces/> tag should be returned.

12.4.2 List of Running Processes

- 1 Tomcat instance, plus the Sesame and OWLIM processes started in Tomcat
- 1 JBoss instance
- 1 MySQL RDBMS

12.4.3 Network interfaces Up & Open

- TCP:8080 (Tomcat + Sesame + OWLIM)
- TCP:8180 (JBoss AS)

12.4.4 Databases

Semantic Application Support GE relies in a set of MySQL databases:

ontologyregistry

This database store all the data related with ontology versioning. In order to test this instance, the next statements must be executed:

```
mysql -u [DB_USER] -h [MSQL_SERVER] -D ontologyregistry  
>select * from ontologyjpa
```

workspacemanagement

This database store all the data related with the semantic workspaces management. In order to test this instance, the next statements must be executed:

```
mysql -u [DB_USER] -h [MSQL_SERVER] -D workspacemanagement  
>select * from workspace
```

Also OLWIM and Sesame may also be used to persist the RDF triples using a database as backend (optional). Check the [Sesame Manual](#) for more information on this matter.

12.5 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.

The following sections have to be filled in with the information or an “N/A” (“Not Applicable”) where needed. Do not delete section titles in any case.

12.5.1 Resource availability

- For the NeOn Toolkit client, client-side of the enabler: A simple common PC or laptop with at least 2GB RAM is enough to run the NeOn toolkit. The NeOn Toolkit is multiplatform and can be used on Linux or Windows.
- For the server-side API: The main critical requirements come from the OWLIM deployment. OWLIM provides [documentation](#) that studies different configurations and the hardware requirements, from a single machine with RAM of 4GB and no more than 100GB of hard-disk space, to a cluster of multicore servers with 32GB RAM each. A good on-server configuration requirements are to have at least 12GB of RAM and 500GB of hard disk.

12.5.2 Remote Service Access

The component provides a REST API to allow the remote access. This API is described in the User Guide.

12.5.3 Resource consumption

The resource consumption depends heavily on the amount of triples being uploaded, as OWLIM provides materialization of inferred knowledge in real-time. Nevertheless the usage of OWLIM in the GE does not pose in principle major threats in terms of inference. Therefore no major resource consumption is envisaged for a regular usage of the GE.

12.5.4 I/O flows

As was mentioned before, the GE provides a REST API which is only used by external user (i.e. Use Cases)

13 StreamOriented - Installation and Administration Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

13.1 Kurento Installation

This guide describes how to install the Stream-Oriented GE - Kurento. Kurento is composed of different nodes, namely the Kurento Application Server (KAS), the Kurento Media Server (KMS) and the KMF Media Connector (KMC). These three nodes can be installed in the same or different hosts. This guide focuses on the installation on a single machine, but comments are done where appropriate explaining how to modify the configuration files for a dual installation.

13.1.1 Prerequisites

Hardware minimal recommended requirements:

- 8GB RAM
- 16GB HDD (this figure is not taking into account that multimedia streams could be stored in the same machine. If so, HDD size must be increased accordingly)

Operating system requirements:

- Currently Kurento Media Server (KMS) only runs on Ubuntu Linux (32 or 64 bits). It is highly recommended using version 13.10 or newer due to the dependency of KMS with gstreamer.
- Kurento Application Server (KAS) and KMF Media Connector (KMC) can run in any platform that supports JDK version 7.

13.1.2 Kurento Application Server (KAS)

First, install *Open JDK 7* and "unzip" packages:

```
$ sudo apt-get update
$ sudo apt-get install openjdk-7-jdk unzip
```

Download *JBoss*, uncompress it and move it to */opt/jboss* by executing:

```
$ sudo wget http://download.jboss.org/jbossas/7.1/jboss-as-7.1.1.Final/jboss-as-7.1.1.Final.tar.gz
$ sudo tar xfvz jboss-as-7.1.1.Final.tar.gz && sudo mv jboss-as-7.1.1.Final /opt/jboss
```

To avoid running JBoss as root create the user *jboss*, the group *jboss* and make that user the owner of JBoss files and folders:

```
$ sudo adduser --system jboss && sudo addgroup jboss
$ sudo chown -R jboss:jboss /opt/jboss/
```

Create the startup/stop script by copying the following content to a new file called */etc/init.d/jboss7*:

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          jboss
# Required-Start:    kurentod
# Required-Stop:
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: JBoss Application Server
# Description:       init script for JBoss Application Server
### END INIT INFO

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
NAME="jboss"
JBOSS_HOME="/opt/jboss"
DAEMON="$JBOSS_HOME/bin/standalone.sh"
SHUTDOWN_CMD="$JBOSS_HOME/bin/jboss-cli.sh"
DAEMON_USER=jboss

PIDFILE=/var/run/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME
DESC="JBoss AS Server"

if [ -r "/lib/lsb/init-functions" ]; then
    . /lib/lsb/init-functions
else
```

```
    echo "E: /lib/lsb/init-functions not found, package lsb-base needed"
    exit 1
fi

# Include defaults if available
if [ -f /etc/default/jboss7 ] ; then
    . /etc/default/jboss7
fi

verify_user () {
# Only root can start Kurento
    if [ `id -u` -ne 0 ]; then
        log_failure_msg "Only root can start JBoss"
        exit 1
    fi
}

if [ "$START_JBOSS" != "true" ]; then
    log_failure_msg "Review activate settings within file
/etc/default/jboss7"
    exit 1
fi

if [ ! -e $JBOSS_HOME ]; then
    log_failure_msg "Unable to access JBoss home directory at:
$JBOSS_HOME"
    exit 1
fi

#[ -z "$BIND_IP" ] && BIND_IP=12.0.0.1
```

```

#[ -n "$DAR_PATH" ] && DAR_PATH="-
Djavax.servlet.sip.dar=file://$DAR_PATH"

JBOSS_OPTS="$JBOSS_OPTS -Djboss.bind.address=0.0.0.0 -
Djboss.bind.address.management=0.0.0.0"

case "$1" in
    start)
        log_daemon_msg "Starting $DESC" "$NAME"
        verify_user

        # Verify pid file directory exists
        if [ ! -e /var/run ]; then
            install -d -m755 /var/run || { log_failure_msg "Unable
to access /var/run directory"; exit 1; }
        fi

        # Make sure HOME directory belongs to $DAEMON_USER
        sudo -u $DAEMON_USER -H [ -O $JBOSS_HOME/standalone/log ]
        if [ $? != 0 ]; then
            chown -R $DAEMON_USER $JBOSS_HOME/* || {
log_failure_msg "Unable to access $JBOSS_HOME"; exit 1; }
        fi

        /sbin/start-stop-daemon --start --pidfile $PIDFILE \
            --chuid $DAEMON_USER --chdir $JBOSS_HOME/bin --
background --make-pidfile --no-close \
            --startas $DAEMON -- $JBOSS_OPTS > /dev/null

        log_end_msg $?
        ;;

    stop)
        log_daemon_msg "Stopping $DESC" "$NAME"

```

```

        # This will just kill the standalone script. Java process
detaches :(
        /sbin/start-stop-daemon --stop --quiet --pidfile $PIDFILE
\
        --chuid $DAEMON_USER --startas $DAEMON
    if [ $? -eq 0 ]; then
        # Send kill command to JBoss
        $SHUTDOWN_CMD --connect command=:shutdown
        rm -f $PIDFILE
        log_end_msg 0
    fi
    ;;

restart|force-reload)
    echo -n "Restarting $DESC: $NAME"
    /sbin/start-stop-daemon --stop --quiet --pidfile $PIDFILE
\
        --exec $DAEMON
    rm -f $PIDFILE
    sleep 1
    echo -e
    $0 start
    ;;

*)
    echo "Usage: $0 {start|stop|restart|force-reload}" >&2
    exit 1
    ;;

esac

exit 0

```

Grant *jboss* user *execution* rights to run the startup/stop script:

```
$ sudo chmod 755 /etc/init.d/jboss7
```

Create the file `/etc/default/jboss7` with the following content (this file is used by the startup/stop script):

```
# Defaults for JBoss7 initscript
# sourced by /etc/init.d/jboss7
# installed at /etc/default/jboss7 by the maintainer scripts

#
# This is a POSIX shell fragment
#

#uncommment the next line to allow the init.d script to start jboss
START_JBOSS=true

# Additional options that are passed to the service.
BIND_IP=0.0.0.0
JBOSS_OPTS=""

# whom the daemons should run as
JBOSS_USER=jboss
```

Finally, configure the server to run JBoss when booted:

```
$ sudo update-rc.d jboss7 defaults
```

13.1.3 Kurento Media Server (KMS)

In order to add Personal Package Archive or PPA's repositories, the `python-software-properties` package must be installed:

```
$ sudo apt-get install python-software-properties
```

Install KMS by typing the following commands, one at a time and in the same order as listed here. When asked for any kind of confirmation, reply affirmatively:

```
$ sudo add-apt-repository ppa:kurento/kurento
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install libevent-dev kurento
```

Finally, configure the server to run KMS when booted:

```
$ sudo update-rc.d kurento defaults
```

13.1.4 KMF Media Connector (KMC)

The KMF Media Connector is a proxy that allows to clients connect to Kurento Media Server through websockets. The main Kurento Media Server interface is based on thrift technology, and this proxy made necessary conversions between websockets and thrift.

Download *KMF Media Connector* and move it to */opt/kmf-media-connector* by executing:

```
$ sudo wget http://builds.kurento.org/release/stable/kmf-media-connector.zip
$ sudo mkdir /opt/kmf-media-connector && sudo mv kmf-media-connector.zip /opt/kmf-media-connector
```

Unzip the *kmf-media-connector.zip* file:

```
$ cd /opt/kmf-media-connector && sudo unzip kmf-media-connector.zip
```

Install *KMF Media Connector* as a service using the following script:

```
$ sudo ./bin/install.sh
```

Finally, configure the server to run *kmf-media-connector* when booted:

```
$ sudo update-rc.d kmf-media-connector defaults
```

Now *KMF Media Connector* has been installed and started. If you want to stop it, simply execute:

```
$ sudo service kmf-media-connector stop
```

To start again KMF Media Connector:

```
$ sudo service kmf-media-connector start
```

Alternatively, you can run the KMF Media Connector passing the arguments directly from the command line:

```
$ cd /opt/kmf-media-connector/lib
$ java -jar kmf-media-connector.jar --server.port=8888 --
thrift.kms.address=127.0.0.1:9090 --thrift.kmf.address=127.0.0.1:9900
```

This can also be launched in background using the following command:

```
$ cd /opt/kmf-media-connector/lib
$ nohup java -jar kmf-media-connector.jar --server.port=8888 --
thrift.kms.address=127.0.0.1:9090 --thrift.kmf.address=127.0.0.1:9900
&
```

KMF Media Connector understands a number of arguments that can be used to call the application, these arguments are:

- `server.port`: The http/websocket port of the proxy. This port will be used for the clients to connect to the port. If not specified, the value 8888 will be used.
- `thrift.kmf.address` : The IP address and port of the Kurento Media Server. If not specified, the address 127.0.0.1:9090 will be used.
- `thrift.kmf.address` : The IP address and port that Kurento Media Server will use to connect to the proxy. If not specified, the address 127.0.0.1:9900 will be used.
- `oauthserver.url`: The url of the ouath service used to authenticate the client requests. The url "<http://cloud.lab.fi-ware.org>" is the official OAuth service in FI-WARE project. If not specified, all clients can use the proxy (that is, no authentication is enforced).

Or if you prefer, instead of using the command line parameters, you can specify these properties in a configuration file. The only requirements to use the configuration file is that this file must be called 'application.properties' and it must be placed inside the *config* directory in the working directory.

The file is formatted as a plain Java properties file:

```
server.port=8888
```



```
thrift.kms.address=127.0.0.1:9090
thrift.kmf.address=127.0.0.1:9900
oauthserver.url=http://cloud.lab.fi-ware.org
```

13.1.5 Kurento Network Configuration.

13.1.5.1 *Running Kurento Without NAT configuration*

KMS can receive requests from the Kurento Application Server (KAS) and from final users. The IP addresses and ports to receive these requests are configured in the configuration file */etc/kurento/kurento.conf*. After a fresh install that file looks like this:

```
[Server]
sdpPattern=pattern.sdp
service=Thrift

[HttpEPServer]
#serverAddress=localhost

# Announced IP Address may be helpful under situations such as the
server needs

# to provide URLs to clients whose host name is different from the one
the

# server is listening in. If this option is not provided, http server
will try

# to look for any available address in your system.
# announcedAddress=localhost

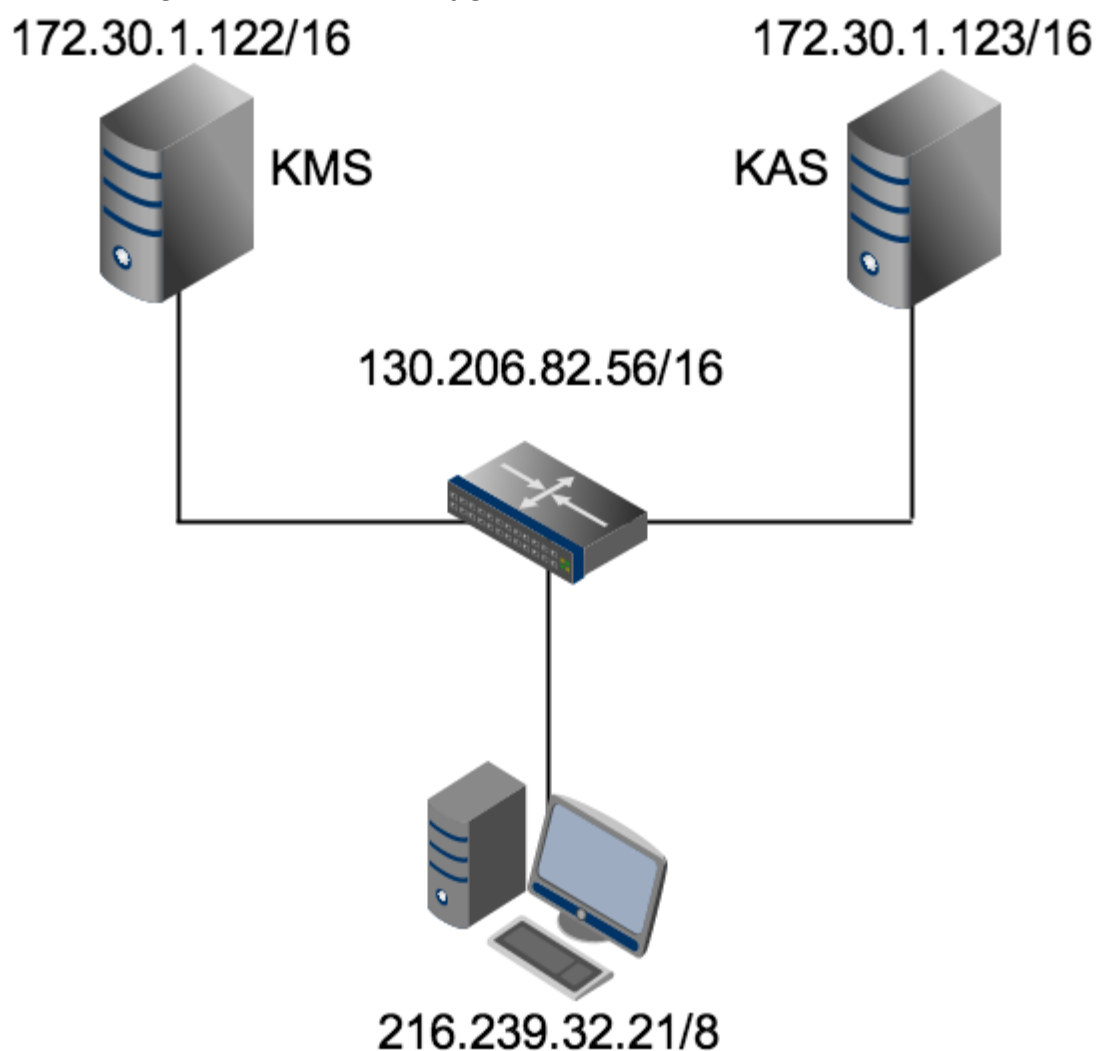
serverPort=9091

[WebRtcEndPoint]
#stunServerAddress = xxx.xxx.xxx.xxx
#stunServerPort = xx
```

```
#pemCertificate = file  
  
[Thrift]  
serverPort=9090
```

That configuration implies that only requests done through Thrift are accepted. By default Thrift server will be attached in all available network interfaces. The section *[Thrift]* allows to configure the port where KMS will listen to KAS requests. The section *[HttpEPServer]* controls the IP address and port to listen to the final users.

13.1.5.2 Running Kurento With NAT configuration



This network diagram depicts a scenario where a NAT device is present. In this case, the client will access the public IP 130.206.82.56, which will connect him with the external interface of the NAT device. KMS serves media on a specific address which, by default, is the IP of the server where the service is running. This would have the server announcing that the media served by an Http Endpoint can be

consumed in the private IP 172.30.1.122. Since this address is not accessible by external clients, the administrator of the system will have to configure KMS to announce, as connection address for clients, the public IP of the NAT device. This is achieved by changing the value of `announcedAddress` in the file `/etc/kurento/kurento.conf` with the appropriate value. The following lines would be the contents of this configuration file for the present scenario.

```
[Server]
sdpPattern=pattern.sdp
service=Thrift

[HttpEPServer]
#serverAddress=localhost

# Announced IP Address may be helpful under situations such as the
server needs

# to provide URLs to clients whose host name is different from the one
the

# server is listening in. If this option is not provided, http server
will try

# to look for any available address in your system.
announcedAddress=130.206.82.56

serverPort=9091

[WebRtcEndPoint]
#stunServerAddress = xxx.xxx.xxx.xxx
#stunServerPort = xx
#pemCertificate = file

[Thrift]
serverPort=9090
```

13.1.6 Sample application and videos

To test part of the functionality of Kurento, a sample app called fi-lab-demo can be used. Next steps in this document focus on how to download the sample app and the complementary video files that are needed.

Download the test video with the following commands:

```
$ sudo wget http://files.kurento.org/video/video.tar.gz
$ sudo tar xfvz video.tar.gz && sudo mv video/ /opt/video && sudo
chown -R jboss:jboss /opt/video
```

And download the fi-lab-demo.war file using the following command:

```
$ sudo wget http://builds.kurento.org/release/stable/fi-lab-demo.war
$ sudo mv fi-lab-demo.war /opt/jboss/standalone/deployments && sudo
chown -R jboss:jboss /opt/jboss/standalone/deployments/fi-lab-demo.war
```

13.1.7 Verifying and starting the servers

To verify that the installation has finished successfully start JBoss by typing:

```
$ sudo /etc/init.d/jboss7 start
```

Open a browser and verify that the default root web page work properly:

```
http://<Service_IP_address>:8080/
```

To verify that the installation has finished successfully start KMS by typing:

```
$ sudo /etc/init.d/kurento start
```

A good way to ensure the state of KAS and KMS is checking out the logs files:

- KMS: /var/log/kurento/media-server.log
- KMC: /var/log/kurento/media-connector.log
- KAS: /opt/jboss/standalone/log/server.log

These files consist a very useful tool for developers to trace errors.

13.2 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.

13.2.1 End to End testing

Open a Chrome or Firefox web browser and type the URL:

`http://<Replace_with_KMS_IP_Address>:8080/fi-lab-demo/`

This will show the web page of the fi-lab-demo sample application. From this web page you can view two links:

13.2.1.1 *HTTP Player*

If you click on this link you can see a drop-down control in the top of the web page. This drop-down show you the different media formats used in this demo. Please select one and click over the Play button:

- WEBM video: After clicking over the "Play" button you can see a short film of "Sintel", independently produced by the Blender Foundation.
- MOV video: After clicking over the "Play" button you can see a short film of "Big Buck Bunny", independently produced by the Blender Foundation.
- MKV video: After clicking over the "Play" button you can see a short film of Japanese animation.
- 3GP video: After clicking over the "Play" button you can see a short tv ad of Blackberry mobile phones.
- OGV video: After clicking over the "Play" button you can see a short video of Pacman.
- MP4 video: After clicking over the "Play" button you can see a short tv ad of Google Chrome.
- JackVader Filter video: After clicking over the "Play" button you can see a video showing the use of filters, in this video a overlayed "pirate hat" is used when a face is detected in the right side of the screen and "Dark Vader mask" is used when a face is detected in the left side of the screen.

13.2.1.2 *HTTP Player with JSON protocol*

This link will load another web page in your browser where you can see the same videos using JSON-based representations for information exchange. The JSON protocol enhances a HTTP Player by implementing a signaling communication between the client (JavaScript API) and the Kurento Application Server (KAS). Using this protocol the client will be able to negotiate the transfer of media using SDP (Session Description Protocol), and also it will be notified with media and flow execution events.

Select one of the videos from the drop-down control located in the top of the web page.

- WEBM video: After clicking over the "Play" button you can see a short film of "Sintel", independently produced by the Blender Foundation.
- MOV video: After clicking over the "Play" button you can see a short film of "Big Buck Bunny", independently produced by the Blender Foundation.
- MKV video: After clicking over the "Play" button you can see a short film of Japanese animation.
- 3GP video: After clicking over the "Play" button you can see a short tv ad of Blackberry mobile phones.
- OGV video: After clicking over the "Play" button you can see a short video of Pacman.
- MP4 video: After clicking over the "Play" button you can see a short tv ad of Google Chrome.
- JackVader Filter video: After clicking over the "Play" button you can see a video showing the use of filters, in this video a overlayed "pirate hat" is used when a face is detected in the right side of the screen and "Dark Vader mask" is used when a face is detected in the left side of the screen.
- ZBar Filer video: After clicking over the "Play" button you can see a video to show the potential of filters. In this video three QR Codes are shown, in the media event text box you can see how the media server detects the different QR codes.

In the text boxes Status, Flow Events and Media Events you can see the results of the different actions that are interpreted by the media server.

If any problem happens with these Sanity Checks, please take a look to the KMS/KAS logs files to trace the error.

13.2.2 List of Running Processes

To verify that KAS is up and running type the following:

```
$ ps -ef | grep jboss
```

The output should be similar to:

```
jboss      4115      1  0 15:16 ?          00:00:00 /bin/sh
/opt/jboss/bin/standalone.sh -Djboss.bind.address=0.0.0.0 -
Djboss.bind.address.management=0.0.0.0

jboss      4159    4115 30 15:16 ?          00:00:08 java -D[Standalone] -
server -XX:+UseCompressedOops -XX:+TieredCompilation -Xms64m -Xmx512m
-XX:MaxPermSize=256m -Djava.net.preferIPv4Stack=true -
Dorg.jboss.resolver.warning=true -
Dsun.rmi.dgc.client.gcInterval=3600000 -
Dsun.rmi.dgc.server.gcInterval=3600000 -
Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true
-Djboss.server.default.config=standalone.xml -
Dorg.jboss.boot.log.file=/opt/jboss/standalone/log/boot.log -
```

```
Dlogging.configuration=file:/opt/jboss/standalone/configuration/loggin
g.properties -jar /opt/jboss/jboss-modules.jar -mp /opt/jboss/modules
-jaxpmodule javax.xml.jaxp-provider org.jboss.as.standalone -
Djboss.home.dir=/opt/jboss -Djboss.bind.address=0.0.0.0 -
Djboss.bind.address.management=0.0.0.0

kuser      4256   2371   0 15:16 pts/0      00:00:00 grep --color=auto
jboss
```

To verify that KMS is up and running use the command:

```
$ ps -ef | grep kurento
```

The output should be similar to:

```
nobody    22527      1   0 13:02 ?          00:00:00 /usr/bin/kurento
kuser     22711    2326   0 13:10 pts/1      00:00:00 grep --color=auto
kurento
```

13.2.3 Network interfaces Up & Open

Unless configured otherwise, KAS listens on the port 8080 to receive HTTP requests from final users. JBoss additionally opens several managemnts ports, namely 4447, 9990, and 9999. If any of those ports is not available in the machine hosting the KAS, it can be changed by editing the following file:

```
$ sudo vim /opt/jboss/standalone/configuration/standalone.xml
```

Concretely, the configuration to be changed is the following:

```
<socket-binding name="management-native"
interface="management" port="{jboss.management.native.port:9999}"/>

<socket-binding name="management-http" interface="management"
port="{jboss.management.http.port:9990}"/>

<socket-binding name="management-https" interface="management"
port="{jboss.management.https.port:9443}"/>
```

To verify the ports opened by KAS execute the following command:

```
$ sudo netstat -putan | grep java
```

The output should be similar to the following:

```

tcp        0      0 0.0.0.0:4447          0.0.0.0:*
LISTEN    4424/java

tcp        0      0 0.0.0.0:9990          0.0.0.0:*
LISTEN    4424/java

tcp        0      0 0.0.0.0:9999          0.0.0.0:*
LISTEN    4424/java

tcp        0      0 0.0.0.0:8080          0.0.0.0:*
LISTEN    4424/java

```

The two additional ports listened are 4447, jBoss remoting port, and 9999, a port for jBoss native management interface.

In addition, if KMS Media Connector has been started, port 8888 will also listening.

Unless configured otherwise, KMS opens the port 9090 to receive HTTP TCP requests from KAS and port 9091 for HTTP TCP requests from final users. To verify the open ports type the command:

```
$ sudo netstat -putan | grep kurento
```

The output should be similar to the following:

```

tcp        0      0 127.0.0.1:9091        0.0.0.0:*
LISTEN    22527/kurento

tcp6       0      0 :::9090               :::*
LISTEN    22527/kurento

```

13.2.4 Databases

N/A

13.3 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.

13.3.1 Resource availability

To guarantee the right working of the enabler RAM memory and HDD size should be at least:

- 8GB RAM
- 16GB HDD (this figure is not taking into account that multimedia streams could be stored in the same machine. If so, HDD size must be increased accordingly)

13.3.2 Remote Service Access

If KMS and KAS are deployed as separate GEs, the admin needs to ensure that the KMS GE can reach the KAS Handler port (default 9091) and that the KAS GE can reach the KMS service port (default 9090).

13.3.3 Resource consumption

Resource consumption documented in this section has been measured in two different scenarios:

- Low load: all services running, but no stream being served.
- High load: heavy load scenario where 100 streams are requested at the same time.

Under the above circumstances, the "top" command showed the following results in the hardware described below:

Machine Info		
	KAS	KMS
Machine Type	Virtual Machine	Physical Machine
CPU	1 Intel Core 2 Duo @ 2,4Ghz	Intel(R) Xeon(R) CPU E5-2620 0 @ 2GHz
RAM	4GB	4GB
HDD	250GB	10GB
Operating System	Mac OS X 10.6.8	Ubuntu 13.10

KAS showed the following results:

KAS

	Low Usage	Heavy Usage
RAM	96MB	200,6MB
CPU	0.2%	44.9%
I/O HDD	1.44GB	1.69GB

KMS gave the following result:

KMS		
	Low Usage	Heavy Usage
RAM	122.88MB	1.56GB
CPU	0.3%	34.6%
I/O HDD	1.18GB	2.47GB

13.3.4 I/O flows

Unless configured otherwise, the GE will open the following ports:

- KAS opens the port 8080 to receive HTTP TCP requests from final users. KAS also opens port 9191 to receive Thrift TCP requests from the KMS.
- KMS opens port 9091 to receive HTTP TCP requests from KAS and final users. KMS also opens the port 9090 to receive Thrift TCP requests from KAS.
- KMC opens the port 8888 to receive HTTP TCP requests from final users. KMC also opens port 9900 to receive Thrift TCP requests from the KMS.

Ports 8080, 9091, and 8888 should be accessible from final users. Therefore these ports should be open and forwarded on existing network elements, such as NAT or Firewall.