
hp Unified Correlation Analyzer



Unified Correlation Analyzer for Event Based Correlation Version 3.0

Topology Extension

Edition: 1.0

For the HP-UX (11.31) and Linux (RHEL 5.8 & 6.3) Operating Systems

June 2013

© Copyright 2013 Hewlett-Packard Development Company, L.P.

Legal Notices

Warranty

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

License Requirement and U.S. Government Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

© Copyright 2013 Hewlett-Packard Development Company, L.P.

Trademark Notices

Adobe®, Acrobat® and PostScript® are trademarks of Adobe Systems Incorporated.

HP-UX Release 10.20 and later and HP-UX Release 11.00 and later (in both 32 and 64-bit configurations) on all HP 9000 computers are Open Group UNIX 95 branded products.

Java™ is a trademark of Oracle and/or its affiliates.

Microsoft®, Internet Explorer, Windows®, Windows Server®, and Windows NT® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Firefox® is a registered trademark of the Mozilla Foundation.

Google Chrome® is a trademark of Google Inc.

Oracle® is a registered U.S. trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of The Open Group.

X/Open® is a registered trademark, and the X device is a trademark of X/Open Company Ltd. in the UK and other countries.

Red Hat® is a registered trademark of the Red Hat Company.

Linux® is a registered trademark of Linus Torvalds in the U.S. and other countries.

Neo4j is a trademark of Neo Technology.

Contents

Preface	7
Chapter 1	9
Introduction	9
Chapter 2	10
UCA for EBC Topology Extension installation	10
2.1 Licensing	10
2.2 Disk requirements	11
2.3 Software prerequisites	11
2.3.1 UCA for EBC Server	11
2.3.2 Java	12
2.4 UCA for EBC Topology Extension package installation	12
2.4.1 Untar the archive in a temporary directory	12
2.4.2 Run the installation script	13
2.4.3 File organization	13
2.5 Un-installation of UCA for EBC Topology Extension	13
Chapter 3	15
UCA for EBC Topology Extension Configuration	15
3.1 Configuring the topology extension	15
3.1.1 Starting an Embedded Topology server	15
3.1.2 Using an external Topology server	16
3.2 Configuring the topology data load function	17
3.3 Configuring the topology dataload scheduler	17
Chapter 4	20
Managing the Topology from the UCA for EBC GUI	20
4.1 The Topology Manager Panel	20
4.2 The Data load Panel	22
Chapter 5	23
Developing a topology aware Value Pack	23
5.1 Introduction	23
5.2 Topology aware Value Pack principles	23
5.3 Alarm enrichment with topology information	24
5.3.1 Alarm Object extension	24
5.3.2 Activating Alarm Enrichment	27
5.3.3 Topology requests implementation	28

5.3.4	Writing rules based on topology data	29
5.4	Topology Value Pack example description.....	29
5.4.1	Objectives	30
5.4.2	Topology aware Value Pack validation	31
5.4.3	Building and Deploying the topology-example Value Pack	32
Chapter 6		34
Creating a new topology aware Value Pack using the UCA eclipse plug-in		34
6.1	Topology aware Value Pack project creation	34
Chapter 7		36
Topology Data load		36
7.1	Introduction.....	36
7.2	CSV files location.....	36
7.3	CSV files names	37
7.4	CSV files format	37
7.4.1	Node files: *.node.csv	38
7.4.2	Relationship files: *.rel.csv	42
7.5	Updates	43
7.6	Deletions	44
7.7	Neo4j transactions	44
7.8	Examples.....	44
7.8.1	Rocks bands.....	44
7.8.2	Network entities	45
7.9	Run the data load	46
7.9.1	From UTM	46
7.9.2	From the UCA GUI.....	47
7.9.3	From Java or Scala code (in Unit tests)	47
7.9.4	Through the Neo4j server plugin and REST API	47
7.10	Data Load output	48
7.10.1	Load Report	48
7.10.2	Discarded entities	49
7.11	Configuration	49
7.12	Logging.....	50
Glossary		52

Figures

Figure 1 - Alarm enrichment with topology information	24
Figure 2 - Topology aware scenario creation using the Eclipse plugin	34
Figure 3 - Graph visualization in Neoclipse.....	46

Tables

Table 1 - Software versions 7

Table 2 - Disk Requirements for UCA for EBC Topology Extension on HP-UX..... 11

Table 3 - Disk Requirements for UCA for EBC Topology Extension on Linux 11

Table 4 - Software Prerequisites for UCA for EBC Topology Extension (HP-UX) 11

Table 5 - Software Prerequisites for UCA for EBC Topology Extension (Linux) 12

Table 6 - UCA for EBC Topology Extension provided files 13

Preface

This manual is dedicated to the Unified Correlation Analyzer for Event Based Correlation Topology Extension. It encompasses both the installation and configuration guide, and the user's guide.

Product Name: Unified Correlation Analyzer for Event Based Correlation

Product Version: V3.0

Intended Audience

Here are some recommendations based on possible reader profiles:

- Solution Developers
- Software Development Engineers

Software Versions

The term UNIX is used as a generic reference to the operating system, unless otherwise specified.

The software versions referred to in this document are as follows:

Product Version	Supported Operating systems
UCA for Event Based Correlation Server Version 3.0	<ul style="list-style-type: none">• HP-UX 11.31 for Itanium• Red Hat Enterprise Linux Server release 5.8 & 6.3
UCA for Event Based Correlation Channel Adapter Version 3.0	<ul style="list-style-type: none">• HP-UX 11.31 for Itanium• Red Hat Enterprise Linux Server release 5.8 & 6.3
UCA for Event Based Correlation Topology Extension Version 3.0	<ul style="list-style-type: none">• HP-UX 11.31 for Itanium• Red Hat Enterprise Linux Server release 5.8 & 6.3
UCA for Event Based Correlation Software Development Kit Version 3.0	<ul style="list-style-type: none">• Windows XP / Vista• Windows Server 2007• Windows 7• Red Hat Enterprise Linux Server release 5.8 & 6.3

Table 1 - Software versions

Typographical Conventions

`Courier` Font:

- Source code and examples of file contents.
- Commands that you enter on the screen.
- Pathnames

- Keyboard key names

Italic Text:

- Filenames, programs and parameters.
- The names of other documents referenced in this manual.

Bold Text:

- To introduce new terms and to emphasize important words.

Associated Documents

The following documents contain useful reference information:

References

[R1] *Unified Correlation Analyzer for Event Based Correlation Reference Guide*

[R2] *Unified Correlation Analyzer for Event Based Correlation Value Pack Development Guide*

[R3] *OSS Open Mediation Installation and Configuration Guide*

[R4] *Unified Correlation Analyzer for Event Based Correlation Installation Guide*

Support

Please visit our HP Software Support Online Web site at www.hp.com/go/hpsoftwaresupport for contact information, and details about HP Software products, services, and support.

The Software support area of the Software Web site includes the following:

- Downloadable documentation,
- Troubleshooting information,
- Patches and updates,
- Problem reporting,
- Training information,
- Support program information.

Chapter 1

Introduction

This guide is a document dedicated to the Topology Extension of the UCA for EBC product.

The Topology extension allows developing Value packs that require topology knowledge for implementing the correlation use case.

The following chapters will describe:

- The installation procedure.
- The configuration of the extension.
- The development guide for Value Packs using the Topology.

Throughout this document, we use the `${UCA_EBC_HOME}` environment variable to reference the root directory (“static” part) of UCA for EBC. The default value for the `${UCA_EBC_HOME}` environment variable is `/opt/UCA-EBC`. The `${UCA_EBC_HOME}` environment variable thus references the `/opt/UCA-EBC` directory unless UCA for EBC “static” part has been installed in an alternate directory.

We also use `${UCA_EBC_DATA}` environment variable to reference the data directory (“variable” part) of UCA for EBC. The default value for the `${UCA_EBC_DATA}` environment variable is `/var/opt/UCA-EBC`. The `${UCA_EBC_DATA}` environment variable thus references the `/var/opt/UCA-EBC` directory unless UCA for EBC “variable” part has been installed in an alternate directory.

Since UCA-EBC V2.0, on Linux and HP-UX systems, the `${UCA_EBC_DATA}` directory may contain multiple instances of UCA-EBC. In this document, we will use the value `${UCA_EBC_INSTANCE}` for referring to `${UCA_EBC_DATA}/instances/<instance-name>` directory on Linux/HP-UX systems and to `${UCA_EBC_DATA}` on Windows systems.

Note that at installation time on Linux/HP-UX, a single `<instance-name>` is configured: default.

For more information on the UCA for EBC product, please refer to the *Unified Correlation Analyzer for Event Based Correlation Reference Guide* and *Unified Correlation Analyzer for Event Based Correlation Value Pack Development Guide*.

UCA for EBC Topology Extension installation

The UCA for EBC Topology Extension product is delivered as a tar file named:

uca-ebc-topo-kit-3.0-<os>.tar

where *<os>* is *linux* on **Linux** and *hpux* on **HP-UX** systems.

This chapter describes the software prerequisites, the installation steps, and gives a brief content description of the UCA for EBC Topology kit.

2.1 Licensing

After you first start UCA for EBC Server, UCA for EBC will activate a trial license for 90 days (Instant-On license) that enables all features of the product (including the 'UCA for EBC Topology Extension' feature) for a trial period. After expiration of this trial period, a commercial license is needed to continue using the product.

A valid license key is required to activate the 'UCA for EBC Topology Extension' feature after the trial period.

In the case you are using the Topology extension (the 'uca.ebc.topology' property set to 'embedded', or 'external' in the *uca-ebc.properties* file). If you have no valid license key for the 'UCA for EBC Topology Extension' feature after the trial period, UCA for EBC Server will not start.

Notes

As soon as a permanent license is installed in the *license.txt* file, the UCA for EBC Instant-On feature (that lets you use all features of UCA for EBC for a 90 day period so that you can evaluate the product) will be disabled.

In this case, you must add all the permanent licenses corresponding to the UCA for EBC features that you need to the *license.txt* file. For example, if you need the UCA for EBC Topology Extension feature, you need to add a UCA for EBC Topology Extension permanent license to the *license.txt* file.

Please refer to [R4] *Unified Correlation Analyzer for Event Based Correlation Installation Guide* chapter 2.1 "Licensing" for more information on how to retrieve and install UCA for EBC license keys.

2.2 Disk requirements

Here are the disk requirements for UCA for EBC Topology Extension kit:

On HP-UX:

Type	Disk requirements
Temporary disk space used during installation	75 MB minimum: <ul style="list-style-type: none">30 MB minimum for the uca-ebc-topo-kit-3.0-hpux.tar file45 MB minimum for the install-uca-ebc-topology.sh and UCAEBCTOPV300.depot files (expanded from the uca-ebc-topo-kit-3.0-hpux.tar file)
Permanent disk space	40 MB minimum for UCA for EBC Topology Extension V3.0 installed on the system

Table 2 - Disk Requirements for UCA for EBC Topology Extension on HP-UX

On Linux:

Type	Disk requirements
Temporary disk space used during installation	75 MB minimum: <ul style="list-style-type: none">30 MB minimum for the uca-ebc-topo-kit-3.0-linux.tar file45 MB minimum for the install-uca-ebc-topology.sh and UCA-EBCTOPO-V3.0.noarch.rpm files (expanded from the uca-ebc-topo-kit-3.0-linux.tar file)
Permanent disk space	40 MB minimum for UCA for EBC Topology Extension V3.0 installed on the system

Table 3 - Disk Requirements for UCA for EBC Topology Extension on Linux

2.3 Software prerequisites

2.3.1 UCA for EBC Server

The UCA for EBC Topology Extension package must be installed on top of the UCA for EBC Server product.

Software	Package Reference
UCA for Event Based Correlation Server Version V3.0	UCA-EBC.UCAEBCSERVERV300

Table 4 - Software Prerequisites for UCA for EBC Topology Extension (HP-UX)

Software	Package Reference
UCA for Event Based Correlation Server Version V3.0	UCA-EBCSERVER-V3.0

Table 5 - Software Prerequisites for UCA for EBC Topology Extension (Linux)

On HP-UX:

To check if you already have the UCA for EBC Server product installed:

```
$ swlist UCA-EBC
```

You should get an output similar to the following:

```
# UCA-EBC V300 UCA-EBC
UCA-EBC.UCAEBCSERVERV300 V300 HP UCA EBC Server
Version V3.0 Level 01 Rev F
```

On Linux:

To check if you already have UCA for EBC Server product installed (as root):

```
$ rpm -qa | grep UCA-EBC
```

You should get an output similar to the following:

```
UCA-EBCSERVER-V3.0
```

2.3.2 Java

The UCA for EBC Topology Extension (based on Neo4J 1.9 Graph Database) needs Oracle(R) Java(TM) Runtime Environment 7.

Red Hat Enterprise Linux Server comes with OpenJDK Java VM, and unfortunately this JVM is not supported by Neo4J 1.9 Graph Database.

So it is recommended that you download (for free) the latest Java packages (HotSpot Java VM) from Oracle from <http://java.com/en/download/manual.jsp>. If this is installed (usually under /usr/java), you should get an output similar to the followings:

```
jdk-1.6.0_23-fcs.x86_64
jre-1.7.0_21-fcs.x86_64
```

Then you will have to set the \$JAVA_HOME variable of user *uca* accordingly.

2.4 UCA for EBC Topology Extension package installation

2.4.1 Untar the archive in a temporary directory

As **root** user, untar the archive in a temporary local directory (For example: /tmp):

On HP-UX:

```
$ cd /tmp
$ tar -xvf <kit location>/uca-ebc-topo-kit-3.0-hpux.tar
```

On Linux:

```
$ cd /tmp
$ tar -xvf <kit location>/uca-ebc-topo-kit-3.0-linux.tar
```

2.4.2 Run the installation script

Still as **root** user, run the package installation script:

On both HP-UX and Linux:

```
$ install-uca-ebc-topology.sh [ -r <UCA-EBC root directory> ]
```

The *<UCA-EBC root Directory>* parameter value represents the absolute path of the Root directory of the UCA for EBC Server product. It corresponds to the value of the `#{UCA_EBC_HOME}` environment variable.

Note

The default value for the UCA for EBC Server root directory is `/opt/UCA-EBC`.

However, since UCA for EBC Server may have been installed at an alternate location, please check with your system administrator to obtain the correct installation directory.

2.4.3 File organization

UCA for EBC Topology Extension files are installed under the *<UCA-EBC root Directory>* directory (the default is `/opt/UCA-EBC`).

The UCA for EBC Topology Extension package installation augments the UCA for EBC package directories as described in the table below.

Subdirectories	Description
<i>bin</i>	Augmented with the <code>uninstall-uca-ebc-topology</code> script.
<i>lib</i>	Augmented with all libraries required for the implementation of the topology feature

Table 6 - UCA for EBC Topology Extension provided files

2.5 Un-installation of UCA for EBC Topology Extension

The UCA for EBC Topology Extension product can be easily uninstalled by running the **`uninstall-uca-ebc-topology`** script provided in the `#{UCA_EBC_HOME}/bin` directory.

This uninstall utility must be run with **root** privileges.

When the **`uninstall-uca-ebc-topology`** tool is launched, it checks for all UCA for EBC Topology native packages installed on your system and prompts you for the number associated with the package to be uninstalled:

On both HP-UX and Linux:

```
$ /opt/UCA-EBC/bin/uninstall-uca-ebc-topology
```

You should get an output similar to the following text:

Here is the list of installed UCA-EBC TOPOLOGY packages:

```
[0]      UCA-EBCTOPO-V3.0
```

Enter the index number of UCA-EBC TOPOLOGY version to un-install:

By entering '0' (as in the example above), UCA for EBC Topology version V3.0 will be removed.

UCA for EBC Topology Extension Configuration

3.1 Configuring the topology extension

After installation of the UCA for EBC Topology extension package, the topology features are not enabled by default.

To be able to use the topology features, the first requirement is to start a topology server. This can be done in two ways:

- Start an embedded topology Server
- Use an external topology Server

3.1.1 Starting an Embedded Topology server

This is the easiest way to start the topology server because in this case the topology server is started in the same process as the UCA for EBC server without any additional commands.

This is accomplished by setting the following property in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file:

```
uca.ebc.topology=embedded
```

When the topology server starts for the first time (or in case the database repository is not found) it creates the database repository. The default database repository location is:

```
${UCA_EBC_INSTANCE}/neo4j
```

This location can be changed by setting the `uca.ebc.topology.location` property to point to another directory. However, it is not recommended to change this default location (the backup tool will restore neo4j DB under `${UCA_EBC_INSTANCE}/neo4j`)

For example:

```
uca.ebc.topology.location=/data/uca-ebc-database
```

The topology server implements a web server in order to serve REST http requests or GUI http requests. This web server is started on the default port number: 7474. If this value conflicts with a port used by another application then you can change the port number by setting the `uca.ebc.topology.webPort` property.

For example:

```
uca.ebc.topology.webPort=7475
```

The backup tool delivered with UCA for EBC is using the neo4j online backup facility that allows backup of a DB while the server is running without requiring a lock. To configure this facility, it is compulsory to set the `neo4j.config.online_backup_enabled` property as following:

```
neo4j.config.online_backup_enabled=true
```

The default port used for the online backup is also configurable. The default instance uses port 6362 if it is not configured.

If you are running multiple instances of UCA for EBC with neo4j topology server, it is mandatory to change this port number by setting the `neo4j.config.online_backup_port` property.

For example:

```
neo4j.config.online_backup_port=6462
```

With UCA for EBC V2.0 the two above properties `online_backup_[enabled|port]` are not defined by default. **Make sure you have them well defined in uca-ebc.properties file if you intend to use the backup utility.**

3.1.1.1 Additional Neo4J Server configuration

The embedded Neo4J Server may need additional configurations by the use of some specific neo4j properties (cache size, mapped memory size, etc..).

In such case these neo4j properties must be set in the `uca-ebc.properties` file with the prefix “neo4j.config.”

Example:

To set the `neostore.nodestore.db.mapped_memory` neo4j property to a value of 25M, the following must be defined in the `uca-ebc.properties` file :

```
neo4j.config.neostore.nodestore.db.mapped_memory=25M
```

3.1.2 Using an external Topology server

The use of an external topology server can be useful in some specific conditions (High Availability for example).

When using an external server we assume this server is started and monitored externally.

The UCA for EBC Topology Extension is designed to work with Neo4J 1.9 Graph Database as the topology server.

For the external topology server configuration, the installation and configuration of this product is a prerequisite.

This topology server can run on the same host as the UCA for EBC server or on a remote host.

Configuring UCA for EBC to use an external topology server requires the following properties to be defined in the `uca-ebc.properties` file:

- The `uca.ebc.topology` property is set to ‘external’
- The `uca.ebc.topology.serverhost` is set to the hostname of the system hosting the external topology server.

- The `uca.ebc.topology.webPort` is set to the web port used by the external topology server

Configuration example:

```
uca.ebc.topology=external
uca.ebc.topology.serverhost=poolv1.gre.hp.com
uca.ebc.topology.webPort=7474
```

Additional optional properties may be defined in the `uca-ebc.properties` file:

- The `uca.ebc.topology.https.enabled` property is by default set to false. It configures the HTTP channel of the REST API of the neo4j server to be secure if it is set to true. Please note that the certificates should be previously well configured on client/server sides.

```
uca.ebc.topology.https.enabled=true
```

3.2 Configuring the topology data load function

The topology data-load function is performed by a specific topology plug-in that runs on the server.

The role of this plugin is to parse and load the topology information contained in CSV files into the Neo4j graph database. This is fully detailed in a subsequent chapter, “Topology Data load”, further in this document.

With an embedded server configuration, there is no additional configuration to be performed.

With an external server configuration, the data-load plugin, which is installed as part of the “UCA for EBC topology Extension”, must be manually copied to the ‘plugins’ directory of the external topology server.

The following files should be copied to the NEO4j topology server ‘plugins’ directory:

- `${UCA_EBC_HOME}/lib/opencsv-2.3.jar`
- `${UCA_EBC_HOME}/lib/scalalogging-slf4j_2.10-1.0.1.jar`
- `${UCA_EBC_HOME}/lib/uca-ebc-topology-dataload-3.0.jar`
- `${UCA_EBC_HOME}/lib/config-0.5.2.jar`

3.3 Configuring the topology dataload scheduler

The topology data-loading is performed by pushing dataload CSV files to the ‘import/csv’ directory of the Neo4J server.

For embedded topology server this directory is located at:

```
${UCA_EBC_INSTANCE}/neo4j/import/csv.
```

Once the CSV files are pushed to the correct location, the dataload operation must be triggered.

This can be done through the UCA for EBC GUI in the following panel:

UCA for EBC > Topology Management > DataLoad

Clicking the “Topology Dataload” button will trigger the dataload.

The Dataload can also be triggered, by the Dataload Scheduler, on a (configurable) regular basis.

The Dataload Scheduler works regardless of the topology server configuration (embedded or external).

Two parameters can be configured for the Topology Dataload Scheduler:

- The start time

The start time indicates the time for the first dataload.

- The period

The period indicates the time (in hours) between subsequent data loads

Example1:

Start time = 03:00:00

Period = 6

Data load will occurs at: 03:00:00, 09:00:00, 15:00:00, 21:00:00

Example2:

Start time = 06:00:00

Period = 8

Data load will occurs at: 06:00:00, 14:00:00, 22:00:00

Example3:

Start time = 06:00:00

Period = 5

Data load will occurs at: 06:00:00, 11:00:00, 16:00:00, 21:00:00, 02:00:00

The Topology Dataload Scheduler is configured by setting properties in the *uca-ebc.properties* file.

Enabling the Topology Dataload Scheduler:

This is done by setting the `uca.ebc.topology.dataload.usescheduler` property to 'true'

For example:

```
uca.ebc.topology.dataload.usescheduler=true
```

Setting the data load scheduler start time:

This is done by setting the `uca.ebc.topology.dataload.starttime` property to the start time of the data-load in HH:MM:SS format.

The Default value if this property is not specified is: 03:00:00.

Example:

```
uca.ebc.topology.dataload.starttime=04:00:00
```

This will schedule a topology dataload at 4:00 AM.

Setting the data load scheduling period

This is done by setting the `uca.ebc.topology.dataload.period` property to the number of hours between subsequent data loads.

The default value if this property is not specified is: 4.

Example:

```
uca.ebc.topology.dataload.starttime=04:00:00  
uca.ebc.topology.dataload.period=4
```

This will schedule a topology dataload at 4 AM and then every 4 hours.

NOTE: These properties are taken in to account at application start-up. Any changes will require restarting the UCA-EBC server.

Managing the Topology from the UCA for EBC GUI

The UCA for EBC Web interface provides a new menu when the topology is enabled.

This new “Topology Management” menu can be reached from the main menu under the “UCA for EBC” entry:

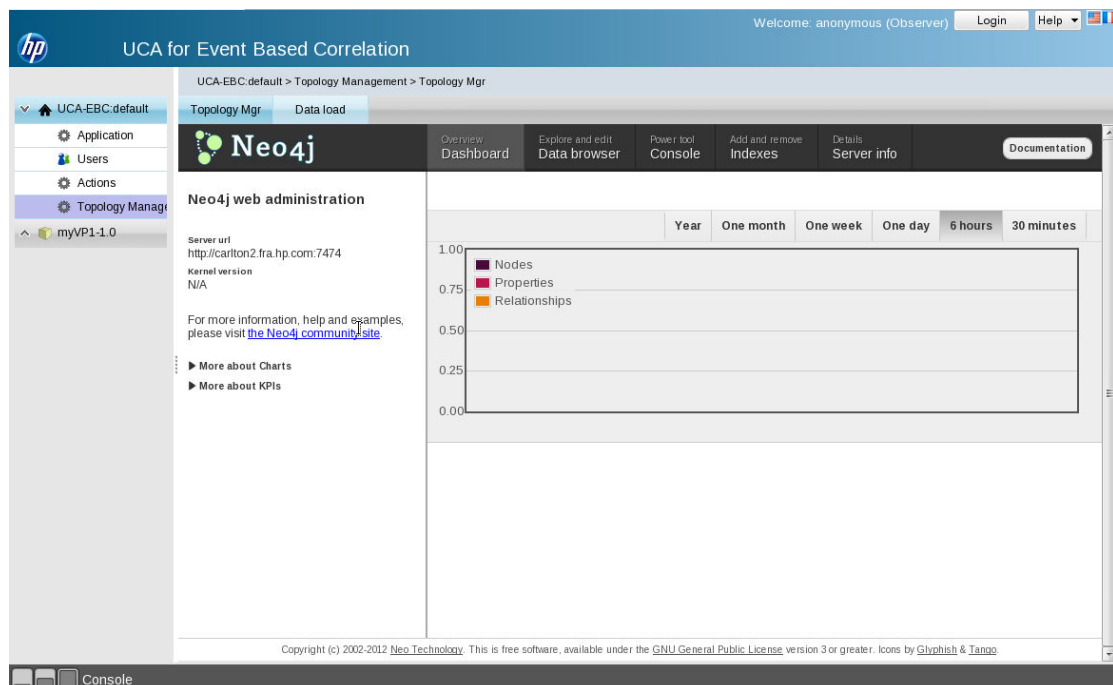
UCA for EBC > Topology Management

This menu gives access to the Topology Management panel that offers two sections in the horizontal menu:



4.1 The Topology Manager Panel

The Topology Manager Panel displays the genuine Neo4j web manager. It appears as shown below:



The Neo4j web manager has several entries. The default one is a dashboard indicating the number of graph database objects and a graph showing the evolution of this number over time.

UCA for EBC - UCA for EBC

localhost:8888/uca_expert_u.html#EXPERT.TOPOLOGY.TOPOLOGYMGR

Extending Eclipse - ... mbaron.hp-develop... EclipseZone - Gettin... Flexible Project Stru... Help - Eclipse Platfo... Jersey - Classloader L... dei pointers

hp UCA for Event Based Correlation

User: admin Role: Administrator, Operator, Developer

UCA for EBC > Topology Management > Topology Mgr

Topology Mgr Data load

Overview Dashboard Editor and view Data browser Project tool Console Details Server info Monitoring resources Index manager Documentation

Neo4j

0

+ Node + Relationship

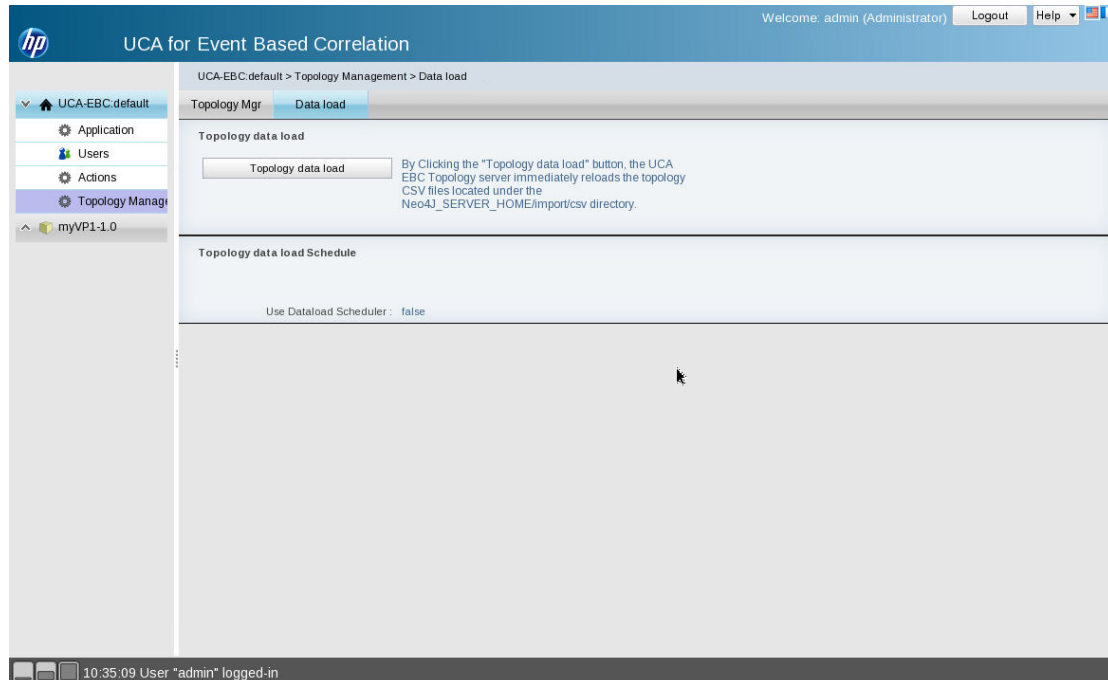
Style Re-layout Clear

Copyright © 2002-2012 Neo Technology. This is free software, available under the [GNU General Public License](#) version 3 or greater.

21

4.2 The Data load Panel

This panel is specific to the UCA for EBC Topology extension. From this panel, it is possible to trigger a data-load on the Neo4j server thanks to the “Topology data load” button. Note that this button is enabled only for user having a ‘developer’ or ‘administrator’ roles.



On “Topology Data load” button click, the server reloads all CSV files present in the import/csv directory as detailed in the “Topology Data load” Chapter in this document.

Developing a topology aware Value Pack

5.1 Introduction

Topology aware Value Packs, like “regular” Value Packs, are developed using the UCA for EBC Development Toolkit under the Eclipse development environment.

Please refer to the *Unified Correlation Analyzer for Event Based Correlation Value Pack Development Guide* Document if you are not familiar with the UCA for EBC Value Pack development environment.

The UCA for EBC Development toolkit V3.0 has been enhanced to provide:

- The libraries required for retrieving topology information.
- The topology database framework used by the Junit tests in order to validate topology aware Value Packs.
- The necessary tools to perform basic dataloading.
- An example of a topology Value Pack (named topology-example)

5.2 Topology aware Value Pack principles

Topology aware Value Packs are mainly based on a new feature provided by the UCA for EBC V3.0 product called “Alarm Enrichment”.

Alarm enrichment consists in adding extra data to Alarms (possibly also adding extra attributes to Alarms). For topology aware Value Packs, this extra information usually comes from the topology database and takes the form of extra topology-related Alarm attributes.

These new topology-related Alarm attributes can then be used in Drools Rules just like any other standard Alarm attributes.

The following picture shows the alarm enrichment process in UCA for EBC, for topology aware Value Packs:

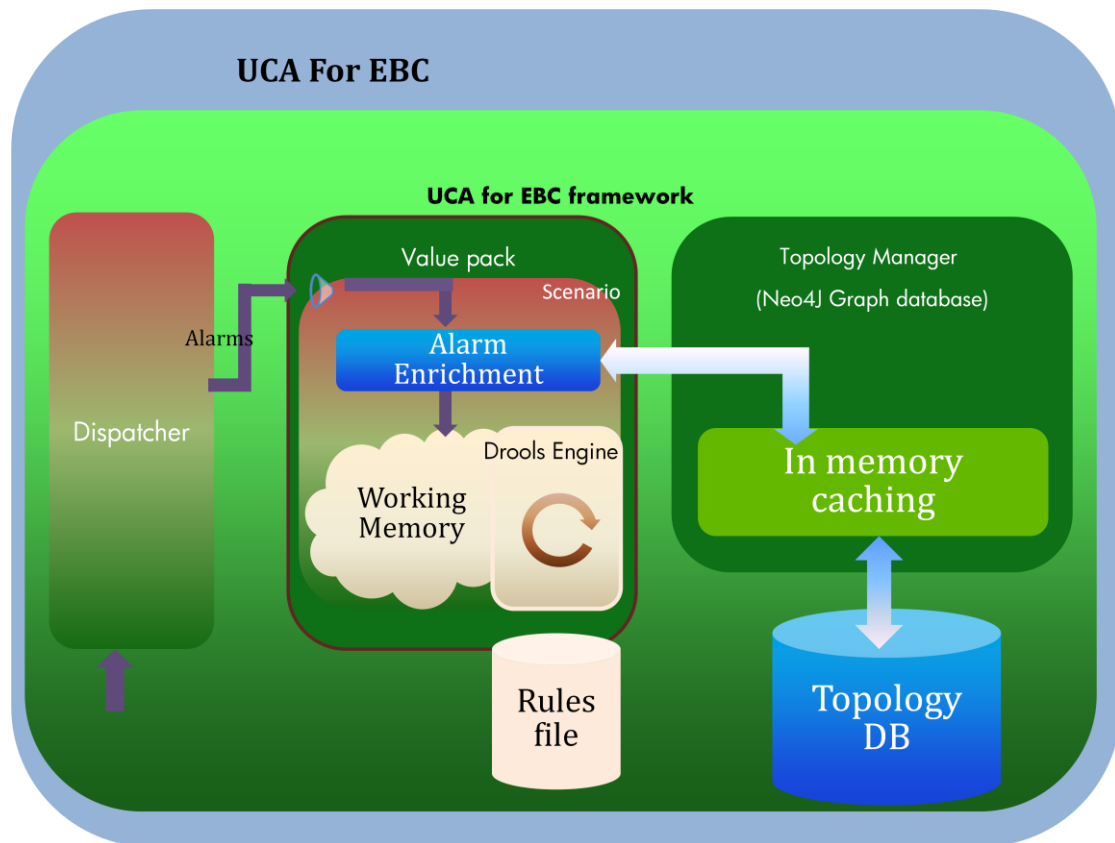


Figure 1 - Alarm enrichment with topology information

5.3 Alarm enrichment with topology information

Alarm enrichment is performed in several steps:

1. Provide one or more new Alarm Object classes that extend the Alarm Object (or one of the AlarmDeletion, AlarmStateChange or AlarmAttributeChange objects), which can hold extra topology-related attributes
2. Implement the topology requests that will set the value of the extra topology-related Alarm attributes.
3. Configure the Value Pack framework to take into account the Alarm Enrichment process

5.3.1 Alarm Object extension

An extended Alarm must extend one of the Alarm classes provided by the UCA for EBC Framework. The Alarm classes are:

```
com.hp.uca.expert.alarm.Alarm
com.hp.uca.expert.alarm.AlarmDeletion
com.hp.uca.expert.alarm.AlarmStateChange
com.hp.uca.expert.alarm.AlarmAttributeValueChange
```

The extended alarm must implement setters and getters for the new attributes (this will ease the process of writing rules).

Below is an example of such an extended Alarm, as provided in the topology-example Value Pack:


```

package com.hp.uca.ebc.topoexample;

import javax.xml.bind.annotation.XmlRootElement;
import org.neo4j.graphdb.Relationship;
import com.hp.uca.expert.alarm.Alarm;
import com.hp.uca.expert.alarm.AlarmHelper;

@XmlRootElement
public class LinkdownEnrichedAlarm extends Alarm {

    private static final String F_TOPONODE = "associated Node ";
    private static final String F_LINKNAME = "link Name      ";

    /**
     *
     */
    private static final long serialVersionUID = -5235137849600943223L;

    /**
     * Alarm extension attributes
     */

    /**
     * Stores the Unique Identifier of the topology Node (Port) this alarm
     * is mapped to.
     */
    private String associatedNodeUniqueId;
    /**
     * pointer to the 'LINK' relation the {@link associatedNodeUniqueId} is
     * associated to
     */
    private Relationship link;

    /**
     * LinkDownEnrichedAlarm Constructor
     */
    public LinkdownEnrichedAlarm() {
        super();
    }

    /**
     * LinkDownEnrichedAlarm Copy Constructor.
     *
     * This constructor constructs an Extended Alarm and uses the Alarm provided as
     * parameter to set all the standard alarm fields
     */
    public LinkdownEnrichedAlarm(Alarm alarm) {
        super(alarm);

        // initialize the associated Node.
        // use the instance mapping facility
        // check the linkdown/mappers.xml file to see how the mapping is done
        associatedNodeUniqueId = alarm.getMappings().get("NetworkInstance");

        // set the link information if any
        link = LinkdownTopoAccess.getPortLink(associatedNodeUniqueId);
    }

    /**
     * Clones the provided Extended Alarm object
     * @return a copy of the current object
     * @throws CloneNotSupportedException

```

```

    */
    @Override
    public LinkdownEnrichedAlarm clone() throws CloneNotSupportedException {
        LinkdownEnrichedAlarm newAlarm = (LinkdownEnrichedAlarm) super.clone();
        newAlarm.link = this.link;
        newAlarm.associatedNodeUniqueId = this.associatedNodeUniqueId;
        return newAlarm;
    }

    /**
     * returns the {@link link} attribute
     * @return the {@link link} attribute
     */
    public Relationship getLink() {
        return link;
    }

    /**
     * sets the {@link link} attribute
     */
    public void setlink(Relationship link) {
        this.link = link;
    }

    /**
     * returns the {@link link} name attribute
     * @return the {@link link} name attribute
     */
    public String getLinkName() {

        return LinkdownTopoAccess.getLinkName(link);
    }

    /**
     * returns the {@link link} state attribute
     * @return the {@link link} state attribute
     */
    public String getLinkState() {

        return LinkdownTopoAccess.getLinkState(link);
    }

    /**
     * sets the {@link link} state attribute
     */
    public void setLinkState(String state) {
        LinkdownTopoAccess.setlinkState(link, state);
    }

    /**
     * returns the {@link associatedNodeUniqueId} attribute
     * @return the {@link associatedNodeUniqueId} attribute
     */
    public String getAssociatedNodeUniqueId() {
        return associatedNodeUniqueId;
    }

    /**
     * sets the {@link associatedNodeUniqueId} attribute
     */
    public void setAssociatedNodeUniqueId(String associatedNodeUniqueId) {
        this.associatedNodeUniqueId = associatedNodeUniqueId;
    }
}

```

```

/*
 * (non-Javadoc)
 *
 * @see com.hp.uca.expert.alarm.AlarmCommon#toFormattedString()
 */
@Override
public String toFormattedString() {
    StringBuffer toStringBuffer= AlarmHelper.toFormattedStringBuffer(this);

    AlarmHelper.addFormatedItem(toStringBuffer, F_LINKNAME, getLinkName());
    AlarmHelper.addFormatedItem(toStringBuffer, F_TOPONODE,
                                getAssociatedNodeUniqueId());

    return toStringBuffer.toString();
}
}

```

5.3.2 Activating Alarm Enrichment

The Alarm Enrichment process is implemented by a Java Class that extends the `com.hp.uca.expert.lifecycle.LifeCycleAnalysis` class provided by the UCA for EBC framework.

Activating Alarm Enrichment is fully detailed in the “*Unified Correlation Analyzer for Event Based Correlation Reference Guide*” Chapter 3.4.5 “Alarm Enrichment”.

Below is an example of how to activate the Alarm Enrichment process, as provided in the topology-example Value Pack:

```

package com.hp.uca.ebc.topoexample;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.hp.uca.common.trace.LogHelper;
import com.hp.uca.expert.alarm.Alarm;
import com.hp.uca.expert.alarm.AlarmCommon;
import com.hp.uca.expert.lifecycle.LifeCycleAnalysis;
import com.hp.uca.expert.scenario.Scenario;

// Just an example of extension of analyzer
public class LinkdownLifeCycle extends LifeCycleAnalysis {

    private static Log log = LogFactory.getLog(LinkdownLifeCycle.class);

    public LinkdownLifeCycle(Scenario scenario) {
        super(scenario);
    }

    @Override
    public AlarmCommon onAlarmCreationProcess(Alarm alarm) {
        LogHelper.enter(log, "onAlarmCreationProcess()");

        LinkdownEnrichedAlarm enrichedAlarm = new LinkdownEnrichedAlarm(alarm);

        LogHelper.exit(log, "onAlarmCreationProcess()");
        return enrichedAlarm;
    }
}

```

The use of this new Lifecycle class must be indicated in the ValuePackConfiguration.xml file, in the corresponding scenario section. This is done by setting the <customLifecycleClass> XML element as follows:

```
<customLifecycleClass>
    com.hp.uca.ebc.topoexample.LinkdownLifecycle
</customLifecycleClass>
```

5.3.3 Topology requests implementation

The UCA for EBC Topology extension provides a static Java class, com.hp.uca.expert.topology.TopoAccess, that gives access to the Neo4J database through the getGraphDB() method.

The getGraphDB() method of the TopoAccess class returns a Neo4J `GraphDatabaseService` instance which is the starting point for any Topology request.

For a full description of the services provided by the GraphDatabaseService Class, please refer to the Neo4J documentation at the following URL: <http://docs.neo4j.org/>

Here is an example of how to retrieve a topology node from an index and getting the associated 'LINK' relation:

```
public static String PORT_UNIQUEID = "uniqueId";
public static String PORT_TAG = "Tag";
public static String PORT_INDEX = "PortsByUniqueId";

/**
 * This method retrieves the link associated to this Port (if any defined)
 * @param portName
 * @return
 */
public static Relationship getPortLink(String portName) {
    Relationship link=null;

    IndexManager indexMgr = TopoAccess.getGraphDB().index();
    Index<Node> nodeIndex = indexMgr.forNodes(PORT_INDEX);

    IndexHits<Node> hits = nodeIndex.get( PORT_UNIQUEID, portName);
    if (hits.size() != 0) {

        Node port=hits.getSingle(); // assume a single port with this name

        Iterable<Relationship> relations=port.getRelationships(PortRelationshipType.LINK);
        for (Relationship relation : relations) {
            link = relation;
            break; // assume there is a single link per port.
        }
    }

    return link;
}
```

Neo4J offers many ways to retrieve topology information. You should especially focus on:

- The “Neo4J graph database” at URL: <http://docs.neo4j.org/chunked/stable/graphdb-neo4j.html>

- The “Neo4J Cypher Query Language” at URL:
<http://docs.neo4j.org/chunked/stable/cypher-query-lang.html>
- The “Neo4J Execute Cypher queries from Java” at URL:
<http://docs.neo4j.org/chunked/stable/tutorials-cypher-java.html>

5.3.4 Writing rules based on topology data

In most cases, the topology data is retrieved during the Alarm Enrichment phase. This means that once the Drools rules are executed there is no need for additional topology access, thus improving the performance of the Drools rules engine.

Rules are evaluated on alarm fields (or Alarm methods) regardless of whether these fields (or methods) are returning values coming from topology enrichment or values coming from standard alarm fields.

Here is an example of a rule based on topology information:

```
//
// This rule detects when there is an alarm on each port of a connection link.
//
// In such case it sets the Link state to 'Down'.
//
rule "Set Link down"
no-loop
  when
    $alarm1 : LinkdownEnrichedAlarm(
      wmStatus == WMOBJECTSTATUS.INJECTED ,
      linkState matches LinkdownTopoAccess.LINK_STATE_UP)
    $alarm2 : LinkdownEnrichedAlarm(
      wmStatus == WMOBJECTSTATUS.INJECTED ,
      identifier != $alarm1.identifier ,
      linkName == $alarm1.linkName )
  then
    LogHelper.enter(theScenario.getLogger(), drools.getRule().getName());

    theScenario.getLogger().info("The link : "+$alarm2.getLinkName()+
                                " was detected down !");

    $alarm1.setLinkState(LinkdownTopoAccess.LINK_STATE_DOWN);

    theScenario.getSession().update($alarm1);
    theScenario.getSession().update($alarm2);

    LogHelper.exit(theScenario.getLogger(), drools.getRule().getName());
  end
```

In this example, the 'linkName' extended alarm attribute is updated during the Enrichment phase, based on the Originating Entity of the Alarm and extra topology information retrieved from the topology database.

The rule simply says that if there are two alarms on port nodes connected by the same 'link' (linkname == \$alarm1.linkName), then it implies that the link has failed.

5.4 Topology Value Pack example description

The UCA for EBC Development toolkit V3.0 comes with a new example Value Pack called “topology-example” located under the %UCA_EBC_DEV_HOME% directory, i.e. the C:\UCA-EBC-DEV\ directory by default.

This example is a good starting point for developing new topology aware Value Packs.

This is also the same example that is created when the UCA for EBC Eclipse plug-in is used to create a new topology aware Value Pack project.

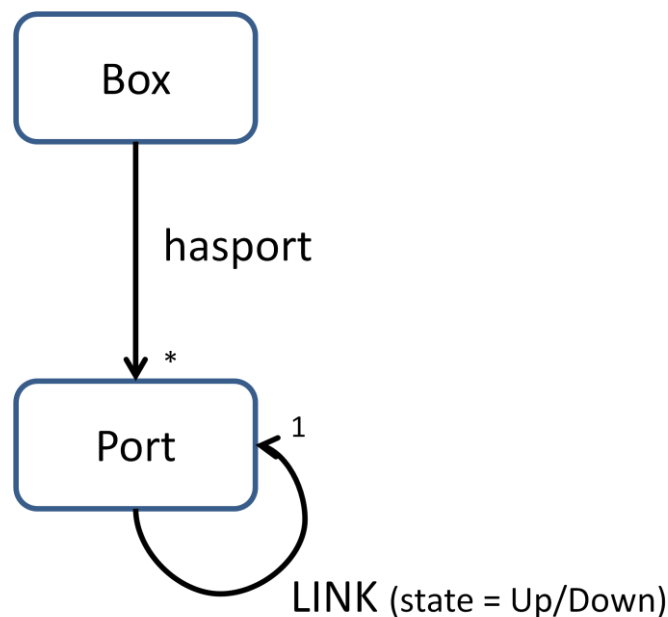
Note

☞ Please refer to Chapter 6 “Creating a new topology aware Value Pack using the UCA eclipse plug-in” for full details on the UCA for EBC Eclipse plug-in usage.

5.4.1 Objectives

The objective of this Value Pack example is to demonstrate a “link down” detection use case.

For this use case the topology model is the following:



In this model, a ‘Box’ node can have several Ports and a Port can be linked to another Port.

When an alarm is received, the `OriginatingManagedEntity` alarm attribute is used to identify the corresponding topology object (Port).

The Alarm is enriched with a field containing a reference to the LINK relation between the Port this alarm corresponds to and the other Port it is connected to. Then the alarm is inserted into the working memory.

A simple rule detects if two alarms have the same LINK reference with an ‘Up’ LINK state (meaning that the two ends of the LINK have just failed because there are alarms on each Port at both ends of the LINK, but the LINK state is wrongly set to ‘Up’). In such a case, the action part of the rule is executed and state of the LINK is set to ‘Down’.

In the ‘topology-example’ Value Pack, in addition to the LINK ‘Down’ rule that is described above, a similar rule deals with the LINK ‘Up’ use case.

An additional rule is executed when the Value Pack is re-synchronized in order to set all the LINK states to 'Up' at initialization time.

5.4.2 Topology aware Value Pack validation

The behavior of a topology aware Value Pack (just like for any “regular” Value Pack) can be validated by JUnit tests.

Some JUnit tests are provided for the topology-example Value Pack.

The requirement for the topology aware Value Pack tests is that the testing environment provided by the UCA for EBC Development toolkit must implement a topology database server ready to be used when the tests are executed.

This is done by setting the following properties in the `uca-ebc.properties` file located in the Value Pack `src/test/resources` directory:

```
uca.ebc.topology=embedded
uca.ebc.topology.location=target/neo4jDB
uca.ebc.topology.webPort=7475
```

The `uca.ebc.topology` property indicates that a topology server must be embedded during the test.

The `uca.ebc.topology.location` indicates where the database should be created (the target directory is a temporary data directory when building or testing a Value Pack).

The `uca.ebc.topology.webPort` indicates the topology server port. This port should not conflict with the default Neo4j port: 7474, in case a Neo4j server using the default port number is running on the same system.

The tests themselves must prepare the topology environment before beginning to process incoming alarms. Especially the topology objects used by the tests must be present in the Database before the first test is executed.

This is usually done in a `@Before` section of the JUnit test. The utility class ‘Loader’ is used to load object from CSV files. In the example below these CSV files are located in the `valuepack/linkdown/topologyDataload` directory of the valuepack.

Note that the CSV files are first copied on a temporary directory because the loader does modify it.

Refer to the “Topology Data load” Chapter of this manual for a full description of the CSV dataload principle.

Below is an example from the topology-example Value Pack:

```
private static TmpDir tmpDir = null;

@BeforeClass
public static void init() {
    tmpDir = new TmpDir("valuepack/linkdown/topologyDataload");
}

@AfterClass
public static void cleanup() {
    tmpDir.cleanup();
}

/**
 * @throws java.lang.Exception
```

```

    */
    @Before
    public void setUp() throws Exception {
        log.info(Constants.TEST_START.val() + linkdownTest.class.getName());

        Loader loader = new Loader(LinkdownTopoAccess.getGraphDB(),
                                   tmpDir.tmpCsvPath());
        Report report = loader.loadAll();

        log.info(report.toString());
    }

```

Please refer to the source code of the topology-example Value Pack for the full implementation of the topology-example Value Pack JUnit tests.

5.4.3 Building and Deploying the topology-example Value Pack

In terms of build and deployment, a topology aware Value Pack is very similar to any other UCA for EBC Value Pack.

The Value Pack is built and unit tested using Apache Ant, based on the build.xml file provided with the Value Pack.

The built package can then be deployed on a UCA for EBC server by copying the Value Pack ZIP file generated by the build procedure to the 'valuepacks' directory of the UCA for EBC server.

From the UCA for EBC GUI, the Value Pack can then be deployed and started as usual.

Before sending alarms to the Value Pack, the topology data must be loaded in the topology database. This is done using the topology data load CSV files provided with the Value Pack under the `${UCA_EBC_INSTANCE}/deploy/topology-example-1.1/linkdown/topologyDataLoad` directory.

The full set of files must be copied to the `${UCA_EBC_INSTANCE}/neo4j/import/csv` directory on the UCA for EBC server and a data load request must be performed from the GUI panel:

UCA for EBC > Topology Management > DataLoad

Clicking on the "Topology Dataload" button will trigger the data load.

When this is done the alarm samples provided with the topology-example Value Pack can be sent using the UCA for EBC alarm injector command-line tool:

`${UCA_EBC_HOME}/bin/uca-ebc-injector.`

The two alarm samples are as follows:

- `${UCA_EBC_INSTANCE}/deploy/topology-example-1.1/Alarms_linkdown.xml`
- `${UCA_EBC_INSTANCE}/deploy/topology-example-1.1/Alarms_linkup.xml`

You can send the sample alarm files to UCA for EBC by executing the following commands:

```
[uca]$ cd ${UCA EBC INSTANCE}/deploy/topology-example-1.1  
[uca]$ ${UCA EBC HOME}/bin/uca-ebc-injector -file  
Alarms_linkdown.xml
```

```
[uca]$ ${UCA EBC HOME}/bin/uca-ebc-injector -file  
Alarms_linkup.xml
```

Creating a new topology aware Value Pack using the UCA eclipse plug-in

6.1 Topology aware Value Pack project creation

The UCA for EBC Value Pack development kit comes with an Eclipse plugin that helps creating UCA for EBC eclipse projects.

The “UCA for EBC Value Pack development Guide” fully describes how to install the Eclipse plugin and create a new UCA for EBC project with the help of this plugin.

This plugin can be used to create topology aware Value Packs.

This is done by selecting the “topology aware Scenario” template from the project creation wizard as shown in the following picture:

Create a UCA EBC Valuepack Project
Create a UCA EBC Scenario

Scenario name: myTopologyAwareScenario

Scenario package: (e.g. com.example.myvaluepack) com.hp.myvaluepack

filters
filter file name: filters.xml

Rule file
Rule file name: (e.g. myfile.drl) rules.drl
Rules Description: myTopologyAwareScenario rules
☐ Use templated rules
Rule parameters file name: (e.g. myparameterfile.xml)

Scenario Template
☐ basic Scenario
☒ topology aware Scenario

? < Back Next > Finish Cancel

Figure 2 - Topology aware scenario creation using the Eclipse plugin

Choosing this template, the plugin will create a new project based on the ‘linkdown’ topology aware Value Pack example described earlier in this document.

You will then have to modify some files (topology access java classes, rules, tests) in order to implement your own scenario.

There is no action needed on your part to reference the proper UCA for EBC and UCA for EBC Topology libraries from your new project. Your new project automatically references the necessary libraries from the %UCA_EBC_DEV_HOME%\lib directory.

A specific *uca-ebc.properties* file is present and configured in your Value Pack in order for the JUnit tests to automatically run an embedded instance of the Neo4j* server. The execution of these JUnit tests will allow you to properly unit test your topology aware Value Pack.

Notes

*Neo4j is the name of the Graph Database used in UCA for EBC to store topology information. For more information on Neo4j please see the following URL: <http://neo4j.org>

**In the current version of the UCA for EBC Eclipse plugin, when the Topology aware Scenario Template is chosen, the "use templated rules" option has no effect. The plugin will generate non-templated rules.

If you want to use templated rules, you must manually configure the *ValuepackConfiguration.xml* file of your ValuePack accordingly.

Topology Data load

7.1 Introduction

UCA for EBC Topology Extension comes with a data loading tool that can help you populate your UCA for EBC (Neo4j) topology database. This tool reads CSV (Comma-Separated Values) files as input.

The CSV Neo4j loader can be used to insert, update or delete nodes or relationships into a local Neo4j database. When developing a UCA value pack, this is particularly useful for unit or integration testing, to easily upload sample topology data required in the various correlation use cases.

In production, the preferred UCA/Neo4j loading application is the HP Unified Topology Manger (UTM). In fact, UTM generates CSV files behind the scene and invokes the UCA Neo4j CSV loader automatically. In such situations, the CSV loader is hidden to the UCA administrator or users.

The loader has three different forms:

- A Neo4j server plugin. If the Neo4j server is embedded in UCA, the loader plugin is installed by default.
- A java API, accessible in the JUnit tests of a Value Pack project. The main class of the API is: `org.neo4j.loader.csv.Loader`
- A standalone command line application (made of a unique jar file), useful to test CSV files or visualize test topology data.

7.2 CSV files location

The loader reads input CSV files from a unique directory on disk. The location of this directory is given as a parameter of the data load command. If absent, a default value is used (see below for examples). This input directory is also called the “import” directory.

The procedure to perform a topology data load is the following:

1. Drop some CSV files in the import directory
2. Invoke the data load command (see below for examples), giving the import directory as a parameter (or just rely on the default location if omitted).

These steps can be automated (JUnit, UTM) or manual (command line, REST call).

As soon as the data load command is invoked, the input CSV files are moved automatically in an archive directory inside the import directory. A new archive directory is created at each data load attempt. Its name contains a timestamp to make it unique and to keep a clear history of successive load attempts (for instance as a scheduled operation). This move prevents subsequent imports of the same files.

Typical locations of the import directory on disk are:

- `${UCA_EBC_INSTANCE}/neo4j/import/csv` for embedded database configurations or
- `${NEO4J_HOME}/import/csv` for external database configurations

The Neo4j server process must have **read and write permissions** on this folder. This will be the case by default for embedded Neo4j server configurations. For external configurations, it's better to verify that the permissions on the `${NEO4J_HOME}/import/csv` folder are properly set.

7.3 CSV files names

There are four types of input CSV files, identified by their extension:

- Insert/update Node files, ending with a ".node.csv" extension
- Insert/update Relationship files, ending with a ".rel.csv" extension
- Delete node files, ending with a ".node.delete.csv" extension
- Delete relationship files, ending with a ".rel.delete.csv" extension

The number of files in the import directory is not limited. All *.node.csv, *.rel.csv, *.node.delete.csv, *.rel.delete.csv files present in this directory (excluding sub-directories) are considered by the loader. Files with other extensions are ignored.

As said previously, loaded CSV files are moved to an archive sub-directory (for example: `dataload_2012-06-11_03_29_33/`) to prevent them from being imported again at the next data load request. They keep their original names in the archive directory.

7.4 CSV files format

The first two rows in the CSV files are reserved for the file header. The remainder of the file correspond to nodes or relationships to be inserted, deleted or updated in/from the persistent Neo4j graph database.

Columns in the CSV files are nodes' or relationship's properties values.

Each CSV file must have a special two-line header, providing useful formatting information to the loader through properties' "annotations".

The remaining lines (rows) in the file are properties' values, for nodes (in .nodes.csv files) or relationships (in .rel.csv files). All properties values must comply with the format given in the second line of the header.

Columns are separated by the current CSV separator character, which is traditionally a comma, but which can be customized.

Note that traditionally, standard CSV files have a unique header line. For the Neo4j CSV loader however, the header is always made of two lines. Also, the header contains important information used during the data load process. It is therefore parsed by the loader, and its syntax must be valid for the file to be loaded correctly.

If the header syntax is invalid, the entire file can be discarded. If some lines in the file do not correspond to the header format (for instance some columns are missing) then only these invalid lines are discarded.

The second line of the header corresponds to a standard CSV header, providing the description for the various columns. The main information for each column is the property name. This name can also be annotated to provide extra information to the loader. Thus, a property can be typed as an integer (instead of a string), if annotated as follows:

```
@type(int)MyProperty
```

Likewise, the loaded node or relationship can be indexed according to a special property if the latter is prefixed with the @index annotation:

```
@index(MyIndex)MyProperty
```

The full syntax of the various headers and annotations, which are specific to each file, is described in the following sections.

Important note: in the syntax description we assume that the column separation character is a comma ‘,’.

Square brackets [] indicate optional fields.

Curly brackets {} indicate fields that can be repeated.

In all descriptions, an example is provided, which should be readable and self-explanatory.

Blank or tab characters are allowed between header elements or after /before the annotations’ brackets.

7.4.1 Node files: *.node.csv

A node file must start with a two-line header of the following format:

```
@ref([rel_name]) {[ ,  
[@const(value)][@type(data_type)]property_name]}  
@index(index_name) [@type(data_type)] property_name  
{[ ,[@index(index_name)] [@type(data_type)]  
property_name]}
```

For example, for storing rock bands as nodes in the graph, we could define the following node file header:

```
@ref(band) , @const(band) type  
@index(bands) name, genre, popularity
```

In this example, we store rock bands as nodes in the Neo4j server. Each node has a property name “type” set to a string value “band” (for instance to differentiate these nodes from the others in the graph). The nodes also have three other properties (name, genre and popularity) whose values are given in the respective CSV columns.

7.4.1.1 First line

The `@ref(relation_name)` annotation

This annotation is mandatory. It tells the loader if newly created nodes should be automatically attached to the Neo4j Reference Node. Just leave the brackets empty (i.e. `@ref()`) if you don't want to create any relationship.

If a string is provided instead (i.e. `@ref(band)`), each newly created node will be linked automatically to the Reference Node with a relationship named "band". No additional property can be attached to this relationship.

The `@const(property_value)property_name` annotation

This annotation tells the loader if a constant value property should be created for each node (i.e. each row) defined in the file. This is especially useful if you want to tag nodes with a "class" name for example, according to an inherent data "model" in your graph. It avoids creating an extra column containing the same value repeated for all rows. As the property has a constant value (among rows), it cannot be used to index corresponding nodes or relationships. Therefore, `@const` annotations cannot be mixed with `@index` ones.

Often, a graph stores an object model (as in UML for example) and nodes are instances of some classes in the model, while properties are attributes of these classes.

For example, if you load some "Book" nodes in the graph, you probably want to add a special property to indicate that these nodes are effectively "books". You can do this easily with the `@const` annotation as follows:

```
@ref() , @const (Book) class
```

Mixing annotations

You can have as many `@const` annotations as required, for example:

```
@ref() , @const (Book) class, @const (csv_import) loaded_by, ...
```

7.4.1.2 Second line

The second line in the header is the most important one, as it provides the full description and format for all properties' values to be found in the files. It can be seen as the standard "column" description header found in regular CSV files.

The number of entries (i.e. columns) in this header must match exactly the number of columns in each row of the CSV file. If the number of column entries in each row does not match the header count, the row is discarded, as it cannot be parsed correctly according to the given format.

The format of the second line is a simple comma-separated list of property names, which can be annotated.

Here is the second line grammar:

```
@index(index_name) [@type(data_type)] property_name  
{[, [@index(index_name)] [@type(data_type)]  
property_name]}
```

For example, for describing "book" nodes:

```
@index(title), author, genre, date_of_publication
```

Properties can be annotated, for instance to be automatically indexed or stored in the database with a special data type.

The first property in the header line must be indexed (meaning annotated with @index), with a unique id.

Managing unique indexes is one strong constraint of the CSV loader. This ensures that we can use readable peer names when creating relationships between nodes. It also prevents from creating duplicate entries and later updates using the exact same node file syntax.

The @index(index_name)property_name annotation

An @index(index_name) annotation can be used if you want to index the corresponding node, with this property as the key.

For instance, consider the following header second line:

```
@index(Books)title,author,genre,date of publication,...
```

This will use the "Books" index to hold book nodes based on their "title" property. All nodes to be connected by a future relationship must be indexed. It's through an index look up that relationships ends ("from" or "to" peers) are referenced in relationship files (*.rel.csv).

The @type(data_type)property_name annotation

The `@type(data_type)` annotation lets you associate a data type to a property, so that the property values are stored natively in this data type in the Neo4j store.

If no type is given (i.e. no annotation) the property is stored as a String. It is then equivalent to `@type(String)`.

Here is an example of a typed property:

```
@type(int)number_of_records
```

This indicates that the "number_of_records" property values should be stored as an integer value. Of course the values to be found in the corresponding column in the CSV file should be convertible to an int. For instance it should "12" and not "twelve".

If the loader cannot "cast" the property value into the data type specified in the header, the property value is not set at all, not even in its String format.

The data types currently supported are the following (data type names are not case sensitive):

- int or int[]
- long or long[]
- boolean or boolean[]
- string (this is the default data type if no `@type` annotation is specified) or string[]
- float or float[]
- double or double[]
- char or char[]
- short or short[]

If another un-supported data type is given in the annotation (for example `@type(my_unkwown_type)`), then the "String" data type is used instead.

The square brackets `[]` refer to "arrays". In this case, the value is cast to a Java array of the same type. The provided string value is parsed to retrieve the elements of the array. The default separator character is "!" but this can be customized. For example, here is a valid array definition:

```
@type(int[])days_in_month
31!28!31!30!31!30!31!31!30!31!30!31
```

Mixing annotations

Properties descriptions in the header second line can support several annotations, as long as you respect the following order:

```
@index(my_index) @type(my_data_type) my_property_name
```

For example, if one wants to index nodes with a special "ID" property, which is a Long integer, it can be written as follows:

```
@index(NodesById)@type(long) ID
```

7.4.1.3 Node definitions

Node definitions are the remaining lines in CSV node files. They should follow the format provided in the second line of the header.

For example, a short books.node.csv file can look like this (including the two-line header):

```
@ref(),@const(Book)class,@const(csv_import)loaded_by
@index(Books)title,author,genre
Les Miserables,Victor Hugo,novel
The art of computer programming,Donald Knuth,computing
...
```

7.4.2 Relationship files: *.rel.csv

A relationship file must start with the following two-line header:

```
@rel([rel_name]) {[,
[@const(value)][@type(data_type)]property_name]}
@from(index_name) [@type(data_type)] property_name ,
@to(index_name) [@type(data_type)] property_name
{[,@index(index_name)] [@type(data_type)]
property_name]}
```

For example:

```
@rel(knows)
@from(persons)name, @to(persons)name
```

7.4.2.1 First line

@rel(relationship_name) annotation

This annotation just gives the name of the relationship to be created. Currently, the name of the relationship has to be a String value. Typical relationship names in a social graph can be "knows", "follows", "contains" or whatever. In a telecom network it can be "connected to", "impacting" or else.

If @rel() is left empty, the file name prefix is used as the relationship name. For instance, if the file name is: knows.rel.csv, then the relationship name will be "knows".

@const(value) annotation

This annotation is the same as for node files: Please refer to section 7.4.1.1 "First line" above.

7.4.2.2 Second line

As for node files, the second line provides the format of the CSV columns. It must begin with two @from and @to annotations, which are special forms of the @index

one. They represent both ends of the new relationship to create (or delete or update).

@from(index_name)key_name annotation

The "from" node of the relationship will be retrieved from an index called "index_name", with the property named "key_name".

For instance, if you want to say that "The Beatles" band has "John Lennon" as a member, here's how to do it:

```
@rel(member)
@from(bands)name, @to(musicians)name
The Beatles, John Lennon
```

@to(index_name,key_name) annotation

The @to annotation has the exact same syntax as the @from one.

@type(data_type) annotation

The indexed properties can be typed. For instance, if we use a Long Id as in a previous example, we can use it in the @from or @to annotations as follows:

```
@from(NodesAsId)@type(int)ID, @to(NodesAsId)@type(int)ID
12,4
```

To know the list of supported data types, please refer to the same node file section.

@index(index_name)

Relationship can also be indexed. To do this, use the indexed annotation as for nodes. For instance, if the "member" relationship has uniqueId property that you want to add in the "members" index, here is how you can do it:

```
@rel(member)
@from(bands)name, @to(musicians)name, @index(members) @type(int) uniqueId
The Beatles, John Lennon, 42
```

7.5 Updates

Since all nodes are indexed, and all relationships are uniquely identified (by their name and node ends), the CSV loader is always able to detect automatically if a node or a relationship is already present in the database or not. If a CSV record (node or relationship) is already there, it is then treated as an update instead of an insert. Existing properties will be updated and new ones created. Old properties not present in the new CSV line are kept (not deleted) in the database. This is to allow "incremental" updates of records. If you wish to delete some properties, the only solution is first to delete the node or relationship before creating it again with the desired new set of properties.

Note that deleting a node will delete all relationships from or to this node.

"Insert" or "update" files are fully identical and can't be differentiated, even with their names. The loader decides whether to perform an update or an insert based

on the current content of the graph. As a side effect, a same file can therefore contain “inserts” or “updates” data indifferently.

7.6 Deletions

Delete files have a special “.node.delete.csv” or “.rel.delete.csv” extension. Apart from that, their syntax is exactly the same than their respective “insert” (.node.csv or .delete.csv extensions) counterpart. The only difference is just that most columns are ignored if provided.

For node files, only the first column, containing the unique index value is considered during a delete operation. This is indeed enough to retrieve the corresponding node from the graph and to delete it. When deleting a node, all relationships, incoming or out-coming, from this node are deleted automatically as well. The corresponding unique entry is also removed from the node index.

For relationship files, only the “@to” and “@from” columns are considered. This is enough to retrieve the corresponding relationship (identified with its name) and delete it from the graph.

7.7 Neo4j transactions

The loader uses one Neo4j transaction per CSV file. It means that if the file is big, the memory consumption may increase significantly. In this case, you can split the file to several ones that will be manage it different sessions.

7.8 Examples

Here follow two simple but complete examples of CSV data files that can be loaded in Neo4j thanks to the UCA for EBC data loading tool.

7.8.1 Rocks bands

This example is made of three files (2 node files and 1 relationship file):

- band.node.csv
- musician.node.csv
- member.rel.csv

It demonstrates a basic database of rock bands and musicians linked by a “member of” relationship.

band.node.csv:

```
@ref(band),@const(band)type
@index(bands)name,genre,popularity
The Beatles,pop,fantastic
The Rolling Stones,rock,great
```

musician.node.csv:

```
@ref(),@const(musician)type
@index(musicians)name,instrument
Paul McCartney,bass
John Lennon,guitar
Georges Harrison,guitar
Ringo Star,drums
Mick Jagger,singer
Keith Richards,guitar
Brian Jones,guitar
Charlie Watts,drums
Bill Wyman,bass
```

member.rel.csv:

```
@rel(member)
@from(bands)name,@to(musicians)name
The Beatles,Paul McCartney
The Beatles,John Lennon
The Beatles,Georges Harrison
The Beatles,Ringo Star
The Rolling Stones,Mick Jagger
The Rolling Stones,Keith Richards
The Rolling Stones,Brian Jones
The Rolling Stones,Charlie Watts
The Rolling Stones,Bill Wyman
```

7.8.2 Network entities

In this other example, we simulate a basic IT network with servers and routers as nodes and wire connections as relationships between nodes.

```
$ ls *.csv
```

```
router_connections.rel.csv
```

```
routers.node.csv
```

```
server_connections.rel.csv
```

```
servers.node.csv
```

```
$ cat router_connections.rel.csv
```

```
@rel(connectedTo)
@from(routers)name,@to(routers)name,bandwidth
Router1, Router2, 10 Gbps
```

```
$ cat routers.node.csv
```

```
@ref(router)
@index(routers)name, vendor
Router1, Cisco
Router2, 3Com
```

```
$ cat server_connections.rel.csv
```

```
@rel(connectedTo)
@from(servers)IP,@to(routers)name,bandwidth
1.1.1.1, Router1, 1 Gbps
1.1.1.2, Router1, 1 Gbps
1.1.1.3, Router1, 1 Gbps
1.1.1.4, Router1, 1 Gbps
2.2.2.1, Router2, 1 Gbps
2.2.2.2, Router2, 1 Gbps
2.2.2.3, Router2, 1 Gbps
2.2.2.4, Router2, 1 Gbps
```

```
$ cat servers.node.csv
```

```
@ref(),@const(server)type
@index(servers)IP,vendor,OS
1.1.1.1,HP,Linux
1.1.1.2,HP,Linux
1.1.1.3,HP,Linux
1.1.1.4,HP,Linux
2.2.2.1,HP,Linux
2.2.2.2,HP,Linux
2.2.2.3,HP,Linux
2.2.2.4,HP,Linux
```

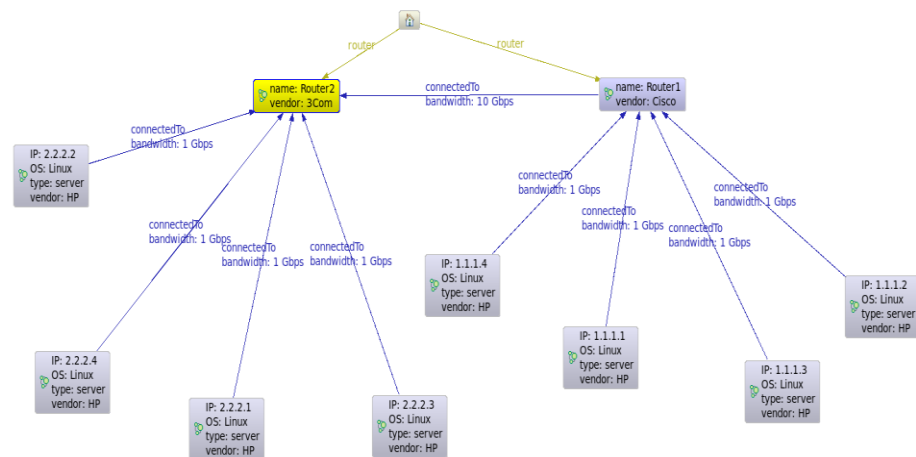


Figure 3 - Graph visualization in Neo4j

7.9 Run the data load

In a UCA runtime environment, the Neo4j data loader is installed as a Neo4j server plug-in. In a UCA development, the loader can be invoked from a Junit test case using a Java API.

7.9.1 From UTM

The UTM populator hides the usage of the CSV loader. Please refer to the UTM documentation for details on how configuring the UCA populator.

7.9.2 From the UCA GUI

Once some CSV files have been placed in the import directory, the UCA GUI can be used to perform the data load manually. In the “Data load” panel, just push the “topology data load” button. Please refer to this section for more details: 4.2.

7.9.3 From Java or Scala code (in Unit tests)

The main direct usage of the CSV loader is probably in the Unit Tests of a UCA Value Pack. This is useful to test correlation rules that make use of topology information in their decision process. A sample topology is then created for the life time of the tests. This sample topology can be defined as a set of CSV files, which is loaded in the embedded Neo4j server at the beginning of a test suite. This is more convenient than creating a graph manually using the Neo4j Java API.

In this case, the CSV loader is invoked through the `org.neo4j.loader.csv.Loader` class. In particular, this class has a `LoadAll()` method that will upload all files present in the import directory.

The procedure to use the CSV loader from Java is the following:

1. Create an instance of the `org.neo4j.loader.csv.Loader` class using the class constructor:

```
Loader loader = new
Loader(ExtendedTopoAccess.getGraphDB(),
tmpDir.tmpCsvPath);
```

The first argument is the `GraphDatabaseService` object representing the access point to the Neo4j server we wish to use. Within a UCA environment it can be retrieved through the static method: `ExtendedTopoAccess.getGraphDB()`.

The second argument of the constructor is the import directory where to find the CSV files to be loaded. It is usually a temporary directory on disk (like `/tmp` on Linux). It can be flushed safely at the end of the test.

2. Copy the desired CSV in the temporary import directory. Be careful in this step to make real copies of the files in a temporary directory. Especially, do not be tempted to directly use the location of some resource files from your development Project (e.g. Maven or Eclipse): remember that the CSV loader will move the CSV files automatically in an archive directory once the data load is complete!

3. Call the `LoadAll()` method:

```
Report report = loader.loadAll();
```

It returns an `org.neo4j.loader.csv.Report` instance holding useful statistics about the performed data load. It can be dumped with the `toString()` method.

7.9.4 Through the Neo4j server plugin and REST API

The CSV loader Neo4j server plug-in is installed by default when UCA for EBC is set up to use an embedded Neo4j server (i.e. when the ***uca.ebc.topology*** property is set to “**embedded**” in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file).

Alternatively, if an external Neo4j server is used (i.e. when the ***uca.ebc.topology*** property is set to “**external**” in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file) then you have to finalize the installation of the data loading plug-in manually as explained in chapter 3.2 “Configuring the topology data load function” of this document.

Once the Neo4j server (embedded in UCA or external) is started, should see a “load_csv” REST service available:

```
$ curl -v http://localhost:7474/db/data/
  "LoadCsv" : {
    "load_csv" :
    "http://localhost:7474/db/data/ext/LoadCsv/graphdb/load_csv"
  },
}
```

To trigger a data load, issue a post request to the corresponding service (change the URL accordingly).

```
$ curl -X POST
http://localhost:7474/db/data/ext/LoadCsv/graphdb/load_csv
```

By default, the CSV loader Neo4j server plugin reads CSV files from a specific directory on disk: `$NEO4J_HOME/import/csv`. In this case, `$NEO4J_HOME` is the root directory of the Neo4j server on your system. Another location can be given as argument as follows:

```
$ curl -X POST
http://localhost:7474/db/data/ext/LoadCsv/graphdb/load_csv \
-H "Content-Type: application/json" \
-d '{"csvpath":"your_import_directory"}'
```

7.10 Data Load output

7.10.1 Load Report

As a result of the data load, a new directory is always created, suffixed by the date and time of the data load. For example:

```
dataload_2012-06-11_03_29_33/
```

The original CSV files are moved to this directory, to prevent them from being imported again at the next load request.

The directory also contains a “report.txt” file, summarizing the data load outcome.

For example:

```
CSV Import from: /opt/UCA-EBC/neo4j/import/csv, at : Mon Jun 11 15:29:34 CEST
2012
-----
musician.node.csv: complete load: 9 loaded successfully, 0 discarded, in 387 ms
band.node.csv: complete load: 2 loaded successfully, 0 discarded, in 41 ms
member.rel.csv: load complete: 9 loaded successfully, 0 discarded, in 187 ms
-----
Total: 20 nodes or relationships inserted (0 discarded), in 616 ms
```

7.10.2 Discarded entities

It may happen that some rows in the CSV files are badly formatted (for instance some columns are missing). In this case, the row is discarded (resulting in no node or relationship being created) and pushed to a “.discarded” file: this file has the same name than the input CSV file, with an additional “.discarded” extension.

The main situation where relationship entries get discarded is when the nodes that make up a relationship referenced using @from or @to annotations cannot be found in the indexes. This typically happens when a node has been previously discarded because a property was missing. As a result all relationships from or to this node will be discarded as well.

7.11 Configuration

The loader can be configured through the following configuration files:

- csv-loader.conf, located wherever on the application classpath
- reference.conf, located wherever on the classpath. This is the fallback configuration file (that can be shared with other applications) in case the csv-loader.conf one is not found.

The file syntax follows the Typesafe config conventions:

```
csv-loader {
    separator = ","
    arraySeparator = "!"
    dumpReport = true
    bulkCommit = 1000
}
```

The following configuration values can be customized:

separator: this is the separator character used to distinguish columns in the CSV files. This is traditionally the comma character: “,”, as in “Comma” Separated Values. It can be changed to any other character. The new character value has to be given between double quotes (as a string).

arraySeparator: this is the character used to separate elements in case a Neo4j property is set to be an “array” data type, through the @type header annotation.

For example if an header property is defined as follows, with 2 example rows:

```
@type(string[])an_array_property  
Hello%uca%world!  
The%meaning%of%life%is%42
```

The arraySeparator character must be set to “%” so that the values are parsed correctly and stored as arrays of string in Neo4j.

dumpReport: this is to tell the loader either to dump or not the report about how many nodes or relationships have been loaded or discarded in what amount of time. This could be useful for statistic purpose and to verify that the loading was fully successful or not.

bulkCommit: this is to support bulk transactions in the graph DB. When not specified, only one single transaction is opened for an entire CSV file, and the commit is done at the end.

If the input file contains many lines of data to load, this could lead to a memory overflow of the JVM because current update is kept in memory for a possible rollback. To avoid such problem, when specifying this value, the loader will perform a commit at regular intervals, interval which is specified by the value (by default 1000).

Note that this value can have a big influence on the performance and memory consumption of the loader (and embedding JVM). The bigger it is, the best the performance is, but more memory is used.

For information, with default settings during a test, the CSV loader has successfully performed a load of 7 million nodes and relationships without any problem in about 1 hour (on a small VM).

The configuration values take effect for all CSV files to be loaded. A fine, per-fine tuning is not possible.

7.12 Logging

UCA-EBC embedded neo4j server:

The traces are activated through the UCA-EBC logging mechanism. Add the following logger definition to the uca-ebc-log4j.xml configuration file.

```
<logger name="org.neo4j.loader.csv" additivity="false">  
  <level value="INFO" />  
  <appender-ref ref="CONSOLE" />  
  <appender-ref ref="FILE" />  
  <appender-ref ref="DB" />  
</logger>
```

External Neo4J server :

The CSV loader uses [logback](#) has a default logging back end.

Logging levels can then be set in the `logback.xml` file, if found on the application classpath.

The package name to use is “`org.neo4j.loader.csv`”. The default log level is “`INFO`”.

Thus, to activate debug traces, just change the following line the `logback.xml` file (replacing “`INFO`” by “`DEBUG`”).

```
<logger name="org.neo4j.loader.csv" level="DEBUG"/>
```

Glossary

CSV: Comma-Separated Values

DRL: Drools Rule file

EBC: Event Based Correlation

EVP: UCA for EBC Value Pack

Inference Engine: Process that uses a Rete algorithm

IP: Installation Package for OSS Open Mediation V6.2

JDK: Java Development Kit

JMS: Java Messaging Service

JMX: Java Management eXtension, used to access or process action on the UCA for EBC product

JNDI: Java Naming and Directory Interface

JRE: Java Runtime Environment

UCA: Unified Correlation Analyzer

X733: Standard describing the structure of an Alarm used in telecommunication environment

XML: Extensible Markup Language

XSD: Schema of an XML file, describing its structure