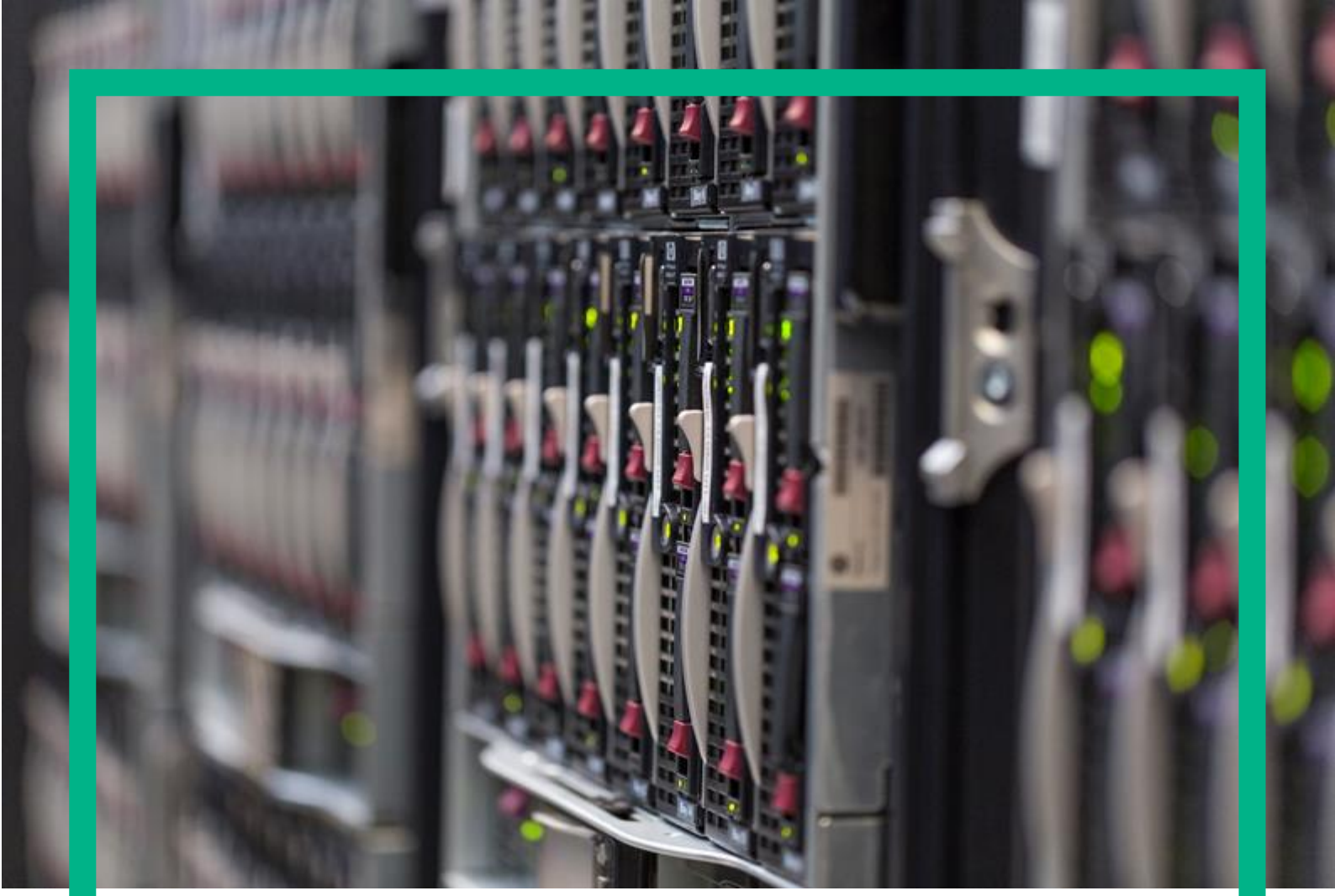


Unified Correlation Analyzer

Version 3.4
Administration, Configuration and Troubleshooting Guide

Edition: 1.0

Notices

Legal notice

Confidential computer software. Valid license from HPE 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.

The information contained herein is subject to change without notice. The only warranties for HPE 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. HPE shall not be liable for technical or editorial errors or omissions contained herein.

Printed in the US

Warranty

The information contained herein is subject to change without notice. The only warranties for HPE 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. HPE shall not be liable for technical or editorial errors or omissions contained herein.

Trademarks

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

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

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

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.

Hazelcast™ is a trademark of Hazelcast Inc.

Apache Kafka™ is a trademark of the Apache Software Foundation.

Apache ZooKeeper™ is a trademark of the Apache Software Foundation.

Contents

List of tables

List of figures

Preface

About this guide

This guide provides an overview of Unified Correlated Analyzer for Event Based Correlation product and describes how to administer, configure, monitor and troubleshoot the UCA for EBC product.

Product Name: Unified Correlation Analyzer for Event Based Correlation (also referred to in this document as UCA for EBC)

Product Version: 3.4

Kit Version: 3.4

Audience

Here are some recommendations based on possible reader profiles:

- Solution Developers and integrators
- 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:

Table 1: Software versions

Product Version	Supported Operating systems
UCA for Event Based Correlation Server Version 3.4	• HP-UX 11.31 for Itanium • Red Hat Enterprise Linux Server release 6.5 & 7.2
UCA for Event Based Channel Adapter 3.4	• HP-UX 11.31 for Itanium • Red Hat Enterprise Linux Server release 6.5 & 7.2
UCA for Event Based Correlation Software Development Kit Version 3.4	• Windows XP / Vista 64 bits • Windows Server 2012 • Windows 7 64 bits

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:

[R1] `HPE UCA for EBC Reference Guide`

[R2] `HPE UCA for EBC Value Pack Development Guide`

[R3] `HPE UCA for EBC User Interface Guide`

[R4] `HPE UCA for EBC Installation Guide`

[R5] `HPE UCA for EBC Topology Extension Guide`

[R6] `HPE UCA for EBC Clustering and HA Guide`

Support

Please visit our HPE Software Support Online Web site at https://softwaresupport.hpe.com/ for contact information, and details about HP Enterprise 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 describes how to administer, configure, monitor and troubleshoot the UCA for EBC product.

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.

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.
At installation, a single `<instance-name>` is configured: default.

NOTE:
For more information on how to install the UCA for EBC product, please refer to [R4] `HPE UCA for EBC Installation Guide`.

NOTE:
For more information on how to install the UCA for EBC product, please refer to [R1] `HPE UCA for EBC Reference Guide`.

Chapter 2
UCA for EBC Administration

2.1 Starting and stopping UCA for EBC

2.1.1 Starting UCA for EBC

To start UCA for EBC, please run the following commands as uca user:

On both HP-UX and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc start
```

Here's a sample output from this command:

```
Using UCA for EBC Home directory specified by the UCA_EBC_HOME environment
variable: /opt/UCA-EBC
Using UCA for EBC Data directory specified by the UCA_EBC_DATA environment
variable: /var/opt/UCA-EBC
*** INFO: Starting UCA for Event Based Correlation version 3.4
```

Traces are logged in the `${UCA_EBC_INSTANCE}/logs/uca-ebc.log` file.

To start UCA for EBC in verbose mode (traces logged to the console), please run the following commands as uca user (note the use of the –v option):

On both HP-UX and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc –v start
```

Since UCA-EBC V2.0, it is possible to launch multiple instances on a same machine. Each instance is managed by the `uca-ebc-instance` command line tool (refer to chapter 2.2.4). If not specified, the default instance is launched.

To start UCA for EBC for a specific instance (specified by `<instance-name>` in the example below), please run the following commands as **uca** user (note the use of the –i option):

On both HP-UX and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc –i <instance-name> start
```

2.1.2 Stopping UCA for EBC

In order to stop UCA for EBC, please run the following commands as **uca** user:

On both HP-UX and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc stop
```

Here's a sample output from this command:

```
Using UCA for EBC Home directory specified by the UCA_EBC_HOME environment
variable: /opt/UCA-EBC
Using UCA for EBC Data directory specified by the UCA_EBC_DATA environment
variable: /var/opt/UCA-EBC
*** INFO: Shutting down UCA for Event Based Correlation version 3.4
*** INFO: UCA for Event Based Correlation version 3.4 has been successfully
stopped
```

Since UCA-EBC V2.0, it is possible to have multiple instances running on a same machine. If not specified, the default instance is stopped.

To stop UCA for EBC for a specific instance (specified by <instance-name> in the example below), please run the following commands as uca user (note the use of the –i option):

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc –i <instance-name> stop
```

2.1.3 Displaying the status of UCA for EBC

In order to show the status of UCA for EBC, please run the following commands as **uca** user:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc status
```

Here's a sample output from this command:

```
Using UCA for EBC Home directory specified by the UCA_EBC_HOME environment
variable: /opt/UCA-EBC
Using UCA for EBC Data directory specified by the UCA_EBC_DATA environment
variable: /var/opt/UCA-EBC
*** INFO: UCA for Event Based Correlation version 3.4 is running
```

The status of UCA for EBC can either be "Running" or "Stopped".

Since UCA-EBC V2.0, it is possible to have multiple instances running on a same machine. If not specified, the status of the default instance is returned.

To get the status of UCA for EBC for a specific instance (specified by <instance-name> in the example below), please run the following commands as uca user (note the use of the –i option):

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc –i <instance-name> show
```

2.2 Command-line tools

Some command-line tools are provided in the ${UCA_EBC_HOME}/bin folder that may prove to be of some help to users of UCA for EBC:

- uca-ebc-inventory: this command-line tool lists the UCA for EBC packages installed on the system
- uca-ebc-injector: this command-line tool provides the capability to inject alarms or events described in XML files directly into the UCA for EBC input queue without going through the mediation layer (OSS Open Mediation V7.2 or UMB V1.1), thus bypassing both OSS Open Mediation V7.2 and UCA for EBC Channel Adapter (or UMB V1.1 and UCA for EBC Adapter if UMB is used)
- uca-ebc-admin: this command-line tool provides a lot of options to configure, administer, and monitor UCA for EBC, but also UCA for EBC value packs and scenarios. Most of the features of this tool are also available using the UCA for EBC User Interface
- uca-ebc-instance: this command line tool manages the different instances of UCA for EBC. It provides options to list current instances, add a new instance, delete or rename an existing instance and set the default instance name
- uca-ebc-backup: this command line tool provides facilities for backup and restore of the instances of UCA for EBC

All command-line tools should be run under the ***uca*** user account.

For more information on the UCA for EBC User Interface, please refer to: [R3] `HPE UCA for EBC User Interface Guide`

2.2.1 uca-ebc-inventory

This command-line tool lists the packages (including patches) installed on the system for the following products:

- UCA for EBC Server
- UCA for EBC Channel Adapter for OSS Open Mediation
- UCA for EBC Development Kit
- OSS Open Mediation and OSS Open Mediation Channel Adapters
- UMB and UMB Adapters

To execute the *`uca-ebc-inventory`* tool, please use the following commands:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc-inventory
```

Here's an example of the output of the execution of uca-ebc-inventory:

```
-------------------------------------------------
      UCA For Event Based Correlation
           Components Inventory
          on <hostname> system
-------------------------------------------------
Installed UCA-EBC components:
UCA-EBCSERVER       3.4-0A    HPE UCA EBC Server Version V3.4 Level 0 Rev A
UCA-EBCCA           3.4-0A    HPE UCA EBC Channel Adapter Version V3.4 Level
0 Rev A
UCA-EBCTOPO         3.4-0A   HPE UCA EBC Topology features Version V3.4 Level
0 Rev A

Installed Mediation components:

ngossopenmediation  V720-RHEL6    HPE CMS Open Mediation Version 7.2.0
---------------- END of UCA INVENTORY ------------------
```

The uca-ebc-inventory tool has no execution options and no associated configuration file.

2.2.2 uca-ebc-injector

This command-line tool provides the capability to easily send events to UCA for EBC by pushing XML files containing these events to the JMS input queue (implemented as a JMS Topic) of UCA for EBC.

Events can be Alarm creation, Alarm Attribute Value Change, Alarm State Change, Alarm Deletion, but can also be any object of classes extending the DefaultEvent class.

The events are directly injected into UCA for EBC without going through the mediation layer (OSS Open Mediation V7.2 or UMB V1.1), thus bypassing both OSS Open Mediation V7.2 and UCA for EBC Channel Adapter (or UMB V1.1 and UCA for EBC Adapter if UMB is used).

This command-line tool can be very helpful for testing UCA for EBC Value Packs in real conditions without having to set up the mediation layer.

The uca-ebc-injector tool can read files containing alarms and more generally files containing any class of event extending DefaultEvent.

Example format of an Alarms file

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

<Alarms xmlns="http://hp.com/uca/expert/x733Alarm">
      <AlarmCreationInterface>
            <sourceIdentifier>src</sourceIdentifier>
            <identifier>1</identifier>
             <originatingManagedEntity>B1</originatingManagedEntity>
            <alarmType>COMMUNICATIONS_ALARM</alarmType>
            <probableCause>Fire</probableCause>
            <perceivedSeverity>MINOR</perceivedSeverity>
            <alarmRaisedTime>2009-09-16T12:00:00</alarmRaisedTime>
            <targetValuePack>myVP##temipFlow</targetValuePack>
      </AlarmCreationInterface>

<AlarmCreationInterface>
[. . .]
</AlarmCreationInterface>
</Alarms>
```

CAUTION:
Events file cannot contain directly the raw description of events.
Events in events file have to be packaged in simple structures called `EventBoxBase` containing the event itself and an attribute indicating the class of the event. The information about the class of the event will be used by UCA to correctly un-marshal the event.

Example format of an Events file

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

<Events xmlns="http://hp.com/uca/expert/event">

<EventBoxBase eventClassName="com.hp.umb.example.metrics.Temperature">
    <eventString><![CDATA[<temperature
xmlns:ns2="http://hp.com/uca/expert/event"
xmlns:ns3="http://hp.com/umb/example/metrics"
xmlns:ns4="http://hp.com/uca/expert/x733Alarm">
<ns2:identifier>100</ns2:identifier>
<ns2:eventTime>0</ns2:eventTime>
<ns2:targetValuePack>MyVP</ns2:targetValuePack>
```

```
<ns3:value>37.2</ns3:value>
</temperature>]]>
    </eventString>
</EventBoxBase>

</Events>
```

IMPORTANT: All the classes of events sent by the uca-ebc-injector must be loaded in UCA-EBC. For that purpose, this classes must be packaged in a .jar file placed in the `${UCA_EBC_INSTANCE}/externallib` directory, and uca-ebc must be restarted.
In the example above, the class `com.hp.umb.example.metrics.Temperature` must be wrapped in a jar file put in `${UCA_EBC_INSTANCE}/externallib` (by default `/var/opt/UCA-EBC/instance/default/externallib`)

NOTE: When events are received by UCA through the UMB mediation layer, and when events logging is activated (ref **6.1.2.2**), the events are logged in the uca-ebc-received-event.log directly in `EventBoxBase` structures as above, making it easy to re-inject them with the uca-ebc-injector. For that purpose, duplicate the uca-ebc-received-event.log file into a myEventsFile.xml and make sure that myEventsFile.xml looks like the file below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Events xmlns="http://hp.com/uca/expert/event">
<!-- put the content of uca-ebc-received-event.log here -->
</Events>
```

The following sections describe how to execute and how to configure the uca-ebc-injector tool.

To execute the uca-ebc-injector tool, please use the following commands:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc-injector <options>
```

<*options*> is a list of valid options for the uca-ebc-injector tool

The uca-ebc-injector command-line tool can be used either in random mode, where random alarms or events are generated automatically based on a template and sent to UCA for EBC, or in file mode, where alarms/events are provided to the uca-ebc-injector tool as an XML file that is then sent to UCA for EBC.

The uca-ebc-injector tool is by default in file mode unless the -r or --random option is used, in which case the uca-ebc-injector tool is in random mode.

To use the uca-ebc-injector tool in file mode, please use the following commands:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc-injector -file /tmp/Events.xml
```

The above command will send 1 burst of alarms to UCA for EBC. The alarms in this burst will be exactly the same as the alarms in the file specified by the -file or --filename option.

To use the uca-ebc-injector tool in random mode, please use the -r or --random option. Below is an example of the uca-ebc-injector tool being used in random mode:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc-injector --random -file /tmp/Alarms.xml --number 10 --delay 5000
```

The above command will send 10 bursts of random alarms to UCA for EBC. The delay between each burst will be 5 seconds. Each burst of alarms will send one alarm unless the --buffer-size option is specified. The alarms sent in the burst will be the same as the alarms in the template file except for the ID of the alarms (sequential IDs will be used instead) and the severity of the alarm (the severity will be chosen at random).

It is possible to have multiple instances running on a same machine. If not specified, the *uca-ebc-injector* tool applies to the default instance.

This tool has the following options available:

Table 2: uca-ebc-injector tool options

Option name	Description
-i <instance-name>	**Default value**: the default instance name This option sets the instance of UCA for EBC to use. Instance <instance-name> must exist. If used, this option must be set as first option.
--buffer-size <Size>	**Default value**: 1 This option is used in random mode (-r, or --random option) to specify the number of alarms per alarm burst.
--delay <Delay>	**Default value**: 0 This option specifies the delay (in milliseconds) between 2 alarms files (in file mode) or 2 alarm bursts (in random mode).
-f, -file <Filename>	**No default value** This option sets the uca-ebc-injector tool in file or random modes. It specifies one alarm file to use as input for the uca-ebc-injector tool. The file specified by <filename> must be a valid XML file complying with the Alarm XSD file located at the following location : `${UCA_EBC_HOME}/schemas/uca-expert-alarm.xsd`
--number <Number>	**Default value**: 1 This option is used in random mode (-r, or --random option) to specify the number of alarm bursts to be sent
-r, --random	This option sets the uca-ebc-injector tool in random mode. This option can be used in conjunction with the -file option to send random alarms (sequential IDs, random severity) based on the alarms provided with the -file option

The uca-ebc-injector tool has some configuration properties defined in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file, but these properties are *FOR INTERNAL USE ONLY*, and are not meant to be updated.

The following table lists these properties for reference only:

Table 3: Properties for uca-ebc-injector in uca-ebc.properties file

Property name	Explanation
java.naming.factory.initial	**Default value** : org.apache.activemq.jndi.ActiveMQInitialContextFactory *FOR INTERNAL USE ONLY. DO NOT UPDATE THE VALUE OF THIS PROPERTY.*
java.naming.provider.url	**Default value** : tcp\://${uca.ebc.serverhost}\:${uca.ebc.jms.broker.port} *FOR INTERNAL USE ONLY. DO NOT UPDATE THE VALUE OF THIS PROPERTY.*
topic.uca-ebc-alarms	**Default value** : com.hp.uca.ebc.alarms *FOR INTERNAL USE ONLY. DO NOT UPDATE THE VALUE OF THIS PROPERTY.*
topic.uca-ebc-events	**Default value** : com.hp.uca.ebc.events *FOR INTERNAL USE ONLY. DO NOT UPDATE THE VALUE OF THIS PROPERTY.*

2.2.3 uca-ebc-admin

This command-line tool provides a lot of options to configure, administer, and monitor UCA for EBC Server, but also UCA for EBC value packs and scenarios. Most of the features of this tool are also available using the UCA for EBC User Interface.

The following sections describe how to execute and how to configure the uca-ebc-admin tool.

To execute the uca-ebc-admin tool, please use the following commands:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc-admin <options>
```

<*options*> is a list of valid options for the uca-ebc-admin tool (both main options and sub-options)

It is possible to have multiple instances running on a same machine. If not specified, the uca-ebc-admin tool applies to the default instance. Otherwise, the instance to administer can be specified with the `-i <instance name>` option. This option must be the first option listed.

The following table lists the main options of the uca-ebc-admin tool (sub-options can be used alongside these main options, the list of which is described further):

Table 4: uca-ebc-admin tool main options

Option name	Description
-h, --help	This option displays the uca-ebc-admin tool usage message
-i <instance-name>	This option sets the instance of UCA for EBC to administer. Instance <instance-name> must exist. If used, this option must be the **first option**.

-a, --audit	This option dumps full audit information (including status, performance information):
	
	• information on UCA EBC instance:
	  • Value pack manager
	  • Collector
	  • Dispatcher
	  • Action Threads, Stats, Queue
	  • Alarm forwarders
	• Information on value packs
	  • Mediation flows
	  • Db flows
	• Information on scenarios
	  • Filters
	  • Queue
	  • Working Memory
	  • Scenario/Watchdog threads
	
	This option always applies to all value packs and scenarios.

-s, --stats	This option dumps specific statistics information (including status and some performance information) on all value packs and scenarios or a specific value pack or scenario depending on the sub-options used. ☞ See Notes: (1) (2) (3)
-l, --list	This option lists all Value Packs and Scenarios
-lg, --log4j	This option reloads the UCA for EBC log4j configuration file
-p, --perf	This option displays performance measurements.
-w, --workingMemory	This option dumps the working memory of one or more scenarios. By default it applies to all scenarios of all value packs except if sub-options are used. ☞ See Notes: (1) (2) (3)
-c, --clean	This option cleans the working memory (retracts all facts) of one or more scenarios. By default it applies to all scenarios of all value packs except if sub-options are used. ☞ See Notes: (1) (2) (3)
-r, --reload	This option reloads the rule engine of one or more scenarios or reloads a specific rules file.

	By default this option reloads the rule engine of all scenarios of all value packs except if sub-options are used. ☞ See Notes: (1) (2) (3) (4)
-rc, --reloadConf	This option reloads the configuration files. The files to be reloaded can be chosen between the: - whole set of files of all actives value packs - whole set of files of a single active value pack - whole set of files concerning a single scenario - a single file within a scenario when used in conjunction with the –conf sub-option. ☞ See Notes: (1) (2) (3) (5)
-dep, --deploy	This option deploys a value pack stored in the *${UCA_EBC_INSTANCE}/valuepacks* directory into the *${UCA_EBC_INSTANCE}/deploy* directory. This option applies to the selected value pack. ☞ See Note: (2) Once deployed, the value pack can be started by executing the uca-ebc-admin tool with the -start, --start option (if UCA for EBC is already running) or by starting UCA for EBC (if UCA for EBC is stopped).
-undep, --undeploy	This option undeploys a value pack from the *${UCA_EBC_INSTANCE}/deploy* directory and creates an archive (ZIP file) of it in the *${UCA_EBC_INSTANCE}/valuepacks* directory. The zipped value pack that was previously present in the *${UCA_EBC_INSTANCE}/valuepacks* directory is moved to the *${UCA_EBC_INSTANCE}/archive* directory and a timestamp is added to the file name. This option applies to the selected value pack. ☞ See Note: (2) Once the value pack has been undeployed, it can be deployed back again by using the -deploy, --deploy option.
-start, --start	This option starts a value pack. This option applies to the selected value pack. ☞ See Note: (2)
-stop, --stop	This option stops a value pack. This option applies to the selected value pack. ☞ See Note: (2)
-d, --disable	This option disables:

	• either rule engine logging (if -rl,--ruleLogging option is also selected) • or scenario logging (if -sl,--scenarioLogging option is also selected).
-e, --enable	This option enables: • either rule engine logging (if -rl,--ruleLogging option is also selected) • or scenario logging (if -sl,--scenarioLogging option is also selected).
-rl, --ruleLogging	Used in conjunction with either the –d, --disable or –e, --enable options, this option enables or disables rule engine logging for one or more scenarios. By default it applies to all scenarios of all value packs except if sub-options are used. ☞ See Notes: (1) (2) (3)
-startflow, --startflow	This option starts a mediation flow. Used with the -vpn <*value pack name*> and –vpv <*value pack version*> sub-options, this option applies to all the mediation flows of the selected value pack. Used with the -vpn <*value pack name*>, –vpv <*value pack version*>, and –flow <*flow name*> sub-options, this option applies to the selected mediation flow of the selected value pack.
-stopflow, --stopflow	This option stops a mediation flow. Used with the -vpn <*value pack name*> and –vpv <*value pack version*> sub-options, this option applies to all the mediation flows of the selected value pack. Used with the -vpn <*value pack name*>, –vpv <*value pack version*>, and –flow <*flow name*> sub-options, this option applies to the selected mediation flow of the selected value pack.
-resyncflow, --resyncflow	This option resynchronizes a mediation flow. Used with the -vpn <*value pack name*> and –vpv <*value pack version*> sub-options, this option applies to all the mediation flows of the selected value pack. Used with the -vpn <*value pack name*>, –vpv <*value pack version*>, and –flow <*flow name*> sub-options, this option applies to the selected mediation flow of the selected value pack.
-statusflow, --statusflow	This option displays the status of a mediation flow.

	Used with the -vpn <*value pack name*> and –vpv <*value pack version*> sub-options, this option applies to all the mediation flows of the selected value pack. Used with the -vpn <*value pack name*>, –vpv <*value pack version*>, and –flow <*flow name*> sub-options, this option applies to the selected mediation flow of the selected value pack.
-dumpfa, --dumpfailedactions	Dumps failed actions of a scenario to the logs. This option applies to the selected scenario. ☞ See Note: ([3])
-retractfa, --retractfailedactions	Retracts failed actions of a scenario from Working Memory This option applies to the selected scenario. ☞ See Note: ([3])
-R, --restartServer	Restart the UCA-EBC Server ☞ See Note: ([6])
-S, --showServer	Shows the status of UCA-EBC Server ☞ See Note: ([6])
-T, --stopServer	Stops the UCA-EBC Server ☞ See Note: ([6])

Here's the list of notes that applies to the above "uca-ebc-admin tool main options" table:

NOTE:

([1]) If no sub-option is selected, then the option applies to all value packs or all their scenarios

([2]) If -vpn <*value pack name*> and –vpv <*value pack version*> sub-options are selected, then the option applies to the specified value pack or all its scenarios

([3]) If -vpn <*value pack name*>, -vpv <*value pack version*>, and -scenario <*scenario name*> sub-options are selected, then the option applies to the specified scenario

([4]) If -vpn <*value pack name*>, -vpv <*value pack version*>, -scenario <*scenario name*>, and -rule <*rules file identifier*> sub-options are selected, then the option applies to the specified rules file.

([5]) If -vpn <*value pack name*>, -vpv <*value pack version*>, -scenario <*scenario name*>, and -conf <*configuration file identifier*> sub-options are selected, then the option applies to the specified configuration file.

([6]) If -i <*instance name*> option is selected, then the option applies to the specified UCA-EBC Server instance. Otherwise it applies to the default UCA-EBC Server instance.

The following table lists the sub-options that can be used in conjunction with the main options of the uca-ebc-admin tool:

Table 5: uca-ebc-admin tool sub-options

Option name	Description
-vpn <value pack name>	Used in conjunction with the -vpv sub-option, this sub-option selects the value pack specified by <*value pack name*> and <*value pack version*>. This sub-option can be used alongside the following options: • -w, --workingMemory • -c, --clean • -r, --reload • -dep, --deploy • -undep, --undeploy • -start, --start • -stop, --stop • -rl, --ruleLogging • -sl, --scenarioLogging • -startflow, --startflow • -stopflow, --stopflow • -resyncflow, --resyncflow • -statusflow, --statusflow -s, --stats
-vpv <value pack version>	Used in conjunction with the -vpn sub-option, this sub-option selects the value pack specified by <*value pack name*> and <*value pack version*>. This sub-option can be used alongside the following options: • -w, --workingMemory • -c, --clean • -r, --reload • -dep, --deploy • -undep, --undeploy • -start, --start • -stop, --stop • -rl, --ruleLogging • -sl, --scenarioLogging • -startflow, --startflow • -stopflow, --stopflow • -resyncflow, --resyncflow • -statusflow, --statusflow • -a, --audit -s, --stats
-scenario <scenario name>	Used in conjunction with the -vpn, and -vpv sub-options, this sub-option selects the scenario specified by <*value pack name*>, <*value pack version*>, and <*scenario name*>. This sub-option can be used alongside the following options: • -w, --workingMemory • -c, --clean

	• -r, --reload • -rl, --ruleLogging -sl, --scenarioLogging
-rule <rules file identifier>	Used in conjunction with the -vpn, -vpv, and -scenario sub-options, this sub-option selects the rules file specified by `<value pack name>`, `<value pack version>`, `<scenario name>`, and `<rules file identifier>`. This sub-option can be used alongside the following options: • -r, --reload The rules file identifier is the name that is associated with a rules file for a specific scenario (see `ValuePackConfiguration .xml` file).
-flow <mediation flow name>	Used in conjunction with the -vpn, and -vpv sub-options, this sub-option selects the mediation flow specified by <*value pack name*>, <*value pack version*>, and <*mediation flow name*>. This sub-option can be used alongside the following options: • -startflow, --startflow • -stopflow, --stopflow • -resyncflow, --resyncflow • -statusflow, --statusflow The mediation flow name is the name that is associated with a specific mediation flow (see `ValuePackConfiguration .xml` file).

-conf <configuration file identifier>	Used in conjunction with the -vpn, -vpv, and -scenario sub-options, this sub-option selects the rules file specified by <*value pack name*>, <*value pack version*>, <*scenario name*>, and <*configuration file identifier*>.
	
	This sub-option can only be used alongside the following options:
	
	• -rc, --reloadConf
	
	The configuration file identifier is either:
	
	• One of the keywords :
	  ◦ filter
	  ◦ mapper
	  ◦ specific
	  ◦ template
	• the filename of a specific configuration file
	• the name of the template

	If the keyword "specific" is used, all specific configuration files are selected.

The uca-ebc-admin tool has some configuration properties defined in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file, but these properties are *FOR INTERNAL USE ONLY*, and are not meant to be updated.

The following table lists these properties for reference only:

Table 6: Properties for uca-ebc-admin in uca-ebc.properties file

Property name	Explanation
uca.ebc.jmx.url	**Default value** : service\:jmx\:rmi\://${uca.ebc.serverhost}/jndi/rmi\://${uca.ebc.serverhost}\:${uca.ebc.jmx.rmi.port}/uca-ebc *FOR INTERNAL USE ONLY. DO NOT UPDATE THE VALUE OF THIS PROPERTY.*

2.2.4 uca-ebc-instance

The uca-ebc-instance command-line tool provides options to create, delete, list or configure instances of UCA for EBC Server.

Instances are created in the *`${UCA_EBC_DATA}/instances directory.`* At installation, a single instance is created. It is named "**default**".

To execute the uca-ebc-instance tool, please use the following commands:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc-instance <options>
```

<*options*> is a list of valid options for the uca-ebc-instance tool

The following table lists the main options of the uca-ebc-instance tool:

Table 7: Main options for the uca-ebc-instance tool

Option name	Description
-h	This option displays the uca-ebc-instance tool usage message
-l	This option lists all available instances.
-a <instance-name>	This option creates a new instance named <instance-name> ☞ See Notes: ([1]) ([2])
-d <instance-name>	This option deletes an existing instance named <instance-name>.
-r <old-name> <new-name>	This option renames an existing instance named <old-name> to <new-name>. Note that <new-name> should not already exist.

-s <instance-name>	This option sets the default instance to use to be: <instance-name>. ☞ See Note: ([3])

NOTE:

([1]) When creating a new instance, the root folder for the new instance is created. This folder is referred to as `${UCA_EBC_INSTANCE}` in this document.

([2]) When creating a new instance, please make sure that there is no port conflict with other applications running on your server.

([3]) When no "-i" option is provided with the uca-ebc, uca-ebc-admin, uca-ebc-injector, or the uca-ebc-backup tool, the default instance is used.

Please refer to chapter 3.1 "Multiple instances configuration" for more information on how to configure multiple instances of UCA for EBC.

2.2.5 uca-ebc-backup

This command-line tool provides the ability to backup and restore UCA for EBC Server instances.

To execute the uca-ebc-backup tool, please use the following commands:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc-backup <command> <options>
```

<*command*> is one of [-b | -backup | -r | -restore | -l | -list]

<*options*> is a list of valid options for the command

2.2.5.1 Backing up

When the –b | -backup option is given to the uca-ebc-backup tool, a backup of the data directory for a specific instance is performed (excluding the logs and work subdirectories). In order to do so, the uca-ebc-backup tool compresses the instance directory hierarchy and stores the resulting file into a directory of the users' choice.

If the UCA for EBC Topology Extension is installed along with UCA for EBC Server and the neo4j Server is configured as embedded, the neo4j subdirectory is also backed up. The backup of the neo4j subdirectory is done using the neo4j Enterprise backup utility, which performs a full backup without acquiring any locks, thus allowing for continued operations on the neo4j instance.

Please make sure that UCA for EBC server is up and running when neo4j is embedded before proceeding with a backup. (☞ See Note below)

To back up a UCA for EBC instance, please execute the following command:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc-backup -b|-backup <options>
```

The following table lists the options of the uca-ebc-backup tool for backing up UCA for EBC instances:

Table 8: Options for backing up UCA for EBC instances using the uca-ebc-instance tool

Option name	Description
-h	This option displays the uca-ebc-backup tool usage message
-i <instance-name>	This option specifies the instance of UCA for EBC to backup. If it is not specified, the default instance is used.
-f|-from <directory>	This option specifies the UCA for EBC data directory. If it is not specified, the `${UCA_EBC_DATA}` directory is used.
-t|-to <directory>	This option specifies the directory where to store the backup file. If it is not specified, the `${UCA_EBC_DATA}/backup` directory is used.
-n|-name <name>	This option specifies the name of the file to use as the backup file. If it is not specified, the name of the file is generated automatically using the following pattern: %instance-%date-%time.

NOTE:

When UCA for EBC is not running during the backup procedure, it is not a problem: a warning is displayed even though the neo4j database is backed up properly.

Important: if your neo4j database is located outside of the `${UCA_EBC_INSTANCE}` directory (for example if you set the value of the uca.ebc.topology.location property to /my-absolute-path in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file), the backup tool will keep a copy in a subdirectory of the `${UCA_EBC_INSTANCE}` directory.

2.2.5.2 Restoring

When the –r | -restore option is given to the uca-ebc-backup tool, a specific instance of UCA for EBC is restored from a compressed file previously created by the uca-ebc-backup tool.

Restoring a backup file is only supported when UCA for EBC server is not running. When UCA for EBC server instance is running, restoring a backup of that instance will result in unexpected behavior.

Restoring a backup of a UCA for EBC instance results in the current configuration of neo4j being replaced by the backup. (☞ See Note ([1]) below)

To restore a UCA for EBC instance from a backup file, please use the following command:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc-backup -r|-restore -name filename <options>
```

The following table lists the options of the uca-ebc-backup tool for restoring UCA for EBC instance backup files:

Table 9: Options for restoring UCA for EBC instances using the uca-ebc-instance tool

Option name	Description
-h	This option displays the uca-ebc-backup tool usage message
-n | -name <name>	This option is mandatory and specifies the fully qualified name of the backup file to restore.
-t | -to <directory>	This option specifies the UCA for EBC data directory where to restore the backup file. If it is not specified, ${UCA_EBC_DATA} is used. ☞ See Note below

NOTE:

([1]) The restore mechanism does restore the neo4J DB in the ${UCA_EBC_INSTANCE}/neo4j directory which is the default location of the neo4j DB.
If you have the location of neo4j DB outside of `${UCA_EBC_INSTANCE}` (for example if you specified uca.ebc.topology.location=/my-absolute-path in the `uca-ebc.properties` file), you will have to manually copy the contents of the neo4j subdirectory to the /my-absolute-path directory

([2]) Be careful! The backup file contains the instance name. If an instance with the same name exists when an instance is restored, the existing instance will be overwritten
However, please note that the current logs and work directories are not removed

2.2.5.3 Listing the available backups

When the –l | -list option is given to the uca-ebc-backup tool, all compressed backup files are listed.

It is helpful to run this command before restoring a backup to know what backup files are available. It may also be helpful if you need to do some cleanup of the backup files.

The list is sorted by creation time. It is up to the end-user to clean the backup directory when backup files become irrelevant and should be removed.

To list all available UCA for EBC instance backup files, please use the following command:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$_uca-ebc-backup -l|-list <options>
```

The following table lists the options of the uca-ebc-backup tool for listing available backup files:

Table 10: Options for listing the available UCA for EBC instance backups using the uca-ebc-backup tool

Option name	Description
-h	This option displays the uca-ebc-backup tool usage message
-f|-from <directory>	This option specifies the directory where the backup files are stored. If it is not specified, the ${UCA_EBC_DATA}/backup directory is used.

2.3 UCA for EBC User Interface

In addition to the command-line tools, the web-based user interface of UCA for EBC also provides administration, monitoring and troubleshooting capabilities for the UCA for EBC product.

NOTE: For more information on how to configure UCA for EBC at the value pack or scenario level please refer to: [R3] `HPE UCA for EBC User Interface Guide`.

Chapter 3
UCA for EBC Configuration

UCA for EBC can be configured using properties located in configuration files.

The following chapters describe all the properties that can be set to configure UCA for EBC at the application level using configuration files (usually located in the `${UCA_EBC_INSTANCE}/conf/` folder). Additional configuration can be performed at the value pack and scenario level.

NOTE:

For more information on how to configure UCA for EBC at the value pack or scenario level please refer to: [R3] `HPE UCA for EBC User Interface Guide`.

3.1 Multiple instances configuration

Since UCA-EBC V2.0, it is possible to configure multiple instances on a same server. There is a command line tool for managing those instances: uca-ebc-instance.
Please refer to "Chapter 2.2.4 uca-ebc-instance" for more information on how to use this tool.

When creating a new instance of UCA for EBC, the port numbers specified in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file are automatically tuned so that they do not interfere with ports of existing instances of UCA for EBC. They are adjusted based on default port numbers delivered in the `${UCA_EBC_HOME}/defaults/conf/uca-ebc.properties` file.

For example, such ports may have following values (the port numbers in the example below correspond to a 3rd instance of UCA for EBC):

```
uca.ebc.jms.broker.port=61866
uca.ebc.jmx.rmi.port=1300
uca.gui.port=9088
```

However, you have to make sure that the above ports do not conflict with ports used by other applications on your server.

If you have added other ports in your properties (for example for topology extension), please make sure to tune these ports accordingly.

```
uca.ebc.topology.webPort=7675
```

In the same way, the port numbers in the `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` file are automatically tuned.

The Port property for the CHAINSAW appender specified in the `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` file should be different for each instance of UCA for EBC:

```
<param name="Port" value="4745"/>
```

3.2 Configuration files

3.2.1 uca-ebc.properties file configuration

The `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file contains the different properties that can be set for an instance of UCA for EBC Server.

The following tables list the different properties that can be set:

Table 11: Host and Port # properties in the uca-ebc.properties file

Property name	Explanation
uca.ebc.serverhost	**Default value** : localhost This property defines the local host name as used by the JMX (administration) and JMS (alarm Broker) connection bindings. The value 'localhost' is usually enough, but it can be changed to enter the host fully qualified DNS name or an IP address (especially if the server has several IP interfaces), depending on whether UCA for EBC Server should bind to one specific DNS Name/IP Address or all DNS Names/IP Addresses configured on the server.
uca.ebc.jms.broker.port	**Default value** : 61666 The port used by the JMS Broker. The value of this property can be set to an alternate port number in case of port number conflict with another application on your system.
uca.ebc.jmx.rmi.port	**Default value** : 1100 The port used by RMI for JMX connections. The value of this property can be set to an alternate port number in case of port number conflict with another application on your system.
uca.gui.port	**Default value** : 8888 The local port number used by the embedded UCA for EBC User Interface web server. The value of this property can be set to an alternate port number in case of port number conflict with another application on your system. The URL for connecting to the UCA for EBC User interface is the following: `http://<hostname or IP address>:<port #>/uca` <*hostname or IP address*> is the actual hostname (full DNS name) or the IP address of the UCA for EBC Server system. <port #> is the port number for UCA for EBC User Interface set by the uca.gui.port property (By default: 8888 for the default instance of UCA for EBC).

If you change the uca.ebc.serverhost, or uca.ebc.jms.broker.port properties, the UCA for EBC Channel Adapter configuration must be changed accordingly (only if you use OSS Open Mediation as mediation layer). The uca-ebc-ca.properties file of the UCA for EBC Channel Adapter must be checked and changed if required:

```
# UCA EBC Server to connect to
uca.ebc.jms.broker.host=localhost
uca.ebc.jms.broker.port=61666
```

The default location for the uca-ebc-ca.properties file of the UCA for EBC Channel Adapter is the following:

```
/var/<OSS Open Mediation root directory>/containers/instance-0/ips/uca-ebc-ca-3.4/etc/uca-ebc-ca.properties
```

Where:

- <OSS Open Mediation root directory> stands for the OSS Open Mediation installation root directory, which, by default, translates to the /opt/openmediation-71 directory
- instance-0 is the OSS Open Mediation container instance folder name. Depending on your configuration, the container number could be different than 0. If this is the case, please adjust the name of the container instance folder accordingly

For full details on how to change this file, please refer to: [R4] `HPE UCA for EBC Installation Guide`.

Table 12: Web GUI properties in the uca-ebc.properties file

Property name	Explanation
uca.gui.webapp	**Default value**: webapp/uca-expert-ui.war The location of the Web application ARchive file of the UCA for EBC User Interface.
uca.ebc.rest.api	**Default value**: webapp/uca-ebc-rest-api.war This value is by default commented. This is the location of the Web application ARchive file of the UCA for EBC REST Interface. For more information, please refer to: [R1] `HPE UCA for EBC Reference Guide`

Table 13: Collector properties in the uca-ebc.properties file

Property name	Explanation
collector.logger.enabled	**Default value**: false When set to true, collector logging is enabled. All alarms sent by OSS Open Mediation to UCA for EBC and alarms injected into UCA for EBC using the uca-ebc-injector tool, will be logged to a file at the following location: `${UCA_EBC_INSTANCE}/logs/uca-ebc-collector.log`
collector.measurementrate.enabled	**Default value**: false When set to true, event rate measurement is enabled for the UCA for EBC collector component. The collection statistics data are available either through JMX (using the standard Java jconsole or jvisualvm tool for example), the uca-ebc-admin tool, or the UCA for EBC User Interface.
collector.messages.validation	**Default value**: true When set to true, validation of all events (Alarms) coming into UCA for EBC is enabled. Validation errors are reported in the statistics of the Collector both at the Java JMX Console and UCA for EBC User Interface.

	Validation errors can occur when Alarms that do not conform to the UCA for EBC Alarm XML schema are received by UCA for EBC. For more information on the UCA for EBC Alarm XML schema, please refer to: [R1] `HPE UCA for EBC Reference Guide`.

Table 14: UMB Received Events

Property name	Explanation
received.events.logger.enabled	**Default value**: false When set to true, logging is enabled. All events collected by UCA for EBC through the UMB UCA adapter, will be logged to a file at the following location: `${UCA_EBC_INSTANCE}/logs/uca-ebc-received-events.log`

Table 15: Action Manager properties in the uca-ebc.properties file

Property name	Explanation
action.threads	**Default value**: 20 This property defines the size of the thread pool size (in number of threads) of the UCA for EBC Action Manager component. These threads are in charge of processing asynchronous actions. This property can be tuned up/down in case you need more/less threads to process a large/small number of asynchronous actions in parallel.
action.timeout	**Default value**: 60000 (ms) This property defines the default timeout for actions (in milliseconds) processed by the UCA for EBC Action Manager component. If an action exceeds the timeout, then the action fails with a status explanation indicating that a timeout has run out. This default action timeout can be overwritten for any single action by using the `public void setActionTimeout(int actionTimeout)` method of any Action object. The `actionTimeout` parameter is also in milliseconds.

Table 16: Rule Engine logger properties in the uca-ebc.properties file

Property name	Explanation
engine.logger.enabled	**Default value**: false When set to true, scenario-specific Drools engine logging is enabled. This setting affects all scenarios of all value packs. Scenario-specific engine log files are named `logEngine_<scenario name>.log` and are located in the `${UCA_EBC_INSTANCE}/logs` directory. Scenario-specific engine log files contain standard Drools engine log entries specific to a scenario. These log files can be easily displayed in Eclipse IDE using the Audit view, provided you have installed the Drools Eclipse plugin. This view is show by default if you switch to the Drools perspective.
engine.logger.interval	**Default value**: 1000 This property represents the interval (in milliseconds) at which engine log entries are written to the scenario-specific engine log.

The `uca-ebc.properties` file also contains topology related properties. These properties, prefixed either uca.ebc.topology or neo4j, are related to the UCA for EBC Topology Extension product. These properties are described in the UCA for EBC Topology Extension guide.

For more information on how to set these properties to configure the UCA for EBC Topology Extension product, please refer to: [R5] `HPE UCA for EBC Topology Extension Guide`.

The property named **uca.ebc.version** in the `uca-ebc.properties` is no more used by the UCA for EBC Server product V3.4.

NOTE:

UCA for EBC Server must be restarted in order for any change to the `uca-ebc.properties` file to be taken into account.

For non-stop update of some of the properties, you can use the uca-ebc-admin tool, or the JMX interface (with jconsole or jvisualvm).

Please see section 2.2.3 "uca-ebc-admin" for more information on the list of properties that can be reloaded using the uca-ebc-admin command-line tool.

Please see section 5.1.3 "JMX Console" for more information on the list of properties that can be updated at the Java JMX Console.

3.2.2 ActionRegistry.xml file configuration

UCA for EBC value pack scenarios have the ability to send action requests to be executed by the mediation layer associated with the UCA for EBC Server:

- Either OSS Open Mediation V7.2
- Or UMB V1.1

Whether the actions are sent to OSS Open Mediation or to UMB depends on the action reference used by the action. Since UCA for EBC V3.3 there are two types of action references that can be defined in the `ActionRegistry.xml` file:

- OSS Open Mediation action references
- UMB action references
- If an action is created based on an OSS Open Mediation action reference then it will be executed on OSS Open Mediation. On the other hand, if the action is created based on an UMB action reference then it will be executed on UMB.
- The next chapters will describe how to create both OSS Open Mediation and UMB action references.

NOTE: In order for UMB actions to be used, it is necessary that the embedded UCA for EBC UMB Adapter be started, i.e. the `use.new.generation.adapter` property must be set to true in the `uca-ebc.properties` file.

The actions are executed by an OSS Open Mediation Channel Adapter (or UMB Adapter) on the mediation layer. Action replies are then returned to the scenario that sent the action requests.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ActionRegistryXML xmlns="http://registry.action.mediation.uca.hp.com/">
  <!-- ActionReferences must be unique across this file-->

  <!-- NOM Actions -->
  <MediationValuePack MvpName="temip" MvpVersion="2.2.0"
    url="http://localhost:26700/uca/mediation/action/ActionService?WSDL"
    brokerURL="failover://tcp://localhost:10000">

    <Action actionReference="TeMIP_AO_Directives_localhostNOM">
      <ServiceName>aoDirective</ServiceName>
      <NmsName>localTeMIP</NmsName>
    </Action>
    <Action actionReference="TeMIP_TT_Directives_localhostNOM">
      <ServiceName>ttDirective</ServiceName>
      <NmsName>localTeMIP</NmsName>
    </Action>
    <Action actionReference="TeMIP_FlowManagement">
      <ServiceName>subscriptionManagement</ServiceName>
      <NmsName>localTeMIP</NmsName>
    </Action>
  </MediationValuePack>

  <MediationValuePack MvpName="exec" MvpVersion="2.0.0"
    url="http://localhost:26700/uca/mediation/action/ActionService?WSDL"
    brokerURL="failover://tcp://localhost:10000">

    <Action actionReference="Exec_localhost">
      <ServiceName>commandsExecution</ServiceName>
      <NmsName>localhost</NmsName>
    </Action>
  </MediationValuePack>

  <!-- UMB Actions -->
  <UMBActions>
    <UMBAction actionReference="TeMIP_AO_Directives_localhost" targetName="TeMIP" targetActionName="AOAction"/>
    <UMBAction actionReference="TeMIP_TT_Directives_localhost" targetName="TeMIP" targetActionName="TTAction"/>
    <UMBAction actionReference="TeMIP_Passthrough_Directives_localhost" targetName="TeMIP"
               targetActionName="PassthroughAction"/>
  </UMBActions>
```

Figure 1: ActionRegistry.xml file

It is important to notice that there are 2 sections in the `ActionRegistry.xml` file. A first section for OSS Open Mediation action, and a second section for UMB actions. Action references can be defined in both sections but the name of action references throughout the whole file must be unique: an OSS Open Mediation action reference cannot have the same name as an UMB action reference.

The default configuration for this file can be retrieved from the *`${UCA_EBC_HOME}`*`/defaults/conf` folder in case you want to revert back to the default configuration.

The *`ActionRegistry.xml`* file is an UCA for EBC application instance level configuration file. It is shared by all UCA for EBC value packs running on the same UCA for EBC Server instance.

NOTE: UCA for EBC Server must be restarted in order for any change to the `ActionRegistry.xml` file to be taken into account, unless you use the Java JMX Console to refresh the UCA for EBC Action Manager with the contents of the `ActionRegistry.xml` file.

Please see 5.1.3.1 "Monitoring UCA for EBC internal components" to learn how to refresh the UCA for EBC Action Manager with the contents of the `ActionRegistry.xml` file using the Java JMX Console.

3.2.2.1 Defining OSS Open Mediation action references

UCA for EBC value pack scenarios use web services to communicate with the Action Service web service of a Channel Adapter, typically the UCA for EBC Channel Adapter.

For these actions to be properly routed to the mediation layer and then to the correct Channel Adapter and target application, the file `${UCA_EBC_INSTANCE}/conf/ActionRegistry.xml` must be configured correctly.

The OSS Open Mediation section of the `ActionRegistry.xml` file defines "mediation value packs", and "action references" for these mediation value packs. The following sections will describe in detail how to configure the `ActionRegistry.xml` file in terms of "mediation value packs", and "action references".

3.2.2.1.1 Defining (OSS Open Mediation) Mediation Value Packs

Each "mediation value pack" defined in the `ActionRegistry.xml` file describes the properties of a gateway to access the Action Service web service on a UCA for EBC Channel Adapter deployed on OSS Open Mediation V7.2.

This gateway will be able to process action requests on the mediation layer by forwarding the action requests to the proper Channel Adapter on OSS Open Mediation V7.2 for processing.

Each OSS Open Mediation "mediation value packs" is defined by an `<MediationValuePack ...>...</MediationValuePack>` XML element.

Each `<MediationValuePack ...>...</MediationValuePack>` XML element defined in the `ActionRegistry.xml` file has the following attributes:

- MvpName: You can give any value to this property (the value is not bound to anything). However, it is recommended to use the name of the Channel Adapter that will be targeted by the action requests. For example:
  - "temip" (as in TeMIP Channel Adapter) or
  - "exec" (as in Exec Channel Adapter)
- MvpVersion: You can give any value to this property (the value is not bound to anything). However, it is recommended to use the version of the Channel Adapter that will be targeted by the action requests. For example:
  - or
  - 2.1 or
  - etc...
- brokerURI: This property contains the correct URI for connecting to the JMS Broker of the OSS Open Mediation V7.2 container instance that contains a UCA for EBC Channel Adapter. By default the port number of the JMS Broker of OSS Open Mediation V7.2 container 0 is 10000. To verify what port number is used for your OSS Open Mediation V7.2 container instance, please check the value of the activemq.port property in the `/var/opt/openmediation-V72/containers/instance-<instance number>/conf/servicemix.properties` file.

JMS Broker URIs have the following pattern:

`tcp://<`*`hostname or IPaddress`***`>:<`***`port#`***`>`** or

`failover://tcp://<`*`hostname or IPaddress`***`>:<`***`port#`***`>`** for the failover URI

where:

`<hostname or IP address>` is the actual hostname (full DNS name) or the IP address of the OSS Open Mediation V7.2 system.

`<port #>` is the port number of the JMS Broker of the OSS Open Mediation V7.2 container instance that contains a UCA for EBC Channel Adapter. The default port # is 10000 for container instance 0.

The brokerURI property is used to connect to the Alarms JMS topic of the UCA for EBC Channel Adapter when using the standard UCA for EBC `OpenMediationAlarmForwarder` Java class for forwarding alarms to OSS Open Mediation V7.2.

For more information on how to forward alarms, please refer to [R2] `HPE UCA for EBC Value Pack Development Guide`.

- url: This property contains the correct URL for connecting to the Action Service web service on a UCA for EBC Channel Adapter. For example, if the UCA for EBC Channel Adapter is on localhost and uses the default port number for its Action Service web service:

  http://localhost:26700/uca/mediation/action/ActionService?WSDL

  An incorrect value for the **url** property will result in action requests not being able to be processed on the mediation layer. Please verify this **url** using a web browser before using it in the `ActionRegistry.xml` file.

NOTE:

Action Service web service URLs have the following pattern:

http://<*hostname or IPaddress*>:<*port#*>/uca/mediation/action/ActionService?WSDL

<hostname or IP address> is the actual hostname (full DNS name) or the IP address of the UCA for EBC CA system.

<port #> is the port number for UCA for EBC CA Action Service, 26700 by default. This port number is set in the *<OSS Open Mediation variable root directory>/containers/instance-<container instance number>/ips/uca-ebc-ca-<UCA for EBC CA version>/etc/action-service.xml* file (see the value of the locationURI property of the cxfbc:consumer XML entity).

<OSS Open Mediation variable root directory> usually translates to `/var/opt/openmediation-V7.2`.

Two mediation value packs are defined by default in the `ActionRegistry.xml` file:

- A "`temip`" services mediation value pack, providing a gateway to a TeMIP Channel Adapter for executing TeMIP Alarm Object directives, TeMIP Trouble Ticket directives, and alarm collection flow creation/deletion/resynchronization
- An "`exec`" services mediation value pack, providing a gateway to an Exec Channel Adapter for executing command-line interface executables/commands

Each mediation value pack can contain one or more action references. Action references are explained in the next section.

3.2.2.1.2 Defining (OSS Open Mediation) Action References

OSS Open Mediation action references define references to be used in the Drools rules files (or in the Java code executed by the rules) associated to scenarios of UCA for EBC value pack for executing synchronous/asynchronous action on products (TeMIP for example) connected to OSS Open Mediation V7.2 via their own Channel Adapter.

Below is an example of how action references can be used in UCA-EBC Value Pack code (we assume in this example that an action reference called "`TeMIP_AO_Directives_localhostNOM`" has been defined in the `ActionRegistry.xml` file)

Basically you need to write the following code in your rules file:

```
Action action = new Action("TeMIP_AO_Directives_localhostNOM");
```

The action reference called "TeMIP_AO_Directives_localhostNOM" is used when creating an Action Java Object in the rules files.

Once an Action object is created, you can specify the parameters that will define what action to perform, in the following example a TeMIP Alarm Object directive:

```
action.addCommand("directiveName", "ACKNOWLEDGE");
action.addCommand("entityName", a.getIdentifier());
action.addCommand("UserId", "UCA EBC");
```

Then, you need to execute the Action. Both synchronous and asynchronous actions are possible:

either:

```
//synchronous execution
action.executeSync();
```

or:

```
//asynchronous execution
action.executeAsync(AODirectiveKey.ENTITY_NAME);
```

CAUTION: A synchronous Action may block the on-going correlation processing as long as the action is not achieved.

For more information on synchronous and asynchronous actions (including how to use synchronization keys for asynchronous actions), please refer to [R1] HPE UCA for EBC Reference Guide.

OSS Open Mediation action references are defined in the ActionRegistry.xml file inside a <MediationValuePack>…</MediationValuePack> section. Each OSS Open Mediation action reference is defined by an <Action>…</Action> XML element.

Each <Action>…</Action> XML element defined in the ActionRegistry.xml file has the following properties:

- actionReference: this is the name of the action reference to use in the Drools rules files associated with scenarios of UCA for EBC value pack

An incorrect value for the actionReference property will result in action requests not being able to be processed on the mediation layer. If that happens, please check uca-ebc.log file for errors and verify that value of the actionReference property is in line with the action reference used in the Drools rules files (or in the Java code executed by the rules) of the scenarios of your UCA for EBC value pack(s).

Each <Action>…</Action> XML element defined in the ActionRegistry.xml file also defines the following sub-elements:

serviceName: this is a valid name of service (type of action) implemented by the target Channel Adapter (TeMIP CA, Exec CA, etc.). This service name is determined by the target Channel Adapter and the services it provides. For example:

- The TeMIP Channel Adapter provides the following services:
  - aoDirective, for executing Alarm Object (AO) directives
  - ttDirective, for executing Trouble Ticket (TT) directives

        - `subscriptionManagement`, for executing alarm collection flow creation/deletion/resynchronization
    - The Exec Channel Adapter provides the following services:
        - `commandsExecution`, for executing command-line interface executables/commands

    An incorrect value for the **serviceName** property will result in action requests not being able to be processed on the mediation layer. Please verify that value of the **serviceName** property is valid for the target Channel Adapter by reviewing the target Channel Adapter documentation.

NmsName: hostname or IP address of the system targeted by the target Channel Adapter. This property is used for information only. It is not bound to anything.

3.2.2.2 Defining UMB action references

From UCA for EBC point of view, UMB action references work exactly the same way as OSS Open Mediation action references.

UMB action references define references to be used in the Drools rules files (or in the Java code executed by the rules) associated to scenarios of UCA for EBC value pack for executing synchronous/asynchronous action on products (TeMIP for example) connected to UMB V1.1 via their own UMB Adapter.

Below is an example of how action references can be used in rules files (we assume in this example that an action reference called "`TeMIP_AO_Directives_localhost`" has been defined in the `ActionRegistry.xml` file)

Basically you need to write the following code in your rules file:

```
Action action = new Action("TeMIP_AO_Directives_localhost");
```

The action reference called "`TeMIP_AO_Directives_localhost`" is used when creating an Action Java Object in the rules files.

Once an Action object is created, you can specify the parameters that will define what action to perform, in the following example a TeMIP Alarm Object directive:

```
action.addCommand("directiveName", "ACKNOWLEDGE");
action.addCommand("entityName", a.getIdentifier());
action.addCommand("UserId", "UCA EBC");
```

Then, you need to execute the Action. Both synchronous and asynchronous actions are possible:

either:

```
//synchronous execution
action.executeSync();
```

or:

```
//asynchronous execution
action.executeAsync(AODirectiveKey.ENTITY_NAME);
```

CAUTION: A synchronous Action may block the on-going correlation processing as long as the action is not achieved.

For more information on synchronous and asynchronous actions (including how to use synchronization keys for asynchronous actions), please refer to [R1] `HPE UCA for EBC Reference Guide`.

UMB action references are defined in the `ActionRegistry.xml` file inside the `<UMBActions>…</UMBActions>` section. Each UMB action reference is defined by an `<UMBAction … />` XML element.

Each `<UMBAction … />` XML element defined in the `ActionRegistry.xml` file has the following properties:

- actionReference: this is the name of the action reference to use in the Drools rules files associated with scenarios of UCA for EBC value pack

  An incorrect value for the **actionReference** property will result in action requests not being able to be processed on the mediation layer. If that happens, please check uca-ebc.log file for errors and verify that value of the **actionReference** property is in line with the action reference used in the Drools rules files (or in the Java code executed by the rules) of the scenarios of your UCA for EBC value pack(s).

- targetName: this is the name of the target UMB Adapter or UMB Adapter Group of the action: either the name of an UMB Adapter or the name of an UMB Adapter action group (if load balancing is used). Adapter names and action group names are defined in the `AdapterConfiguration.xml` file of UMB Adapters. If an UMB Adapter name is used, then the action will be executed on a specific UMB Adapter. If an UMB Adapter Group name is used, then the action will be executed on a randomly selected UMB Adapter part of the group.
- targetActionName: this is the name of the type of action (action service) to be executed on the UMB Adapter or UMB Adapter Group. Action services are specific to each Adapter. They are defined in the `AdapterConfiguration.xml` file of each Adapter. For example, the UMB TeMIP Adapter defines the following action services:
  - `AOAction`, for executing Alarm Object (AO) directives
  - `TTAction`, for executing Trouble Ticket (TT) directives
  - `PassthroughAction`, for executing any directive. The XML of the request must be provided in TWS XML schema in the rawData of the action request

An incorrect value for either the targetName or the targetActionName property will result in action requests not being able to be processed on the mediation layer. Please verify that values of the **targetName** and **targetActionName** properties are valid by reviewing the `AdapterConfiguration.xml` file of the targeted UMB Adapter(s)

3.2.3 uca-ebc-log4j.xml file configuration

The *`${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml`* file is the Log4J configuration file for the whole UCA for EBC instance. It is a standard Apache Log4J configuration file. ([1])

This file contains three main sections where the following items are defined:

- Appenders: appenders mainly define where the log messages are sent, and the pattern used for logging the messages. There are three main appenders defined.
  - CONSOLE: for logging to the console
  - FILE: for logging to the ${UCA_EBC_INSTANCE}/logs/uca-ebc.log file
  - DB: for logging to a database. This log database is used for displaying the logs on the UCA for EBC User Interface. ([2])

In addition to the three main appenders, a sample CHAINSAW appender is also provided for integration with the Apache Chainsaw GUI-based log viewer. ([3])

- Loggers: loggers are defined by Java package names. Each logger defines its own log level and appender references. The loggers are grouped into several sections (the different sections are identified by comments in the file):
    - Detailed Traces for UCA Scenarios
    - Detailed Traces for UCA Components
    - Detailed Traces for UCA ClassLoader
    - Third Party Products Internals
- Root: the root section defines the default log level and the default appender references to use for logging

You can make your own changes to the *${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml* file, for example:

- Modifying existing appenders or creating new ones
- Modifying existing loggers: changing the log level or the appender references
- Adding new loggers, for 3rd party products for example
- Adding new loggers for your own scenarios
- Modifying the default log level and appender references in the root section of the file

Once you have made changes to the *${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml* file, you either need to restart UCA for EBC Server or reload the Log4J configuration at the command-line using the uca-ebc-admin tool, the Java console or the UCA for EBC User Interface.

There are several levels of logging provided by UCA for EBC ([4]):

- standard application logging
- scenario specific rule logging.

Log files (both standard application log file, and scenario specific log files) are stored in the `${UCA_EBC_INSTANCE}/logs` directory or at the UCA for EBC User Interface.

NOTE:

([1]) Please see http://logging.apache.org/log4j/1.2/ to learn more about Apache Log4J configuration files.

([2]) Please note that for big size of the log the correlation can dramatically slow down due to the cost of every insertion into the database.

([3]) Please see http://logging.apache.org/chainsaw/index.html to learn more about Apache Chainsaw.

([4]) Please see 6.1 "UCA for EBC Logging Mechanism" to learn about the different levels of logging provided by UCA for EBC (standard application logging, and scenario specific rule logging) and to learn how to enable/disable and configure these logs

3.2.4 Additional configuration files

Some configuration files are present in addition to the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties`, `${UCA_EBC_INSTANCE}/conf/ActionRegistry.xml`, and `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` files.

3.2.4.1 UCA EBC Spring Framework configuration files

UCA for EBC is integrated with Spring Framework. The main components of UCA for EBC are defined using Spring Framework. Three configuration files located in the `${UCA_EBC_HOME}/conf` directory are present by default:

- `application-context.xml`
- `main-context.xml`

These files are for INTERNAL USE ONLY and should not be modified.

However, in rare case, for instance when you need to support storing events into a single DB for multiple Value Packs, you may add some Spring beans in those files.

3.2.5 How to revert back to the default configuration files

A reference copy of of each of the configuration files present in the `${UCA_EBC_INSTANCE}/conf` folder can be found in the `${UCA_EBC_HOME}/defaults/conf` folder.

In case you want to revert back the default configuration of any of the configuration files present in the `${UCA_EBC_INSTANCE}/conf` folder, you just need to copy the reference copy of the configuration file from the `${UCA_EBC_HOME}/defaults/conf` folder to the `${UCA_EBC_INSTANCE}/conf` folder.

NOTE:

UCA for EBC Server must be restarted in order for any change to the configuration files in the `${UCA_EBC_INSTANCE}/conf` folder to be taken into account (except for the uca-ebc-log4j.xml file for which a dynamic reload is possible)

3.3 UCA-EBC UMB Mediation Adapter Configuration

The configuration requirements of the UCA-EBC UMB Mediation Adapter are the same as for any other UMB mediation adapter. It requires a properties file to set the Adapter properties, the `Hazelcast.xml` file for the Common registry access, and the AdapterConfiguration.xml file to define the provided services.

All the requested configuration files are searched in the UCA-EBC configuration directory: `${UCA_EBC_DATA}/conf`.

3.3.1 The properties file

As the UCA-EBC UMB Mediation Adapter is embedded in the UCA-EBC application, there is no specific `adapter.properties` file for this mediation adapter. Instead the properties are defined in the standard `uca-ebc.properties` file.

The following properties are defined by default in this file as follow:

```
################################################################
#                      UMB Mediation properties
use.new.generation.adapter=true

# UMB Consumer properties
consumer.zookeeper.connect=localhost:2181
consumer.zookeeper.session.timeout.ms=6000
consumer.zookeeper.sync.time.ms=203
```

```
consumer.auto.commit.interval.ms=1000
consumer.auto.offset.reset=smallest

# UMB Consumer properties
producer.metadata.broker.list=localhost:9092
producer.request.required.acks=1
##################################################################
```

Please refer to the [R4] `HPE UCA for EBC Installation Guide` for details on how to configure the Adapter's properties.

3.3.2 The hazelcast.xml file

The Adapter's Hazelcast configuration file: `hazelcast.xml` defines how to connect to the UCA for EBC Hazelcast instance(s).

Please refer to the [R4] `HPE UCA for EBC Installation Guide` for details on how to configure the `hazelcast.xml` file.

3.3.3 The logging configuration file

The Adapter's Log4j configuration is done through the standard UCA-EBC configuration file: `uca-ebc-log4j.xml`

3.3.4 The AdapterConfiguration.xml file

The Adapter configuration file `AdapterConfiguration.xml` defines the Event flows that UCA-EBC provides.

For every UMB Adapter the `AdapterConfiguration.xml` file defines the adapter name (by default set to "UCA-EBC"). This name must be changed if the solution is made of several UCA-EBC server instances.

3.3.4.1 Defining static flows:

For static Flows the collectorClass must be set to: com.hp.uca.expert.mediation.adapter.UcaStaticCollector

No flow parameters need to be defined.

Here is an example of Static Flow Service definitions for UCA-EBC:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<adapter name="UCA-EBC" version="1.0" xmlns="http://hp.com/umb/config">
    <flowServices>
        <flow name="UcaStaticForwarderFlow" type="Static"
collectorClass="com.hp.uca.expert.mediation.adapter.UcaStaticCollector">
        </flow>
        <flow name="UcaStaticEventForwarderFlow" type="Static"
collectorClass="com.hp.uca.expert.mediation.adapter.UcaStaticCollector">
        </flow>
    </flowServices>
</adapter>
```

NOTE:

The static flows provided by UCA-EBC do not support resynchronization.

3.3.4.2 Defining dynamic flows:

Contrary to static flows, with dynamic flows, parameters have to be defined.

Here is an example of a Dynamic Flow Service definition for UCA-EBC:

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

<adapter name="UCA-EBC" version="1.0" xmlns="http://hp.com/umb/config">
  <flowServices>
    <flow name="DB-Flow" type="Dynamic"

collectorClass="com.hp.uca.expert.mediation.adapter.UcaDbCollector">
      <parameters>
       <parameter key="vp" defaultValue="" />
       <parameter key="notifier" defaultValue="dbNotifier" />
       <parameter key="summarize" defaultValue="false" />
       <parameter key="eligibilityScope" defaultValue="true" />
       </parameters>
    </flow>
  </flowServices>
```

For more information on DB Flow configuration, please refer to: [R1] `HPE UCA for EBC Reference Guide`

3.4 High-Availability (HA) configuration

3.4.1 Simple cluster configuration using NFS

The simplest cluster configuration is a set of (minimum 2) members UCA for EBC servers accessing the same Storage Area Network providing access to a single data storage.

To setup such a cluster configuration, the following steps are required:

- Step 1: Install UCA for EBC using the -d option to specify the same "data" directory. See Note ([1])

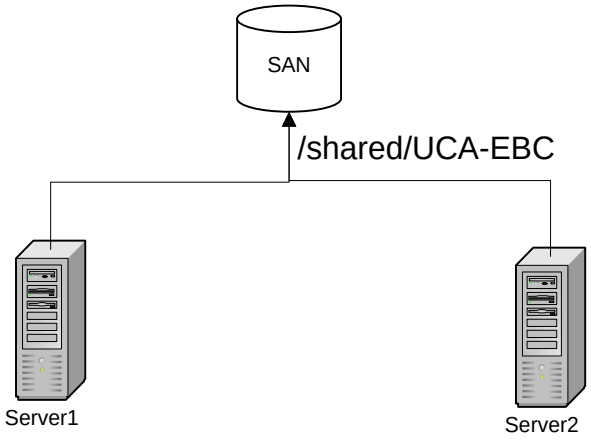
  For example, given that `/shared/UCA-EBC` is the NFS mount point for the UCA for EBC data directory, you need to execute the following command on all servers:

  ```
  [root] # install-uca-ebc.sh -d /shared/UCA-EBC
  ```

  On first installation of UCA for EBC (on server1), the subdirectories under `/shared/UCA-EBC` are automatically created. On subsequent installations (on server2 and +), the subdirectories are not recreated because they already exist. Using this method, you can install an extra server even if UCA for EBC is running on another server.

- Step 2: Start UCA for EBC on the first server. See Note ([2])

  ```
  [uca@server1] # uca-ebc start
  ```



- Step 3: When server1 is to be stopped for some reason, then server2 is able to recover the work, once started.

```
[uca@server2] # uca-ebc start
```

NOTE:

([1]) It is mandatory that the "uca" user account used to run UCA for EBC should have the same uid / gid on all the servers sharing a same data directory. If this is not the case, UCA for EBC will not be able to recover from one server to the other due to file ownership issues. It is therefore recommended to use a NIS user account across servers.

([2]) Log and work files are stored in a shared NFS data storage. It is not supported to have more than 1 UCA for EBC server instance running on the same data storage due to possible file synchronization issues.

([3]) For more information on High-Availability setup, please refer to: [R1] `HPE UCA for EBC Reference Guide`

3.4.2 Neo4j database High-Availability (HA) configuration for Topology Extension

The simplest configuration of neo4j is to have the database server embedded in UCA for EBC server. As such, it can run only on a single machine, accessible through a single port. When configured as embedded, the database is stored under the `${UCA_EBC_INSTANCE}/neo4j` directory.

When a simple cluster configuration is used along with an embedded neo4j topology, the High-Availability (HA) mechanism is implemented by the shared location of the `${UCA_EBC_INSTANCE}` directory which includes the neo4j database. When a member of the cluster starts, it inherits the neo4j database state, i.e. the topology state, from the last cluster member that stopped.

This solution does not use the HA mechanism of neo4j. (See Note ([1]) below).

To deploy the UCA for EBC database, i.e. the neo4j database, in a multiple machine setup, you have to tune the **uca.ebc.topology** property in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file, as follows:

```
uca.ebc.topology=external
```

This property is set by default to "embedded" but it needs to be changed to "external" for HA configuration. (See Note ([1]) below)

Neo4j HA can be set up to accommodate differing requirements for load, fault tolerance and available hardware. The typical setup when running multiple Neo4j instances in HA mode is: (See Note ([2]) below)

- A HTTP REST load-balancer, namely HA proxy
- a single Neo4j master
- 0 or more Neo4j slaves
- a mechanism for master election, namely a Coordinator cluster (See Note ([3]) below)

To configure UCA for EBC to use a Neo4j HA cluster, you need to setup the **uca.ebc.topology.serverhost** and **uca.ebc.topology.webPort** properties in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file to be equal to the Neo4J HA proxy configuration. For example:

```
uca.ebc.topology.serverhost=server3.local.domain
uca.ebc.topology.webPort=7474
```

Then, you have to configure the Neo4j cluster to run in HA mode. Please refer to the Neo4j high-availability setup tutorial for more information. (See Note ([4]) below)

NOTE:

([1]) The "embedded" value for the **uca.ebc.topology** property in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file does not currently support the neo4j HA mode.

([2]) Suggested reading: http://docs.neo4j.org/chunked/stable/ha.html. Please note that only neo4j-enterprise edition supports HA features.

([3]) The Coordinator function is based on Apache Zookeeper service: http://hadoop.apache.org/zookeeper/

([4]) The Neo4j high-availability setup tutorial is available at the following URL: http://docs.neo4j.org/chunked/stable/ha-setup-tutorial.html

3.5 Backup and restore

3.5.1 Standalone UCA for EBC

A standalone UCA for EBC server is a server running on a single machine. If the UCA for EBC Topology Extension is installed and configured, the neo4j server is running embedded within UCA for EBC Server. (See Note below)

On both HP-UX and Linux:

To perform a backup/restore, please use the ***uca-ebc-backup*** command line tool (Please refer to Chapter 2.2.5" uca-ebc-backup" for command usage).

On Windows:

To perform a backup/restore, as no command line tool is provided, please use the following procedure:

For backups:

- `cd %UCA_EBC_DATA%`
- zip all directories (except logs and work) into a backup .zip file, and store it in a safe place

For restores: (Please make sure that UCA for EBC is not running)

- `cd %UCA_EBC_DATA%`
- remove all directories (except logs and work)
- unzip the backup .zip file that was created during the backup

NOTE:

neo4j embedded server online backup feature must be activated.

This is done by setting the **neo4j.config.online_backup_enabled** property to **true** in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` configuration file.

3.5.2 Clustered UCA for EBC

A clustered UCA for EBC server is a set of multiple servers running on separate machines but using the same data directory under NFS. This is described in Chapter 3.4.1 "Simple cluster configuration using NFS".

As data is stored on a unique place, it is only necessary to perform the backup once for the cluster, on any machine. To perform a backup/restore, please use the procedure explained above (in Chapter 3.5.1 "Standalone UCA for EBC") which is applicable in a clustered context as well.

3.5.3 UCA for EBC with external topology server

A UCA for EBC server using an external neo4j topology server has to be backed up (or restored) in two steps.

3.5.3.1 First step: backup/restore of UCA for EBC

To backup/restore UCA for EBC, use the procedure explained in Chapter 3.5.1 "Standalone UCA for EBC" above. This procedure will back up everything that is stored in the UCA for EBC instance directory, except the neo4j database, which is external.

3.5.3.2 Second step: backup/restore of neo4j database

When neo4j server is configured to be external to UCA for EBC, it is necessary to backup/restore this external machine separately. Please be aware that the neo4j backup utility is only available when using the Enterprise Edition of Neo4j.

When neo4j database is embedded into UCA for EBC server, please follow the steps described below to perform a backup/restore of the neo4j database.

For backups:

- Do a full backup using the neo4j-backup command line tool on a safe new directory (☞ see Note ([1]) below)

For restores:

- Restore the backup by replacing the current database (usually stored in `${NEO4J_HOME}/data/graph.db`) by the contents of the directory generated during the backup.

NOTE:

([1]) neo4j embedded server online backup feature must be activated.
This is done by setting the neo4j.config.online_backup_enabled property to true in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` configuration file

Chapter 4
UCA for EBC Monitoring

4.1 Monitoring the alarm flow in real-time

The purpose of monitoring the alarm flow is to offer any integrator and/or rules designer (at development time) or any user (in production) the capability to better understand what happens in the UCA for EBC engine (in particular in each rule engine/working memory of a scenario).

A UCA for EBC solution can be complex including several values packs, each of them containing several scenarios. At each level, filtering at the scenario level indicates the scope of interest of the scenario, in terms of what type of events the scenario will process.

Monitoring the alarm flow is key to a better understanding of what goes on inside UCA for EBC in terms of processing of the alarm flow in real-time, when a complete UCA for EBC solution, with possibly several value packs and scenarios, is deployed.

Monitoring the alarm flow involves taking measurements of the alarm flow at several key processing points in the UCA for EBC solution:

- At the UCA for EBC Collector layer, i.e. alarm collection layer (this component is the entry point for alarms/events into UCA for EBC)
- At the UCA for EBC Dispatcher layer, i.e. alarm dispatcher layer (this component processes alarms/events sent by the UCA for EBC Collector and dispatches them to value packs and scenarios)
- At the Value Pack layer
- At the Scenario layer, i.e. the Drools engine layer

The following figure explains the "points-of-control" where measurements of the alarm flow are performed:

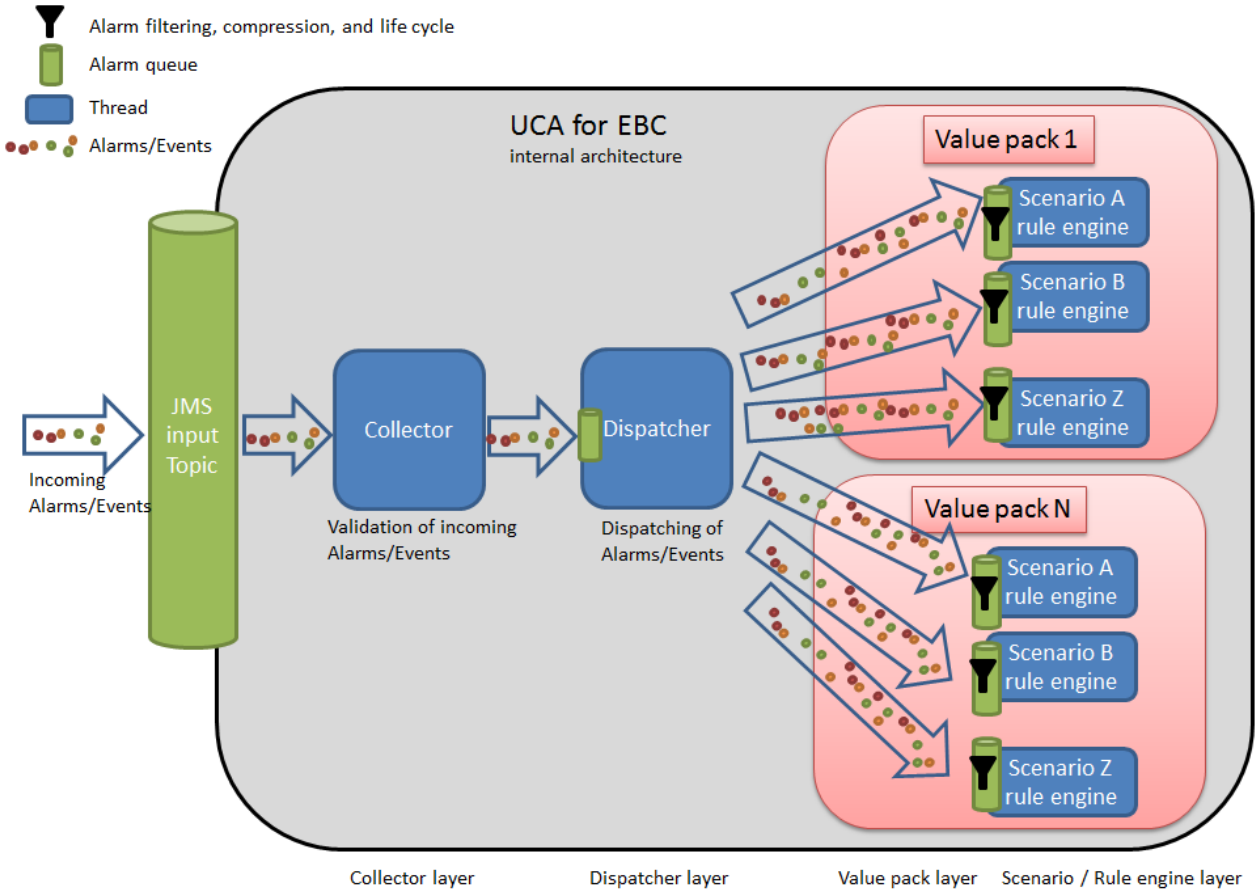


Figure 2: UCA for EBC – Monitoring the Alarm Flow

Monitoring of the alarm flow is performed at the Collector layer, Dispatcher layer, Value Pack layer and Scenario / Rule engine layer is shown in the above figure.

These measurements of the alarm flow are presented as statistics, and counters, and can be displayed with the Java JMX Console, the uca-ebc-admin command line tool and through the UCA for EBC User Interface (in the Troubleshooting / Statistics panel).

The following sections describe, for each layer of the UCA for EBC product, the different 'points-of-control' where statistics about the alarm flow are available. These statistics can help developers and integrators better understand how scenarios consume the input Event/Alarm stream. Monitoring these statistics can provide insight into the internal processing of a scenario in real time that can help troubleshooting issues or possibly lead to improvements in terms of performance.

NOTE: For more information on the UCA for EBC User Interface, please refer to [R3] `HPE UCA for EBC User Interface Guide`

Please see section 5.1.3 "JMX Console" for more information on the statistics, and counters displayed at the Java JMX Console

4.1.1 Collector layer

The Collector component is responsible for receiving and validating incoming Events/Alarms from the mediation layer (when the mediation layer is OSS Open Mediation V7.2) and forwarding them to the next layer (the Dispatcher layer). The following indicators can help monitoring the alarm flow at the Collector layer in real-time:

- How many objects (alarms) were received since startup
- The last time an object (alarm) was received
- How many errors occurred during alarm validation
- The last time an error occurred during alarm validation

NOTE: These statistics can be displayed both at the Java JMX Console and at the UCA for EBC User Interface (in the Troubleshooting / Statistics panel)

For more information on the UCA for EBC User Interface, please refer to: [R3] `HPE UCA for EBC User Interface Guide`

4.1.2 Dispatch layer

The Dispatcher is responsible for storing incoming events (Alarms), analyzing and dispatching these events to the running value packs and scenarios. The following indicators can help monitoring the alarm flow at the Dispatcher layer in real-time:

- Current number of objects (alarms) dispatched
- Last time an object (alarms) has been dispatched
- Rate of alarms reception

NOTE: These statistics can be displayed both at the Java JMX Console and at the UCA for EBC User Interface (See 5.1.2 "UCA for EBC Graphical User Interface")

4.1.3 Value Pack layer

Additional statistics regarding the alarm flow are available at the Value Pack layer:

- How many objects (alarms) were received since startup (per alarm type)
- Last time an object (alarm) was received
- Alarm input rate
- Percentage of events received by the Value Pack compared to the total of events received by the UCA for EBC Dispatcher

NOTE: These statistics can be displayed both at the Java JMX Console and at the UCA for EBC User Interface (See 5.1.2 "UCA for EBC Graphical User Interface")

4.1.4 Scenario/Engine layer

Additional statistics regarding the alarm flow are available at the Scenario (Drools engine) layer:

- Number of facts* inserted into Working Memory since startup
- Current number of facts* in real-time
- Last time an object (alarm) was injected, retracted, modified in Working Memory
- Number of facts* retracted from the Working Memory since start-up
- Number of facts* modified in Working Memory since start-up
- Rate of alarms reception
- Percentage of events inserted into Working Memory compared to the total of events received by the Scenario (this indicator measures what percentage of incoming events are filtered out by the scenario)

* Facts are Drools Working Memory objects. Once any Java object is inserted into Drools Working Memory, it becomes a "Fact".

NOTE: These statistics can be displayed both at the Java JMX Console and at the UCA for EBC User Interface (See 5.1.2 "UCA for EBC Graphical User Interface")

Chapter 5
UCA for EBC Troubleshooting

5.1 Troubleshooting tools

Below is the list of tools that you can use to troubleshoot UCA for EBC.

5.1.1 Log files

Log files can be of great help when troubleshooting issues with UCA for EBC. UCA for EBC log files are located in the `${UCA_EBC_INSTANCE}/logs` directory.

You can view the log files directly on the file system using any text file editor or you could also use the UCA for EBC User Interface to view the logs. This latter method for viewing the logs has the advantage of providing easy navigation and filtering capabilities. The UCA for EBC application log can also be cleaned to focus on new log messages only.

Configuration of the logs is driven by the content of the `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` file ([1]). Several types of logs are available, both at application level and at scenario level ([2]).

NOTE:

Please refer to section 3.2.3 "uca-ebc-log4j.xml file configuration" to learn more about the configuration of the `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` file.

Please see section 6.1 "UCA for EBC Logging Mechanism" to learn about the different levels of logging provided by UCA for EBC, how to enable/disable and how to configure these logs.

Recommendation: logging has an impact on performance. To avoid issues, please do not use too much logging on a production environment.

5.1.2 UCA for EBC Graphical User Interface

The UCA for EBC User Interface provides troubleshooting tools.

At each level, be it application level, value pack level or scenario level, a troubleshooting panel is provided that contains information that will help to troubleshoot issues with the UCA for EBC application, a specific value pack or a scenario.

The following screenshot shows Troubleshooting/Log panel at application level:

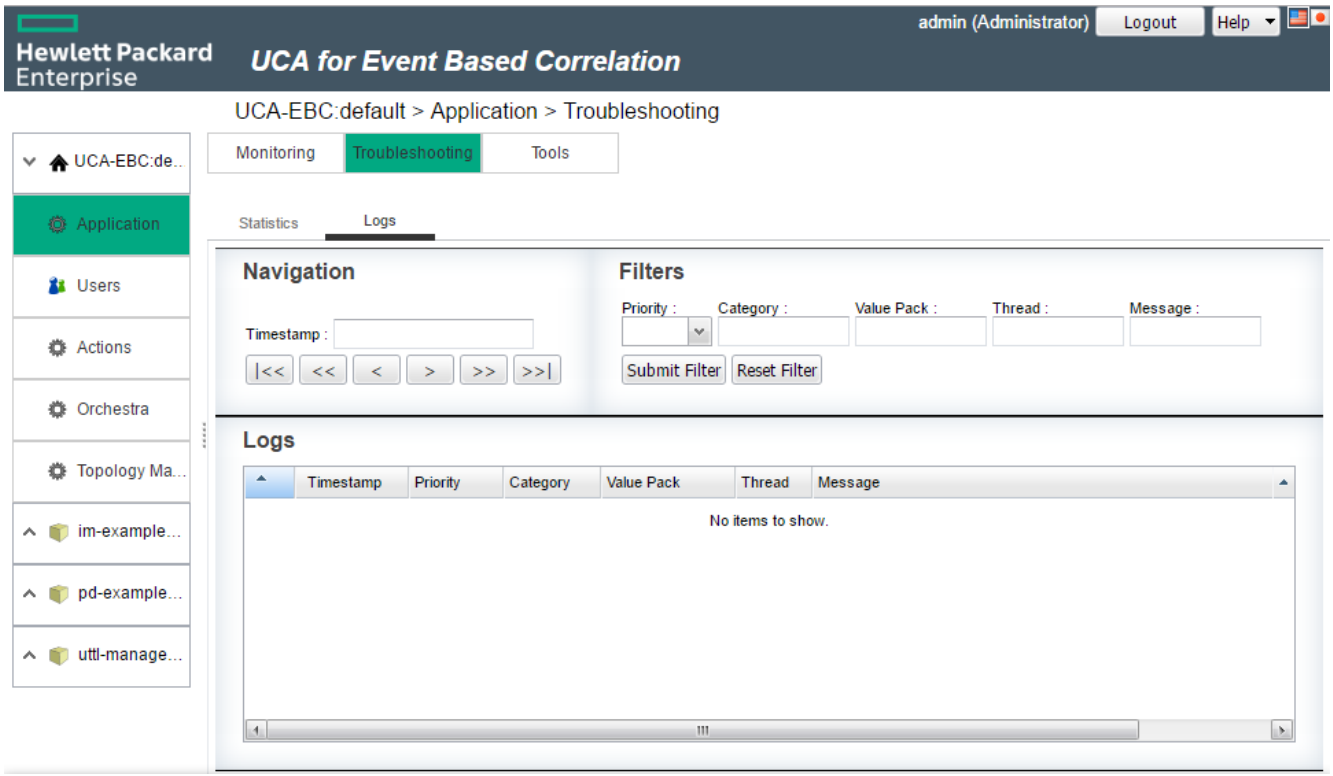


Figure 3: Troubleshooting/Log panel at Application level

Each troubleshooting panel at each level (application, value pack, and scenario) contains two sub-panels:

- A "Statistics" subpanel that contains key performance indicators that help understanding the behavior of UCA for EBC, a value pack or a scenario
- A "Logs" subpanel that displays the full UCA for EBC application logs, the Value Pack logs or a scenario specific logs depending on the level.

The following screenshot shows Troubleshooting/Statistics panel at application level:

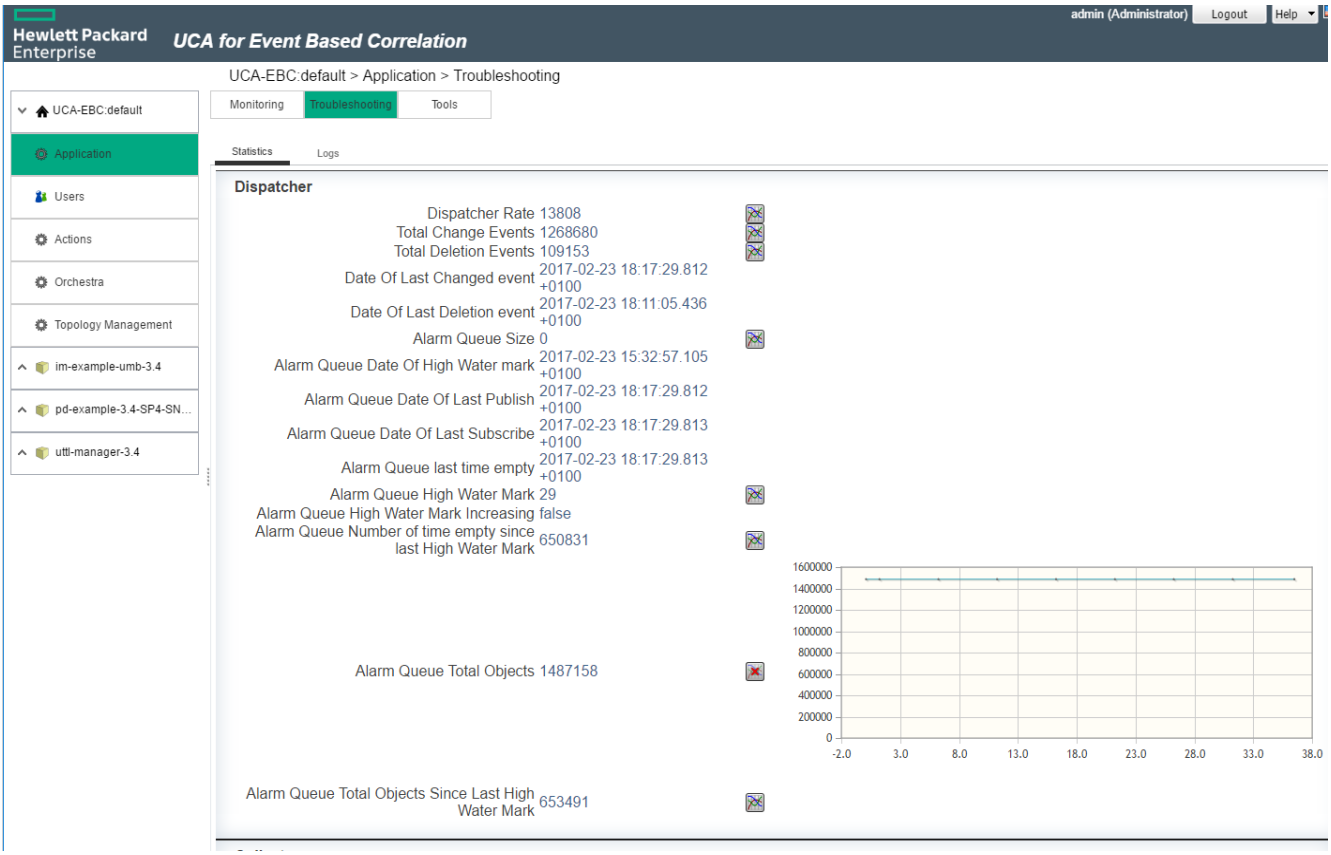


Figure 4: Troubleshooting/Statistics panel at Application Level

NOTE: For more information on how to connect to the UCA for EBC User Interface or to learn about the troubleshooting tools available in the UCA for EBC User Interface, please refer to [R3] `HPE UCA for EBC User Interface Guide`.

5.1.3 JMX Console

To start the Java JMX Console, either locally on the system hosting the UCA for EBC Server or remotely from another system (in which case you will need to adjust the JMX URL accordingly), please execute the following commands:

On both HP-UX and Linux:

```
$ $JAVA_HOME/bin/jconsole
```

Select the "Remote Process" radio button and type the following URL in the input text field:

service:jmx:rmi://<*hostname or IP address*>/jndi/rmi://<*hostname or IP address*>:<*port #*>/uca-ebc

`<hostname or IP address>` is the actual hostname (full DNS name) or the IP address of the UCA for EBC Server system. The default value is localhost.

`<port #>` is the port number for UCA for EBC Server RMI port, 1100 by default for the default instance. Please check the value of the "uca.ebc.jmx.rmi.port" property in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file if you're unsure what RMI port number your UCA for EBC Server is using.

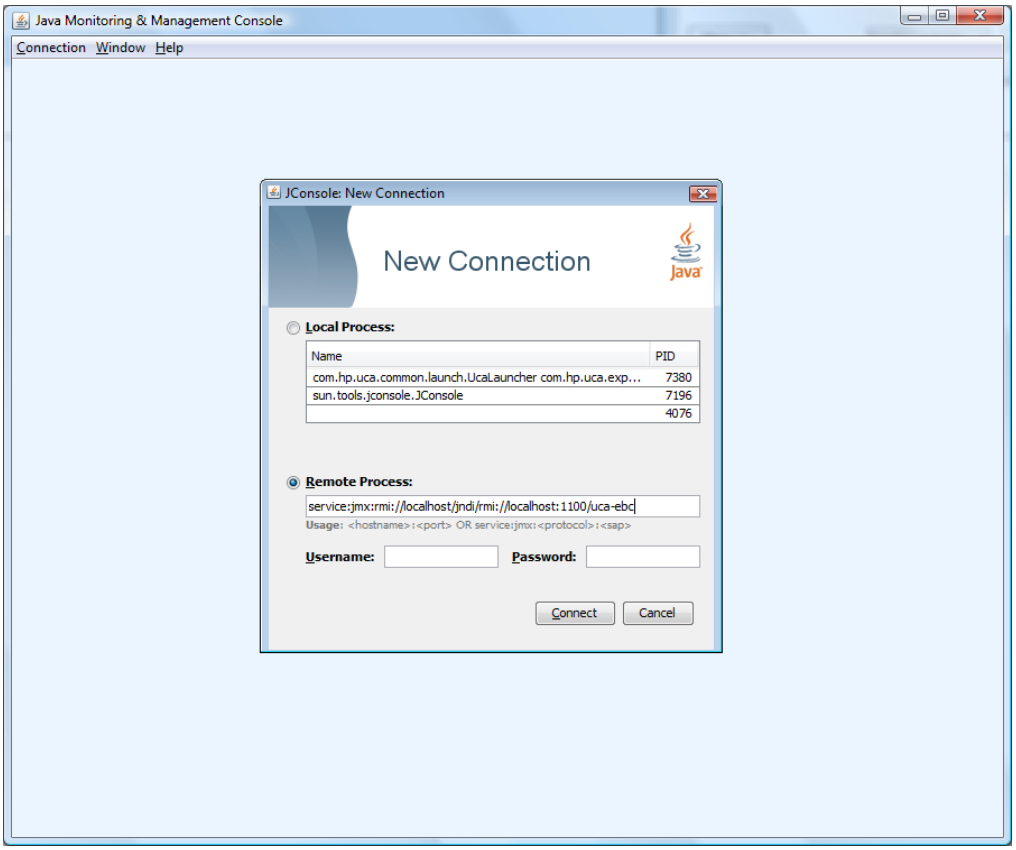


Figure 5: Java JMX Console: Connecting to UCA for EBC Server

Then click on the "Connect" button.

Once you're connected to the Java JMX console for UCA for EBC, you can go to the MBeans tab to get access to the managed Java beans that have been defined specifically for administering UCA for EBC.

All managed beans for UCA for EBC are located under the uca_ebc folder, as seen in the following screenshot:

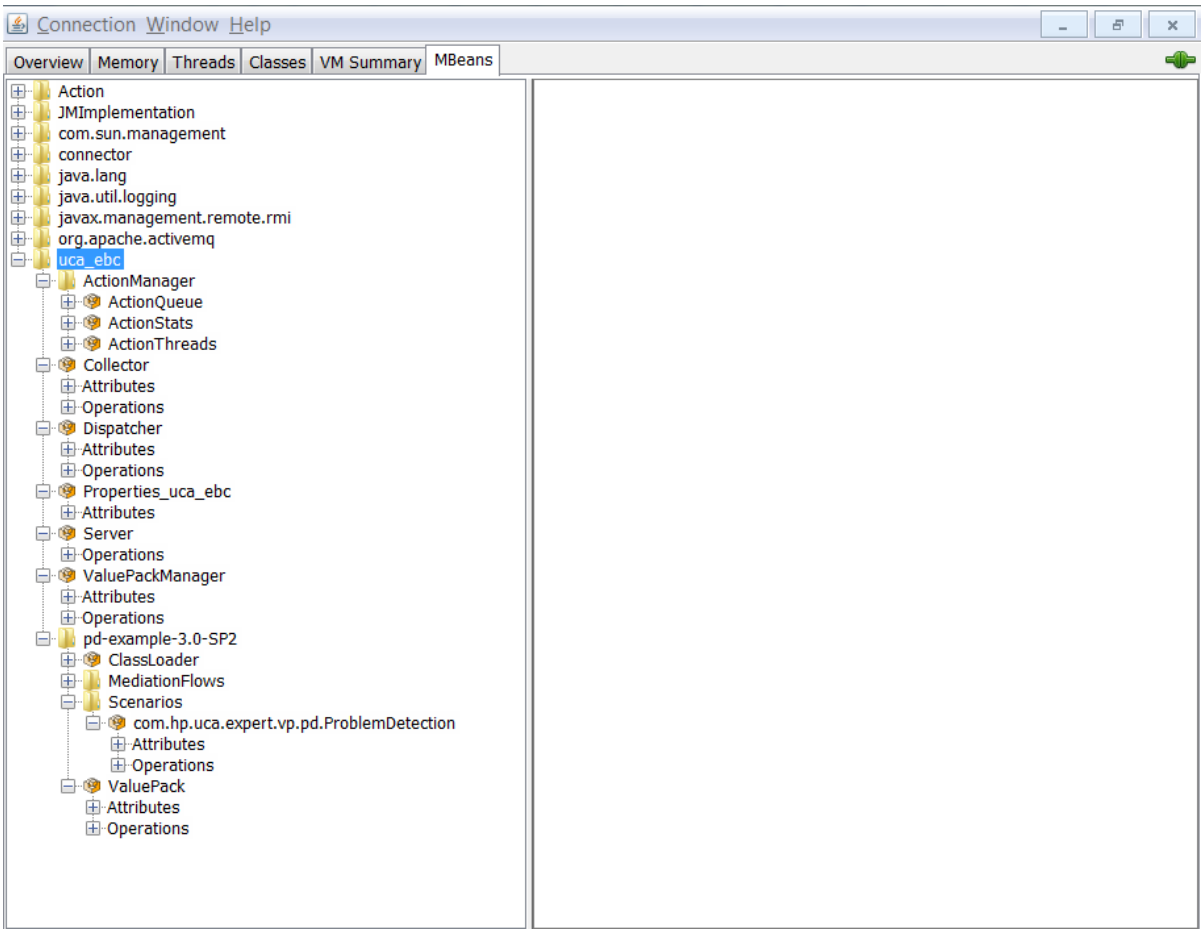


Figure 6: Java JMX Console: UCA for EBC MBeans

Under the uca_ebc folder, there are several folders providing information/statistics*/monitoring/administration features on:

- Internal UCA for EBC components:
  - Action Manager
  - Collector
  - Dispatcher
  - Properties
  - Server
  - Value Pack Manager
- UCA for EBC value packs: there is one folder per running pack

The following sections will provide more details on these folders.

NOTE:

* The statistics available in the Java Console are also available at the UCA for EBC User Interface. Some additional features are available in the Java Console, for example to reset the statistics counters or to get information about internal UCA for EBC components that are not yet available at the UCA for EBC User Interface..

5.1.3.1 Monitoring UCA for EBC internal components

5.1.3.1.1 Monitoring UCA for EBC Action Manager

The UCA for EBC Action Manager is an internal UCA for EBC component that provides the capability to process asynchronous actions requested in the Drools rules files of an UCA for EBC Value Pack scenario. Asynchronous actions are created when the following code is present in a Drools rules file of a scenario:

```
Action action = new Action("TeMIP_AO_Directives_localhost");
action.addCommand("directiveName", "ACKNOWLEDGE");
action.addCommand("entityName", a.getIdentifier());
action.addCommand("UserId", "UCA EBC");
action.executeAsync(AODirectiveKey.ENTITY_NAME);
```

These asynchronous actions are handled by the UCA for EBC Action Manager internal component and are processed by the proper Channel Adapter or UMB Adapter on the mediation layer (either OSS Open Mediation V7.2 or UMB V1.1).

In the Java Console, the Action Manager folder contains the following sub-folders:

- Action Queue: this queue contains the list of asynchronous actions that are currently being processed
- Action Statistics: Information about the performance rate of the Action Manager
- Action Threads: Information about the Action Manager thread pool

The following screenshots shows the UCA for EBC Action Manager component at the Java JMX Console:

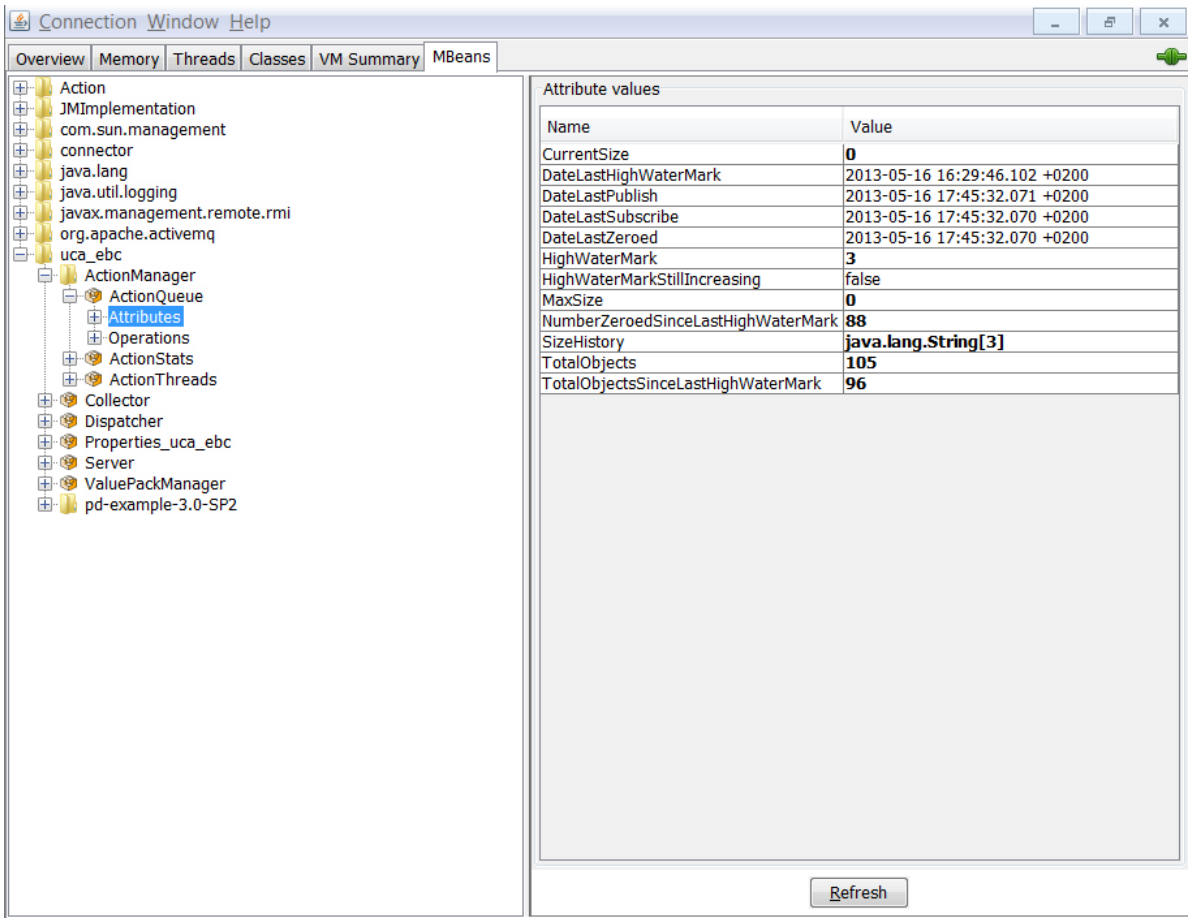


Figure 7: Java JMX Console: UCA for EBC Action Manager

The following sections will provide more detail on the sub-components of the UCA for EBC Action Manager available at the Java JMX console.

NOTE:

For more information on asynchronous actions please refer to: [R2] `HPE UCA for EBC Value Pack Development Guide`.

Action Queue

The Action Queue can be monitored at the Java JMX console using both attributes and operations.

The following table lists the attributes of the Action Queue that are shown on the Java JMX console:

Table 17: Java JMX Console: UCA for EBC Action Manager – Action Queue – Attributes

Attribute name	Settable	Description
CurrentSize	No	The current size of the Action Queue (in number of asynchronous actions in the queue)
DateLastHighWaterMark	No	Date and time of the last high water mark for the Action Queue
DateLastPublish	No	Date and time of the last time an asynchronous action was added to the queue
DateLastSubscribe	No	Date and time of the last time an asynchronous action was removed from the queue to be processed by a thread
DateLastZeroed	No	Date and time of the last time the Action Queue was empty
HighWaterMark	No	Value of the last high water mark for the Action Queue (in number of asynchronous actions in the queue)
HighWaterMarkStillIncreasing	No	Whether the high water mark for the Action Queue is still increasing or not
MaxSize	No	Maximum size of the ActionQueue (in number of asynchronous actions in the queue)
NumberZeroedSinceLastHighWaterMark	No	The number of times the Action Queue size was 0 since the last high water mark
SizeHistory	No	A history of the size of the ActionQueue (in number of asynchronous actions in the queue)
TotalObjects	No	Total number of asynchronous actions that have been added to the Action Queue since start-up
TotalObjectsSinceLastHighWaterMark	No	Total number of asynchronous actions that have been added to the Action Queue since last high water mark

The following table lists the operations that can be executed on the Action Queue using the Java JMX console:

Table 18: Java JMX Console: UCA for EBC Action Manager – Action Queue – Operations

Operation name	Explanation
resetQueueHistory()	Resets all Action Queue counters (attributes)

Action Statistics

Action Statistics can be monitored at the Java JMX console using both attributes and operations.

The following table lists the attributes of the Action Statistics that are shown on the Java JMX console:

Table 19: Java JMX Console: UCA for EBC Action Manager – Action Statistics – Attributes

Attribute name	Settable	Description

ConsolidatedRate	No	The consolidated (average) performance rate of the Action Manager (in number of asynchronous actions processed per second)
HighestRate	No	The highest performance rate of the Action Manager (in number of asynchronous actions processed per second)
LastRate	No	The last performance rate of the Action Manager (in number of asynchronous actions processed per second)
LongestBurstRate	No	The performance rate of the longest burst of the Action Manager (in number of asynchronous actions processed per second)

The following table lists the operations that can be executed on the Action Statistics using the Java JMX console:

Table 20: Java JMX Console: UCA for EBC Action Manager – Action Statistics – Operations

Operation name	Explanation
resetRates()	Resets all Action Statistics rates (i.e. attributes)

Action Threads

Action Threads can be monitored at the Java JMX console using both attributes and operations.

The following table lists the attributes of the Action Threads that are shown on the Java JMX console:

Table 21: Java JMX Console: UCA for EBC Action Manager – Action Threads – Attributes

Attribute name	Settable	Description
FailedActions	No	The total number of failed asynchronous actions of the Action Manager
NbActiveThread	No	The current number of active threads in the thread pool of the Action Manager
NbPoolThread	No	The total number of threads in the thread pool of the Action Manager

The following table lists the operations that can be executed on the Action Threads using the Java JMX console:

Table 22: Java JMX Console: UCA for EBC Action Manager – Action Threads – Operations

Operation name	Explanation
resetCounters()	Resets all Action Threads counters (i.e. attributes)

5.1.3.1.2 Monitoring UCA for EBC Collector

The UCA for EBC Collector is an internal UCA for EBC component that collects all events (Alarms, etc...) coming into UCA for EBC either from the OSS Open Mediation V7.2 mediation layer (events coming from UMB do not go through the Collector) or from the uca-ebc-injector tool.

Monitoring the UCA for EBC Collector component is akin to measuring the input rate of UCA for EBC.

All incoming events are first validated to weed out invalid/unrecognized types of events. Validation errors will result in the events being rejected by the Collector.

The following screenshot shows the UCA for EBC Collector component at the Java JMX Console:

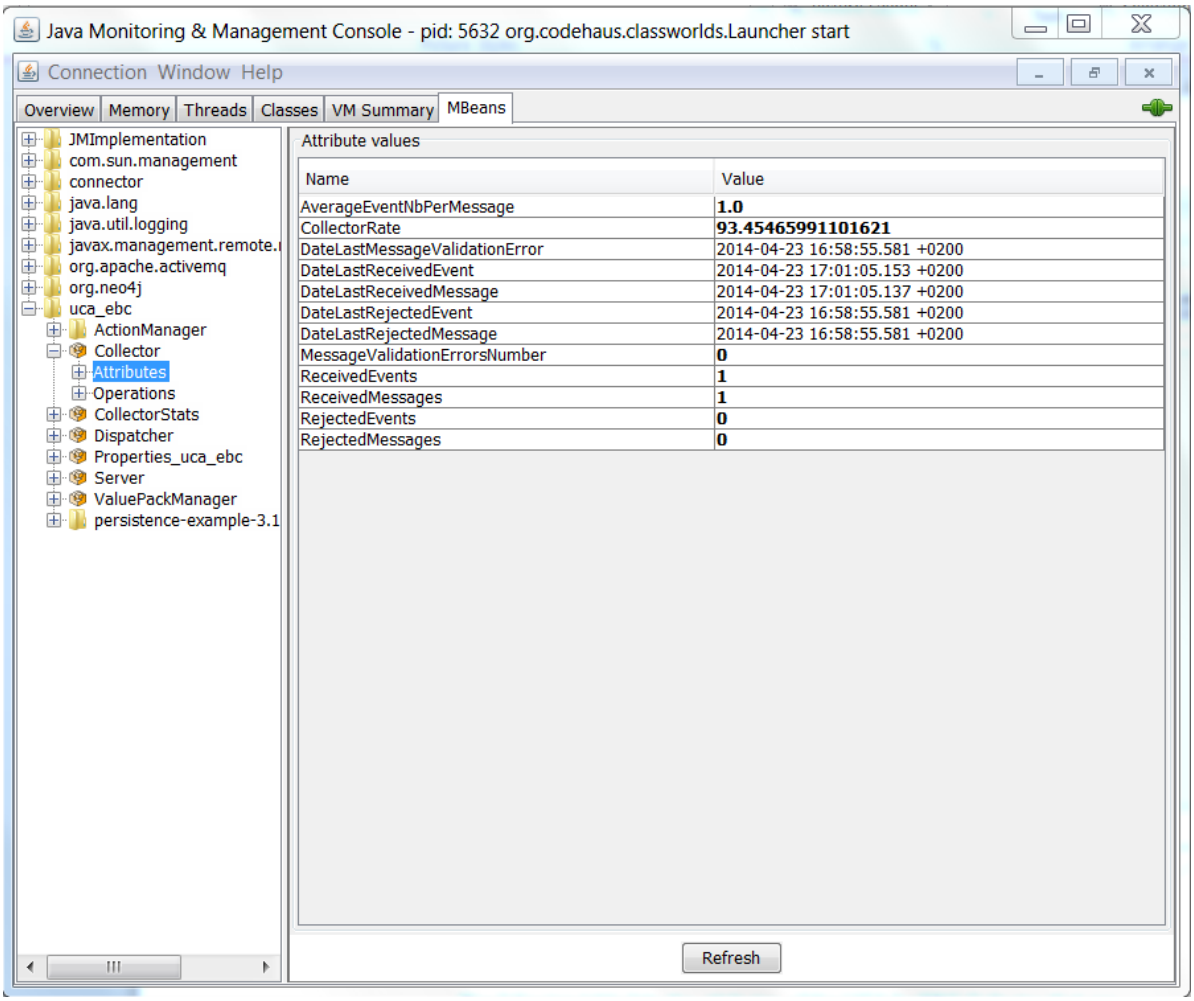


Figure 8: Java JMX Console: UCA for EBC Collector - Attributes

The UCA for EBC Collector can be monitored at the Java JMX console using both attributes and operations.

The following table lists the attributes of the UCA for EBC Collector that are shown on the Java JMX console:

Table 23: Java JMX Console: UCA for EBC Collector - Attributes

Attribute name	Settable	Description
AverageEventNbPerMessage	No	Average number of events(1) per JMS message received by the collector, i.e. batching factor
CollectorRate	No	Collector rate is the average event rate going through the Collector (in events per second)
DateLastMessageValidationError	No	Date and time of the last event(1) in error (due to validation error) received by the Collector
DateLastReceivedEvent	No	Date and time of the last event(1) (Alarms, etc...) received by the Collector
DateLastReceivedMessage	No	Date and time of the last JMS message(1) received by the Collector
DateLastRejectedEvent	No	Date and time of the last event(1) rejected by the Collector
DateLastRejectedMessage	No	Date and time of the last JMS message(1) rejected by the Collector
MessageValidationErrorsNumber	No	Number of events(1) in error (due to validation error) received by the Collector
ReceivedEvents	No	Number of events(1) (Alarms, etc...) received by the Collector
ReceivedMessages	No	Number of JMS messages(1) received by the Collector
RejectedEvents	No	Number of events(1) (Alarms, etc...) rejected by the Collector
RejectedMessages	No	Number of JMS messages(1) rejected by the Collector

NOTE:

(1) The UCA for EBC Collector receives JMS message which can contain any number of events (Alarms, etc.), i.e. a batch of events. This explains why there are Collector statistics for both JMS messages and events.

The following table lists the operations that can be executed on the UCA for EBC Collector using the Java JMX console:

Table 24: Java JMX Console: UCA for EBC Collector – Operations

Operation name	Explanation
resetCounters()	Resets all Action Threads counters (i.e. attributes)

NOTE: For more information on the uca-ebc-injector tool please refer to the following section 2.2.2 "uca-ebc-injector".

5.1.3.1.3 Monitoring UCA for EBC Dispatcher

The UCA for EBC Dispatcher is an internal UCA for EBC component that receives events (Alarms, etc...) coming from the UCA for EBC Collector and forwards those events to any eligible scenario (a property of the scenario states whether a scenario is eligible to receiving incoming events or not) of any value pack currently running on UCA for EBC.

The following screenshot shows the UCA for EBC Dispatcher component at the Java JMX Console:

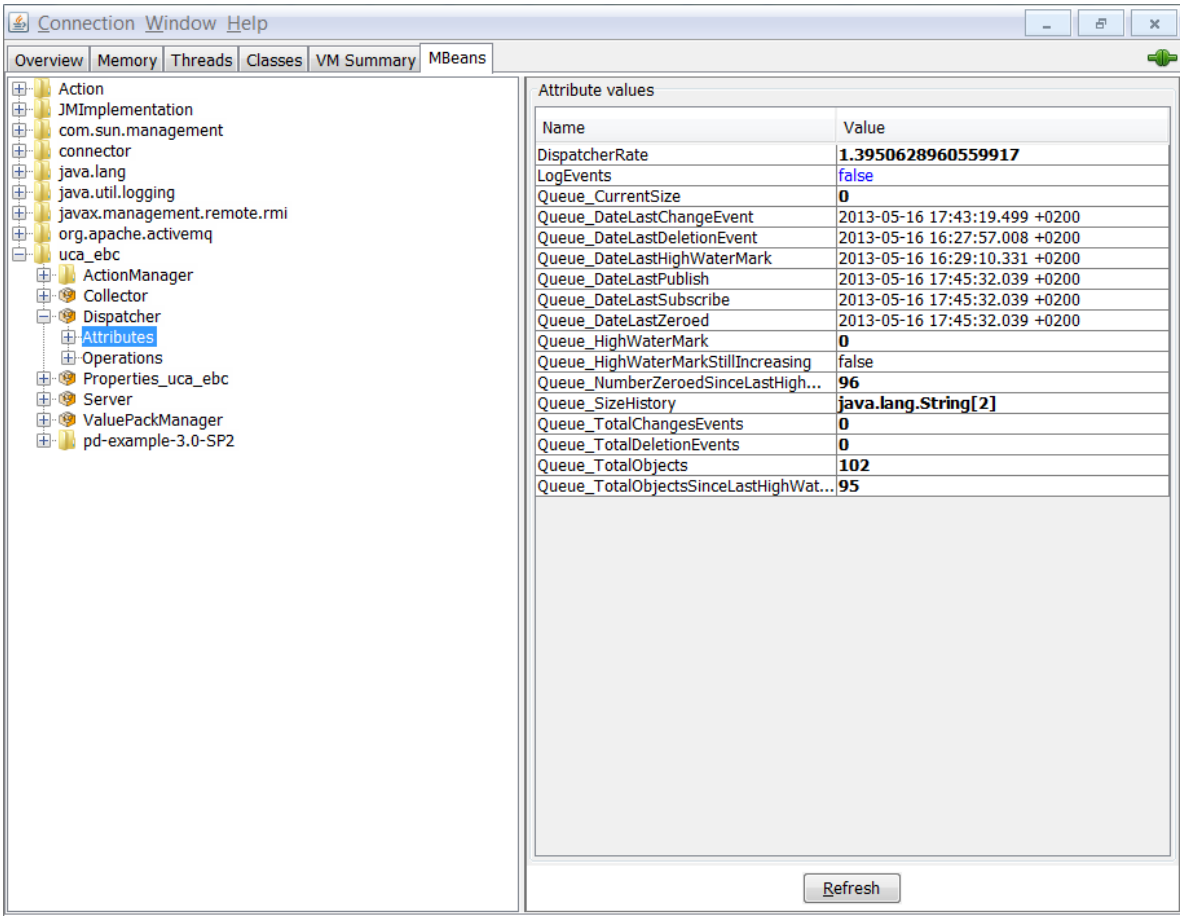


Figure 9: Java JMX Console: UCA for EBC Dispatcher - Attributes

The UCA for EBC Dispatcher can be monitored at the Java JMX console using both attributes and operations.

The following table lists the attributes of the UCA for EBC Dispatcher that are shown on the Java JMX console:

Table 25: Java JMX Console: UCA for EBC Dispatcher - Attributes

Attribute name	Settable	Description
DispatcherRate	No	The event rate of the dispatcher (in number of events per second)
LogEvents	Yes	A flag indicating whether the Dispatcher should log the list of events that it processes or not. THIS ATTRIBUTE IS OBSOLETE. DO NOT USE IT.
Queue_CurrentSize	No	The current size of the Dispatcher queue (in number of events)
Queue_DateLastChangeEvent	No	The date and time of the last "change event" that was added to the Dispatcher queue
Queue_DateLastDeletionEvent	No	The date and time of the last "deletion event" that was added to the Dispatcher queue
Queue_DateLastHighWaterMark	No	The date and time of the last high water mark of the Dispatcher queue
Queue_DateLastPublish	No	Date and time of the last time an event was added to the queue
Queue_DateLastSubscrib	No	Date and time of the last time an event was removed from the queue to be processed
Queue_DateLastZeroed	No	The date and time of the last time the Dispatcher queue was empty
Queue_HighWaterMark	No	The value of the high water mark of the Dispatcher queue (in number of events)
Queue_HighWaterMarkStillIncreasing	No	Whether the high water mark of the Dispatcher queue is still increasing or not
Queue_NumberZeroedSinceLastHighWaterMark	No	The number of times that the Dispatcher queue was empty since the last high water mark
Queue_SizeHistory	No	The history of the Dispatcher queue size
Queue_TotalChangesEvents	No	The total number of "change events" that have been added to the Dispatcher Queue since start-up
Queue_TotalDeletionEvents	No	The total number of "deletion events" that have been added to the Dispatcher Queue since start-up
Queue_TotalObjects	No	The total number of "objects" that have been added to the Dispatcher Queue since start-up
Queue_TotalObjectsSinceLastHighWaterMark	No	The total number of "objects" that have been added to the Dispatcher Queue since the last high water mark

The following table lists the operations that can be executed on the UCA for EBC Dispatcher using the Java JMX console:

Table 26: Java JMX Console: UCA for EBC Dispatcher - Operations

Operation name	Explanation
resetCounters()	Resets all Dispatcher counters (i.e. attributes), except the LogEvents attribute

5.1.3.1.4 Monitoring UCA for EBC Properties

The UCA for EBC Properties folder at the Java JMX Console shows the file system location of each sub-folder of the UCA for EBC application.

The following screenshot shows the UCA for EBC Properties component at the Java JMX Console:

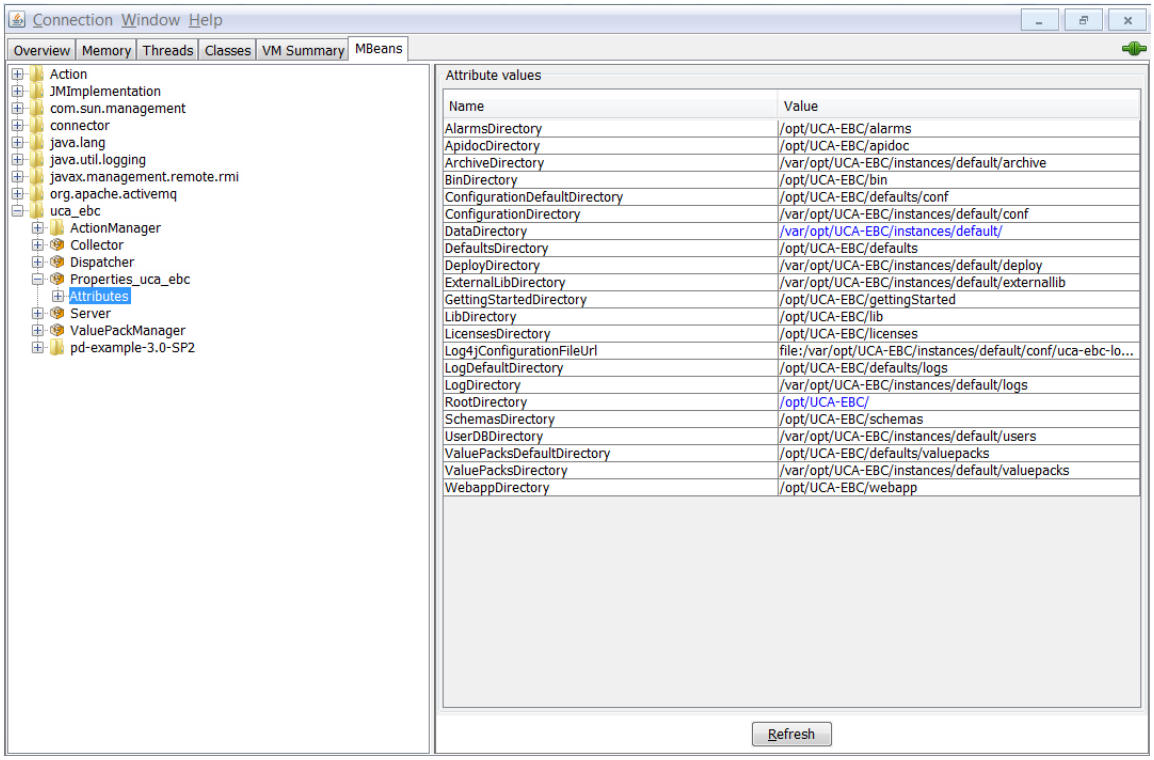


Figure 10: Java JMX Console: UCA for EBC Properties - Attributes

There are no operations that can be executed at the Java JMX Console on the UCA for EBC Properties.

The following table lists the attributes of the UCA for EBC Properties that are shown on the Java JMX console:

Table 27: Java JMX Console: UCA for EBC Properties - Attributes

Attribute name	Settable	Description
AlarmsDirectory	No	Default Value: ${UCA_EBC_HOME}/alarms
ApidocDirectory	No	Default Value: ${UCA_EBC_HOME}/apidoc
ArchiveDirectory	No	Default Value: ${UCA_EBC_INSTANCE}/archive
BinDirectory	No	Default Value: ${UCA_EBC_HOME}/bin
ConfigurationDefaultDirectory	No	Default Value: ${UCA_EBC_HOME}/defaults/conf
ConfigurationDirectory	No	Default Value: ${UCA_EBC_INSTANCE}/conf
DataDirectory	Yes	Default Value: ${UCA_EBC_INSTANCE}
DefaultsDirectory	No	Default Value: ${UCA_EBC_HOME}/defaults
DeployDirectory	No	Default Value: ${UCA_EBC_INSTANCE}/deploy
ExternalLibDirectory	No	Default Value: ${UCA_EBC_INSTANCE}/externallib
GettingStartedDirectory	No	Default Value: ${UCA_EBC_HOME}/gettingStarted
LibDirectory	No	Default Value: ${UCA_EBC_HOME}/lib
LicensesDirectory	No	Default Value: ${UCA_EBC_HOME}/licenses
Log4jConfigurationFileUrl	No	Default Value: file:${UCA_EBC_VAR}/conf/uca-ebc-log4j.xml
LogDefaultDirectory	No	Default Value: ${UCA_EBC_HOME}/defaults/logs

LogDirectory	No	Default Value: ${UCA_EBC_INSTANCE}/logs
RootDirectory	Yes	Default Value: ${UCA_EBC_HOME}
SchemasDirectory	No	Default Value: ${UCA_EBC_HOME}/schemas
ValuePacksDefaultDirectory	No	Default Value: ${UCA_EBC_HOME}/defaults/valuepacks
ValuePacksDirectory	No	Default Value: ${UCA_EBC_INSTANCE}/valuepacks
WebappDirectory	No	Default Value: ${UCA_EBC_HOME}/webapp

5.1.3.1.5 Monitoring UCA for EBC Server

The following screenshot shows the UCA for EBC Server component at the Java JMX Console:

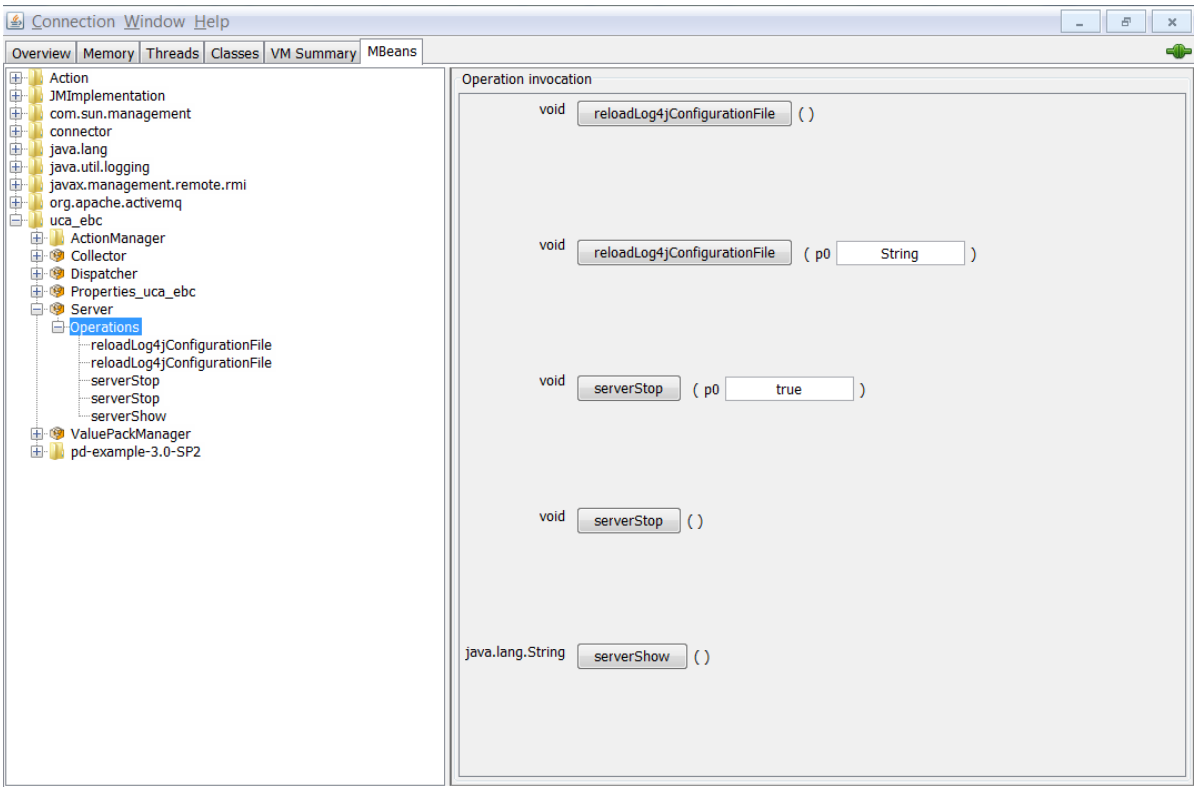


Figure 11: Java JMX Console: UCA for EBC Server - Operations

The UCA for EBC Server can be monitored at the Java JMX console using operations.

The following table lists the operations that can be executed on the UCA for EBC Server using the Java JMX console:

Table 28: Java JMX Console: UCA for EBC Server - Operations

Operation name	Explanation
reloadLog4jConfigurationFile()	Reloads the log4J configuration file.
reloadLog4jConfigurationFile(String)	Reloads the log4J configuration file, using the log4J configuration file located at the path passed as parameter
serverStop(boolean)	Stops UCA for EBC Server. The parameter is a boolean flag that indicates whether to restart (true) UCA for EBC Server once it has stopped or not (false).
serverStop()	Stops UCA for EBC Server.
serverShow()	Displays the status of UCA for EBC Server, whether it's running or not.

5.1.3.1.6 Monitoring UCA for EBC Value Pack Manager

The UCA for EBC Value Pack Manager is an internal UCA for EBC component. It manages all the Value Packs of the UCA for EBC application.

The following screenshot shows the UCA for EBC Value Pack Manager component at the Java JMX Console:

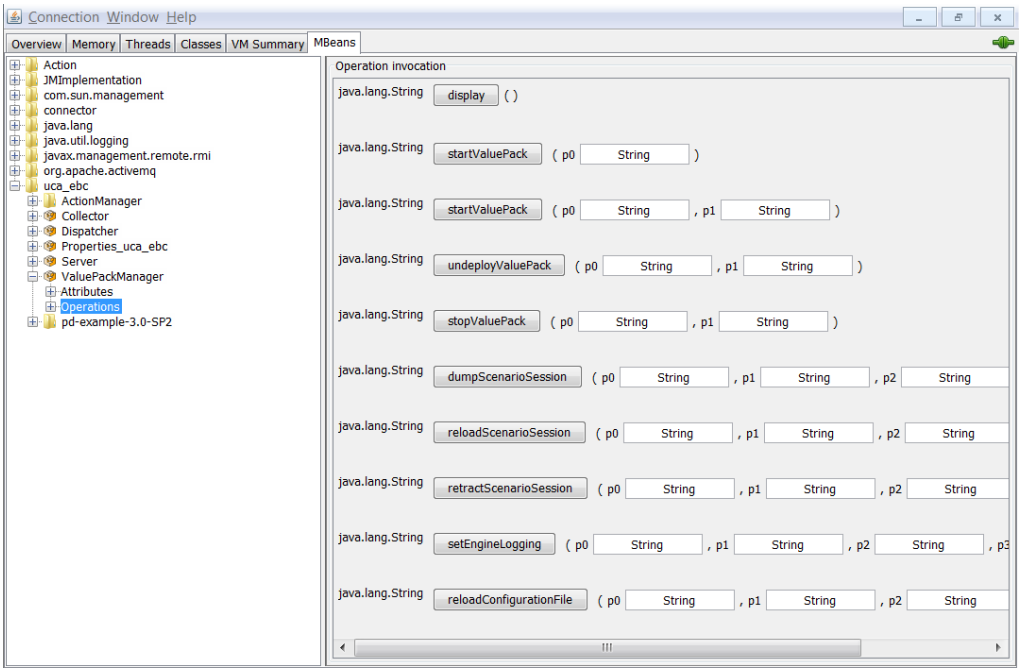


Figure 12: Java JMX Console: UCA for EBC Value Pack Manager - Operations

The UCA for EBC Value Pack Manager can be monitored at the Java JMX console using both attributes and operations.

The following table lists the attributes of the UCA for EBC Value Pack Manager that are shown on the Java JMX console:

Table 29: Java JMX Console: UCA for EBC Value Pack Manager - Attributes

Attribute name	Settable	Description
ActiveValuePacks	No	The list of active value pack currently running on UCA for EBC
AllValuePacks	No	The list of all value pack currently running/degraded/stopped/not deployed on UCA for EBC
DeploymentHistory	No	The complete history of deployments of value packs on UCA for EBC

The following table lists the operations that can be executed on the UCA for EBC Value Pack Manager using the Java JMX console:

Table 30: Java JMX Console: UCA for EBC Value Pack Manager - Operations

Operation name	Explanation
display()	Lists all Value Packs and scenarios currently running on UCA for EBC
startValuePack(String)	Starts a Value Pack identified by the path of the Value Pack in the ${UCA_EBC_INSTANCE}/deploy folder passed as parameter.

	For example: "deploy/<*Value Pack Name*>-<*Value Pack Version*>" Parameter 1: path of the Value Pack
startValuePack(String, String)	Starts a Value Pack identified by its name and version passed as parameters. Parameter 1: Value Pack Name Parameter 2: Value Pack Version
undeployValuePack(String, String)	Undeploys a Value Pack identified by its name and version passed as parameters Parameter 1: Value Pack Name Parameter 2: Value Pack Version
stopValuePack(String, String)	Stops a Value Pack identified by its name and version passed as parameters Parameter 1: Value Pack Name Parameter 2: Value Pack Version
dumpScenarioSession(String, String, String)	Dumps the Drools Working Memory of a scenario of a value pack identified by the value pack name, version, and the scenario name Parameter 1: Value Pack Name Parameter 2: Value Pack Version Parameter 3: Scenario Name If parameter 3 is omitted, then the Drools Working Memory of all the scenarios of the Value Pack specified in parameters 1, and 2 is dumped. If parameter 1, 2, and 3 are omitted, then the Drools Working Memory of all the scenarios of all the value packs is dumped.
reloadScenarioSession(String, String, String, String)	Reloads a specific rule file of a scenario of a value pack identified by the value pack name, version, the scenario name, and the rule file name Parameter 1: Value Pack Name Parameter 2: Value Pack Version Parameter 3: Scenario Name Parameter 4: Rule File Name If Parameter 4 is omitted, then all rules files of the scenario of the Value Pack specified in parameters 1, 2, and 3 are reloaded.

	If parameter 3 and 4 are omitted, then all rules files of all the scenarios of the Value Pack specified in parameters 1, and 2 are reloaded. If parameter 1, 2, 3 and 4 are omitted, then all rules files of all the scenarios of all the value packs are reloaded.
retractScenarioSession(String, String, String)	Clears the Drools Working Memory of a scenario of a value pack identified by the value pack name, version, and the scenario name Parameter 1: Value Pack Name Parameter 2: Value Pack Version Parameter 3: Scenario Name If parameter 3 is omitted, then the Drools Working Memory of all the scenarios of the Value Pack specified in parameters 1, and 2 is cleared. If parameter 1, 2, and 3 are omitted, then the Drools Working Memory of all the scenarios of all the value packs is cleared.
setEngineLogging(String, String, String, Boolean)	Enables/Disables scenario specific Drools engine logging for a Value Pack scenario specified by the Value Pack name, version, and scenario name. The 4th parameter is a Boolean value: true for enabling, false for disabling scenario specific Drools engine logging. Parameter 1: Value Pack Name Parameter 2: Value Pack Version Parameter 3: Scenario Name Parameter 4: A Flag indicating whether to enable/disable engine logging (true/false) If parameter 3 is omitted, then the engine logging of all the scenarios of the Value Pack specified in parameters 1, and 2 is enabled or disabled depending on the value of parameter 4. If parameter 1, 2, and 3 are omitted, then the engine logging of all the scenarios of all the value packs is enabled or disabled depending on the value of parameter 4.
reloadConfigurationFile(String, String, String, String)	Reloads a configuration file for a Value Pack scenario specified by the Value Pack name, version, and scenario name. The 4th parameter is the name of the configuration file to reload. Parameter 1: Value Pack Name Parameter 2: Value Pack Version Parameter 3: Scenario Name Parameter 4: Configuration file name

	If parameter 4 is omitted, all configuration files of the scenario are reloaded. If parameters 3 and 4 are omitted, all configuration files of all scenarios of the value pack are reloaded. If parameters 1, 2, 3 and 4 are omitted, all configuration files of all scenarios of all value packs are reloaded.

5.1.3.2 Monitoring UCA for EBC value packs

Each UCA for EBC Value Pack running has its own sub-folder at the Java JMX Console, under the "uca_ebc" top folder. Each Value Pack sub-folder is named after the Value Pack name and version.

In the Java Console, each Value Pack folder contains the following sub-folders:

- Class Loader: this sub-folder is displayed only if the `uca.ebc.classloader` property in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file has been set to `ucaclassloader` (this is not the case by default) and contains information about the UCA for EBC class loader specific to the Value Pack
- DB flows: this sub-folder contains information about the DB flows specific to the Value Pack
- Mediation flows: this sub-folder contains information about the mediation flows specific to the Value Pack
- Scenarios: this sub-folder contains information on each of the scenarios of the value pack (the contents of this sub-folder is explained in the next section: 5.1.3.3 "Monitoring UCA for EBC scenarios")
- Value Pack: this sub-folder contains information on the value pack itself

The following screenshot shows a sample UCA for EBC Value Pack sub-folder at the Java JMX Console:

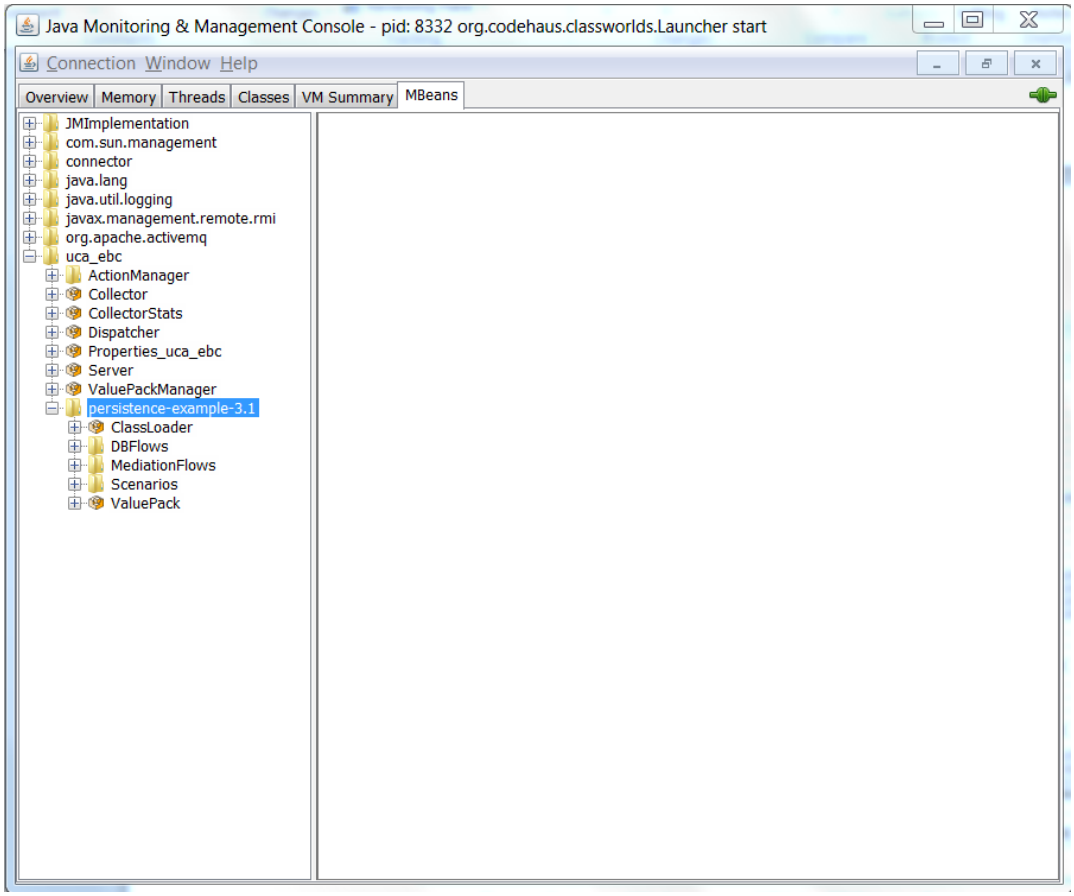


Figure 13: Java JMX Console: a UCA for EBC Value Pack

The following sections will provide more detail on the Class Loader, DB Flows, Mediation flows, Scenarios and Value Pack sub-folders of any UCA for EBC Value Pack at the Java JMX console.

5.1.3.2.1 Class Loader

This sub-folder is displayed only if the `uca.ebc.classloader` property in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file has been set to `ucaclassloader` (this is not the case by default).

The UCA for EBC Value Pack Class Loader represents the UCA EBC class loader for a specific UCA for EBC Value Pack.

The following screenshot shows the attributes available for a UCA for EBC Value Pack Class Loader component at the Java JMX Console:

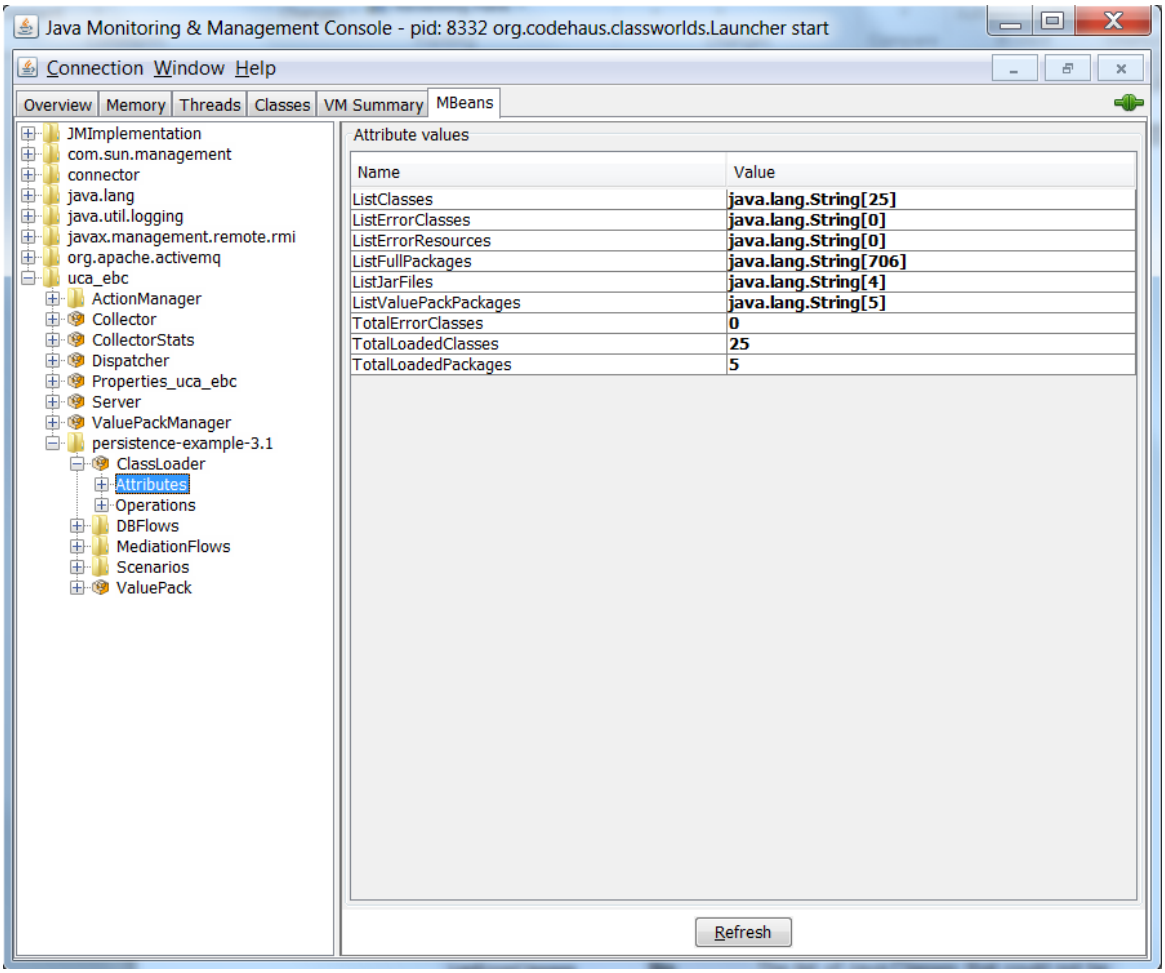


Figure 14: Java JMX Console: UCA for EBC Value Pack - Class Loader - Attributes

Any UCA for EBC Value Pack Class Loader can be monitored at the Java JMX console using both attributes and operations.

The following table lists the attributes of the UCA for EBC Value Pack Class Loader that are shown on the Java JMX console:

Table 31: Java JMX Console: UCA for EBC Value Pack - Class Loader - Attributes

Attribute name	Settable	Description

ListClasses	No	The list of Java Classes loaded by the Value Pack Class Loader
ListErrorClasses	No	The list of Java Classes that could not be loaded by the Value Pack Class Loader
ListErrorResources	No	The list of Java Resources that could not be loaded by the Value Pack Class Loader
ListFullPackages	No	The full list of Java Packages loaded by the Value Pack Class Loader
ListJarFiles	No	The list of JAR files loaded by the Value Pack Class Loader
ListValuePackPackages	No	The list of Value Pack Java Packages loaded by the Value Pack Class Loader
TotalErrorClasses	No	The total number of Java Classes that could not be loaded by the Value Pack Class Loader
TotalLoadedClasses	No	The total number of Java Classes loaded by the Value Pack Class Loader
TotalLoadedPackages	No	The total number of Java Packages loaded by the Value Pack Class Loader

The following screenshot shows the operations available for a UCA for EBC Value Pack Class Loader component at the Java JMX Console:

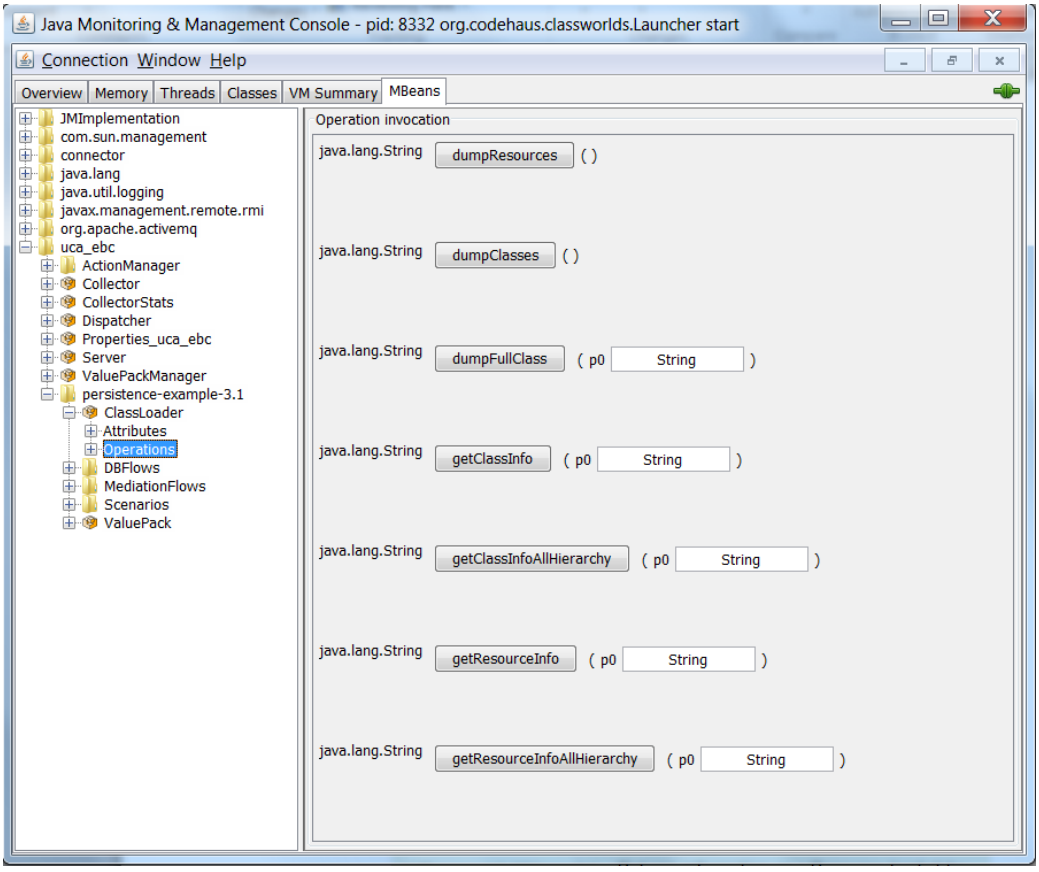


Figure 15: Java JMX Console: UCA for EBC Value Pack - Class Loader - Operations

The following table lists the operations that can be executed on the UCA for EBC Value Pack Class Loader using the Java JMX console:

Table 32: Java JMX Console: UCA for EBC Value Pack - Class Loader - Operations

Operation name	Explanation

dumpResources()	Dumps the list of all the Resources loaded by the Value Pack Class Loader
dumpClasses()	Dumps the list of all the Java Classes loaded by the Value Pack Class Loader
dumpFullClass(String)	Dumps a Java Class loaded by the Value Pack Class Loader. The Java Class is identified by the name of the class passed as a parameter. Parameter 1: Full Class Name
getClassInfo(String)	Returns information on a Java Class loaded by the Value Pack Class Loader. The Java Class is identified by the name of the class passed as a parameter. Parameter 1: Full Class Name
getClassInfoAllHierarchy(String)	Returns information on a Java Class loaded by the Value Pack Class Loader or by the Main Class Loader. The Java Class is identified by the name of the class passed as a parameter. Parameter 1: Full Class Name
getResourceInfo(String)	Returns information on a Resource loaded by the Value Pack Class Loader. The Resource is identified by the name passed as a parameter. Parameter 1: Resource Name
getResourceInfoAllHierarchy(String)	Returns information on a Resource loaded by the Value Pack Class Loader or Main Class Loader. The Resource is identified by the name passed as a parameter. Parameter 1: Resource Name

5.1.3.2.2 DB Flows

The UCA for EBC Value Pack DB Flows represent the DB flows for a specific UCA for EBC Value Pack.

The following screenshot shows the attributes available for a UCA for EBC Value Pack Mediation Flows component at the Java JMX Console:

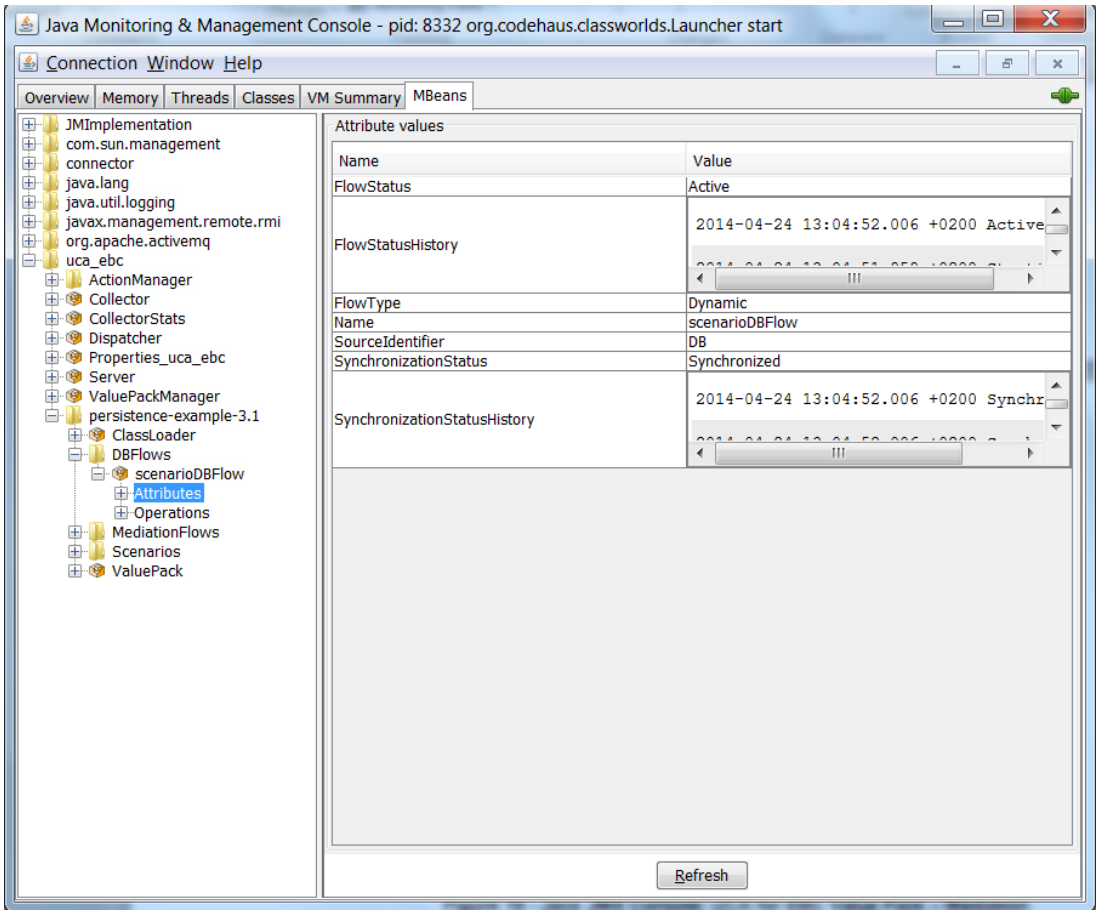


Figure 16: Java JMX Console: UCA for EBC Value Pack – DB Flows - Attributes

Any UCA for EBC Value Pack DB Flow can be monitored at the Java JMX console using both attributes and operations.

The following table lists the attributes of the UCA for EBC Value Pack DB Flows that are shown on the Java JMX console:

Table 33: Java JMX Console: UCA for EBC Value Pack – DB Flows - Attributes

Attribute name	Settable	Description
FlowStatus	No	The status of the DB Flow
FlowStatusHistory	No	A history of the status of the Mediation DB over time
FlowType	No	Either dynamic or static
Name	No	The name of the DB Flow
SourceIdentifier	No	The source identifier of the DB Flow
SynchronizationStatus	No	Either synchronized or synchronizing
SynchronizationStatusHistory	No	A history of the synchronization status of the DB Flow over time

The following screenshot shows the operations available for a UCA for EBC Value Pack Class Loader component at the Java JMX Console:

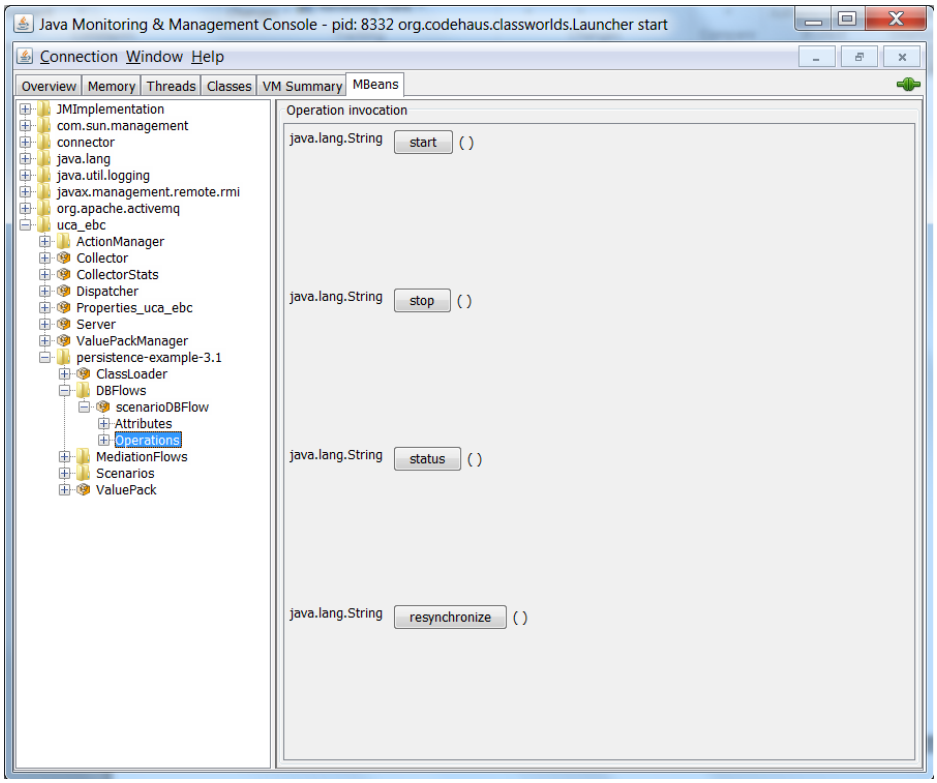


Figure 17: Java JMX Console: UCA for EBC Value Pack – DB Flows - Operations

The following table lists the operations that can be executed on the UCA for EBC Value Pack DB Flows using the Java JMX console:

Table 34: Java JMX Console: UCA for EBC Value Pack – DB Flows - Operations

Operation name	Explanation
start()	Start the DB Flow
stop()	Stop the DB Flow
status()	Displays the status of the DB Flow
resynchronize()	Resynchronizes the DB Flow

5.1.3.2.3 Mediation Flows

The UCA for EBC Value Pack Mediation Flows represent the mediation flows for a specific UCA for EBC Value Pack.

The following screenshot shows the attributes available for a UCA for EBC Value Pack Mediation Flow component at the Java JMX Console:

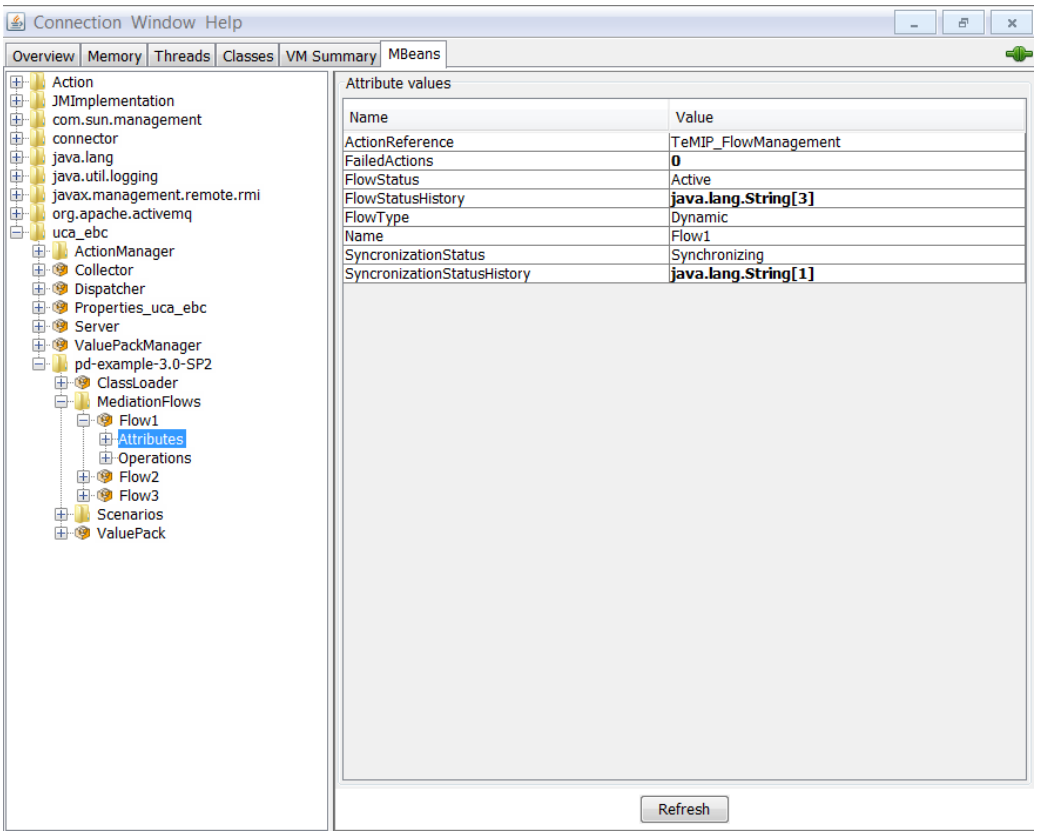


Figure 18: Java JMX Console: UCA for EBC Value Pack – Mediation Flows - Attributes

Any UCA for EBC Value Pack Mediation Flow can be monitored at the Java JMX console using both attributes and operations.

The following table lists the attributes of the UCA for EBC Value Pack Mediation Flows that are shown on the Java JMX console:

Table 35: Java JMX Console: UCA for EBC Value Pack – Mediation Flows - Attributes

Attribute name	Settable	Description
ActionReference	No	The Action Reference (from the ActionRegistry.xml configuration file) associated with the Mediation Flow
FailedActions	No	The number of Failed actions associated with the Mediation Flow (Each action is either a CreateFlow, DeleteFlow, ResynchronizeFlow, or a StatusFlow action)
FlowStatus	No	The status of the Mediation Flow
FlowStatusHistory	No	A history of the status of the Mediation Flow over time
FlowType	No	Either dynamic or static
Name	No	The name of the Mediation Flow
SynchronizationStatus	No	Either synchronized or synchronizing
SynchronizationStatusHistory	No	A history of the synchronization status of the Mediation Flow over time

The following screenshot shows the operations available for a UCA for EBC Value Pack Mediation Flow component at the Java JMX Console:

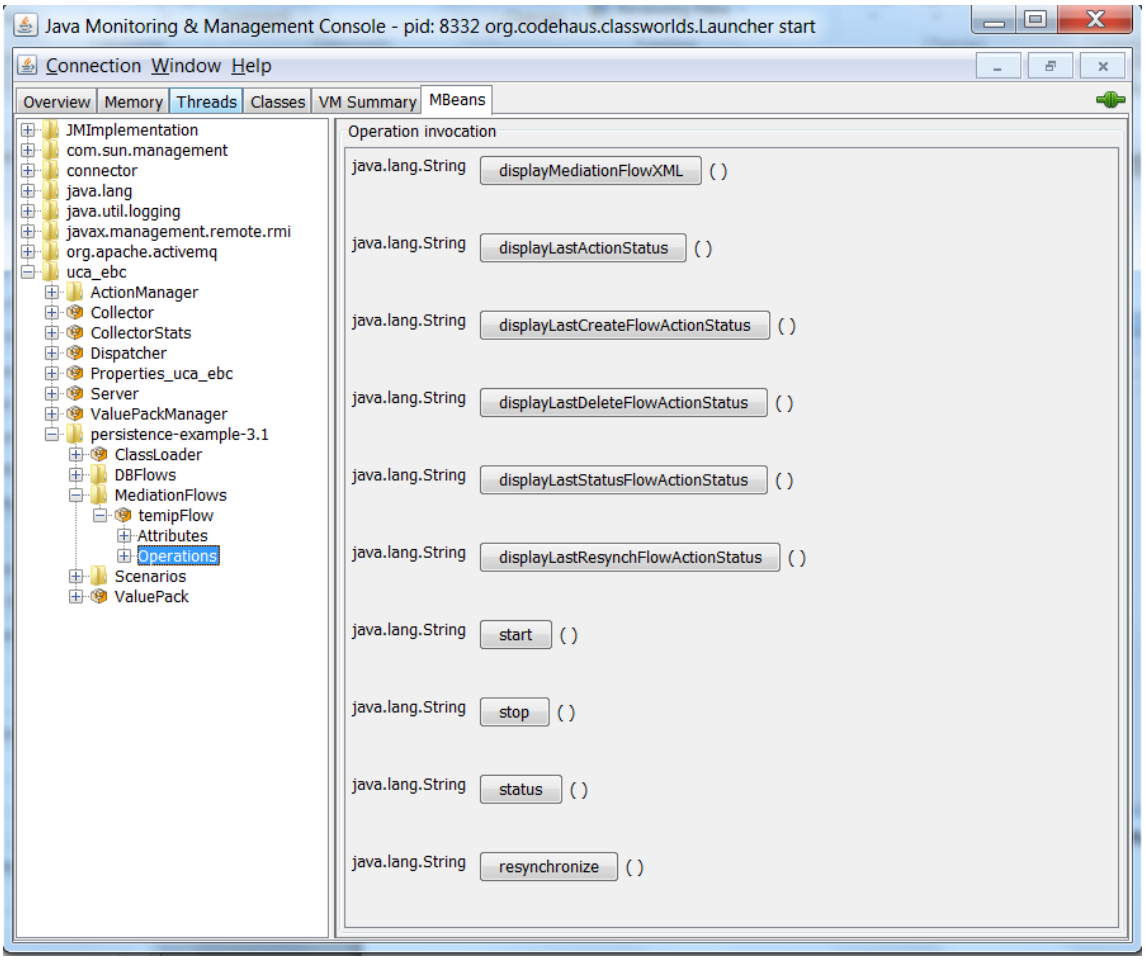


Figure 19: Java JMX Console: UCA for EBC Value Pack – Mediation Flows - Operations

The following table lists the operations that can be executed on the UCA for EBC Value Pack Mediation Flows using the Java JMX console:

Table 36: Java JMX Console: UCA for EBC Value Pack – Mediation Flows - Operations

Operation name	Explanation
start()	Start the Mediation Flow
stop()	Stop the Mediation Flow
status()	Displays the status of the Mediation Flow
resynchronize()	Resynchronizes the Mediation Flow
displayMediationFlowXML()	Displays the XML definition of the Mediation Flow (extracted from the ValuePackConfiguration.xml file)
displayLastActionStatus()	Displays the output of the last action performed on the Mediation Flow (either a CreateFlow, DeleteFlow, ResynchronizeFlow, or a StatusFlow action)
displayLastCreateFlowActionStatus()	Displays the output of the last CreateFlow action performed on the Mediation Flow
displayLastDeleteFlowActionStatus()	Displays the output of the last DeleteFlow action performed on the Mediation Flow
displayLastStatusFlowActionStatus()	Displays the output of the last StatusFlow action performed on the Mediation Flow
displayLastResynchFlowActionStatus()	Displays the output of the last ResynchronizeFlow action performed on the Mediation Flow

5.1.3.2.4 Scenarios

All the scenarios of a value pack are listed under the Scenarios sub-folder of the value pack folder, like in the screenshot below:

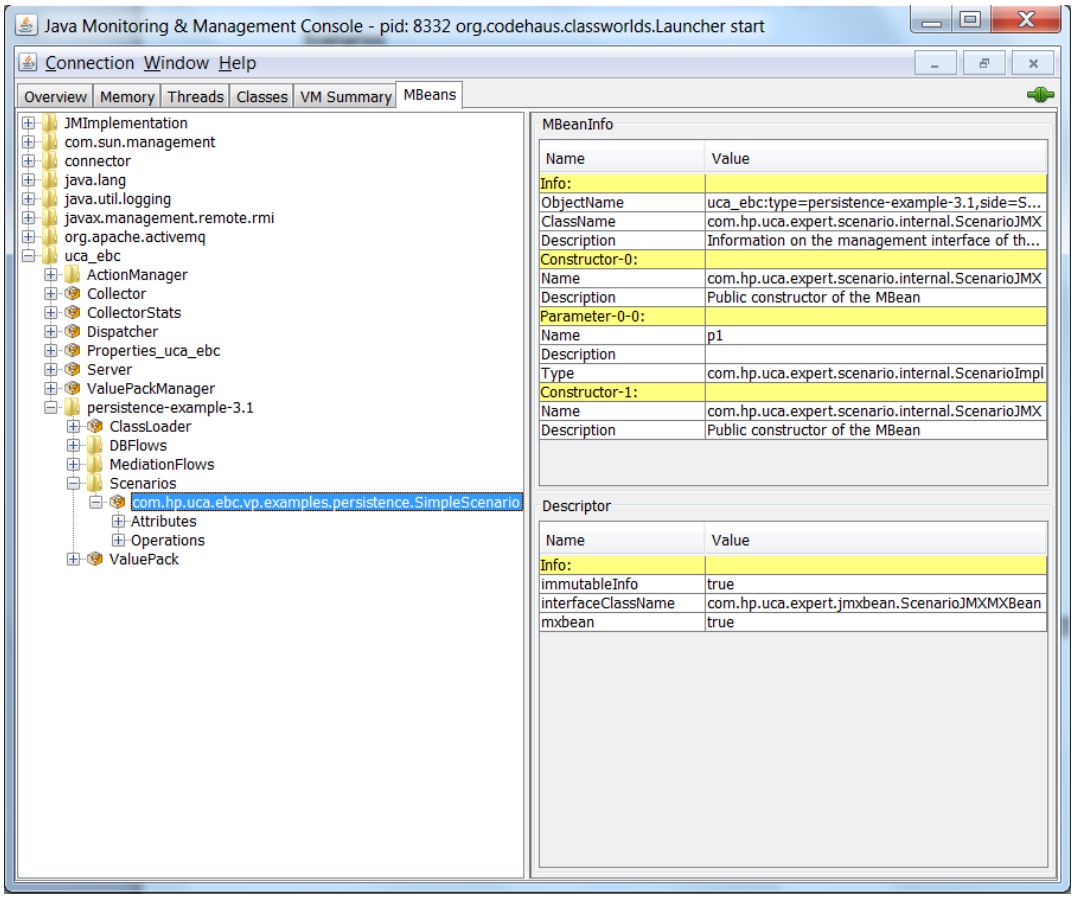


Figure 20: Java JMX Console: UCA for EBC Value Pack – Scenarios

Each scenario sub-folder is named after the scenario. Please see chapter 5.1.3.3 "Monitoring UCA for EBC scenarios" for detailed information on the contents of each scenario sub-folder.

5.1.3.2.5 Value Pack

The Value Pack sub-folder of a UCA for EBC Value Pack presents the attributes and operations for a specific UCA for EBC Value Pack.

The following screenshot shows the attributes available for a Value Pack sub-folder of a UCA for EBC Value Pack at the Java JMX Console:

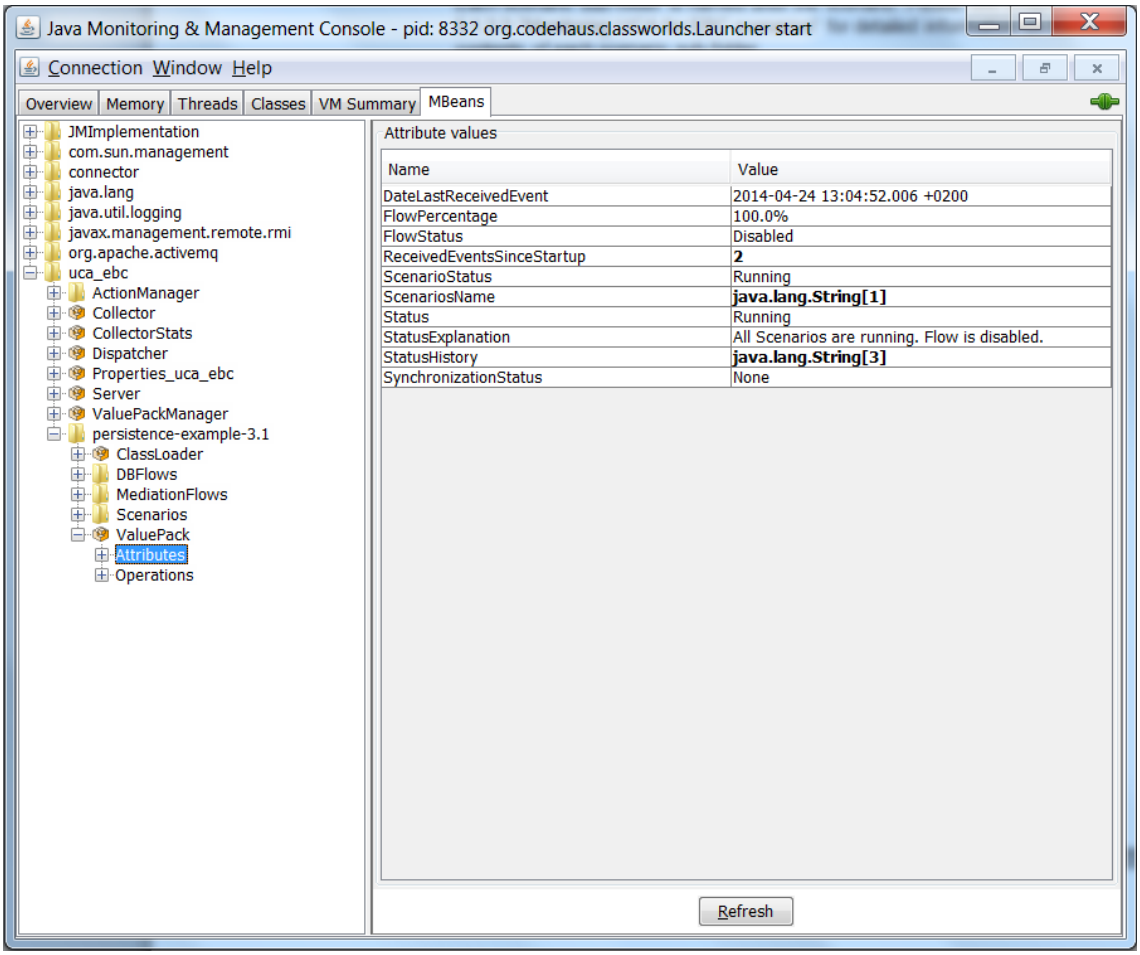


Figure 21: Java JMX Console: UCA for EBC Value Pack – Value Pack - Attributes

Any UCA for EBC Value Pack can be monitored at the Java JMX console using both attributes and operations.

The following table lists the attributes of the UCA for EBC Value Pack that are shown on the Java JMX console:

Table 37: Java JMX Console: UCA for EBC Value Pack – Value Pack - Attributes

Attribute name	Settable	Description
DateLastReceivedEvent	No	The date and time of the last event received by the Value Pack
FlowPercentage	No	Percentage of events received by the Value Pack compared to the total of events received by the UCA for EBC Dispatcher
FlowStatus	No	The status of the Mediation Flow for the Value Pack, either: • Unknown • Disabled • Inactive • Failover • Failed • Active • Starting • Stopping
ReceivedEventsSinceStartup	No	The number of events received by the Value Pack since start-up

ScenarioStatus	No	The status of the Scenarios for the Value Pack, either: • Starting • Running • Degraded • Failed • Stopped • Unknown
ScenariosName	No	The list of scenario names associated with the Value Pack
Status	No	The status of the Value Pack, either: • Starting • Running • Degraded • Failed • Stopping • Stopped • NotDeployed • Unknown
StatusExplanation	No	A detailed explanation of the status of the Value Pack
StatusHistory	No	The full history of the Value Pack statuses, since it was first started
SynchronizationStatus	No	The synchronization status of the Value Pack, either: • Synchronizing • Synchronized

The following screenshot shows the operations available for a Value Pack sub-folder of a UCA for EBC Value Pack at the Java JMX Console:

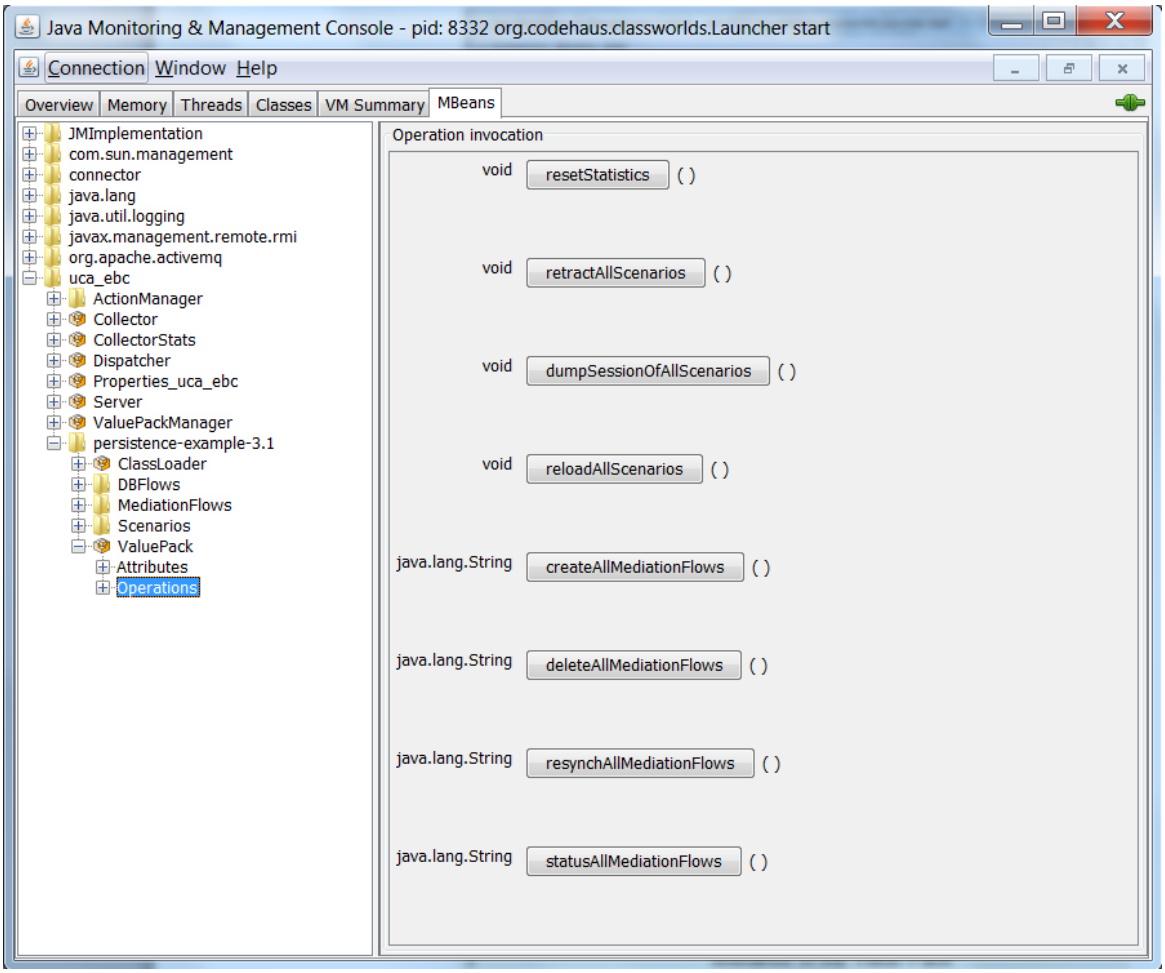


Figure 22: Java JMX Console: UCA for EBC Value Pack – Value Pack - Operations

The following table lists the operations that can be executed on the UCA for EBC Value Pack using the Java JMX console:

Table 38: Java JMX Console: UCA for EBC Value Pack – Value Pack - Operations

Operation name	Explanation
resetStatistics()	Resets the statistics for the Value Pack
retractAllScenarios()	Clears the Drools Working Memory of all the scenarios of the Value Pack
dumpSessionOfAllScenarios()	Dumps the Drools Working Memory of all the scenarios of the Value Pack
reloadAllScenarios()	Reloads all rules files of all the scenarios of the Value Pack
createAllMediationFlows()	Creates all the mediation flows associated with the Value Pack
deleteAllMediationFlows()	Deletes all the mediation flows associated with the Value Pack
resynchAllMediationFlows()	Resynchronizes all the mediation flows associated with the Value Pack
statusAllMediationFlows()	Retrieves the status of all the mediation flows associated with the Value Pack

5.1.3.3 Monitoring UCA for EBC scenarios

Each scenario of a running UCA for EBC Value Pack has its own sub-folder at the Java JMX Console, under the "uca_ebc/<value pack name>-<value pack version>/Scenarios" folder. Each Scenario sub-folder is named after the Scenario.

The following screenshot shows the attributes available for a Scenario sub-folder of a UCA for EBC Value Pack at the Java JMX Console:

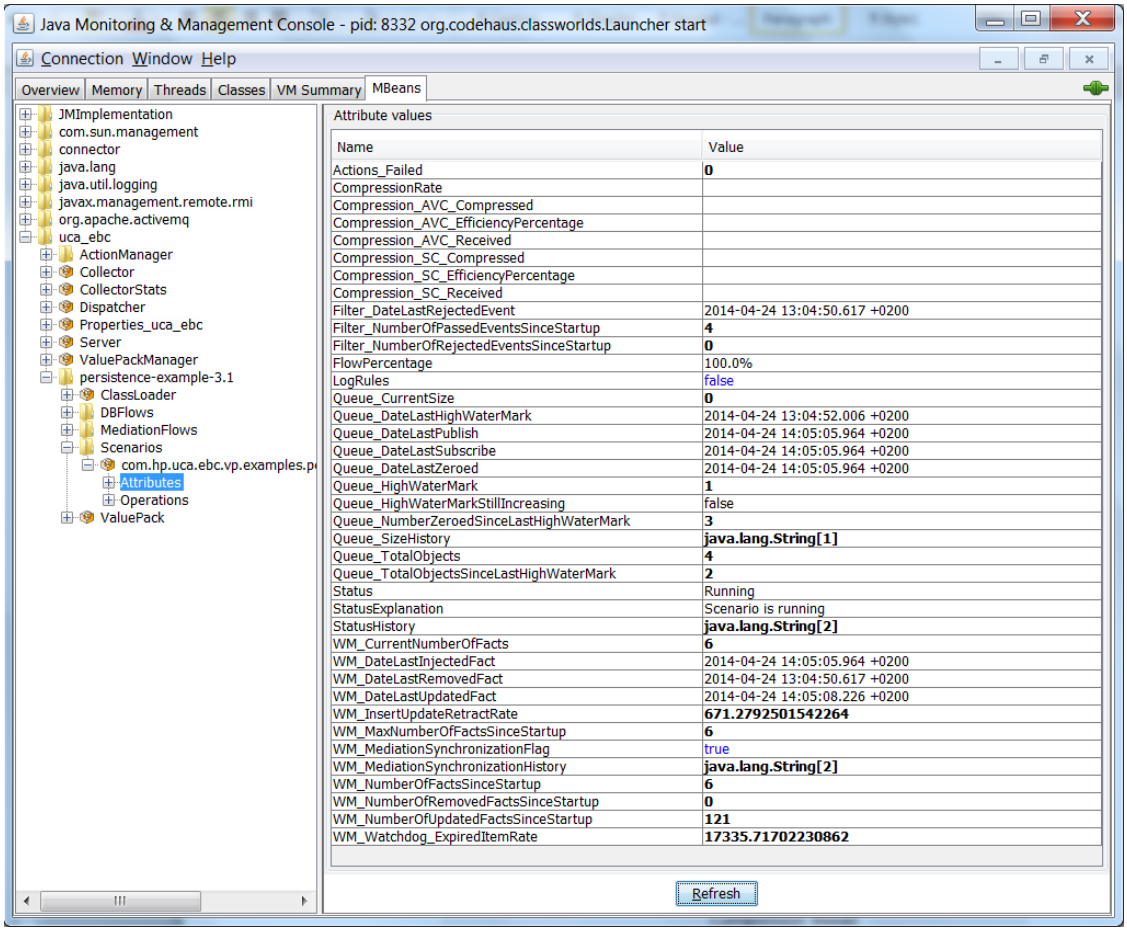


Figure 23: Java JMX Console: UCA for EBC Value Pack – Scenario - Attributes

Any Scenario of a UCA for EBC Value Pack can be monitored at the Java JMX console using both attributes and operations.

The following table lists the attributes of any Scenario of a UCA for EBC Value Pack that are shown on the Java JMX console:

Table 39: Java JMX Console: UCA for EBC Value Pack – Scenario - Attributes

Attribute name	Settable	Description
Actions_Failed	No	The number of failed actions for the scenario
Compression_AVC_Compressed	No	The number of AVC (Attribute Value Change) events compressed by the Compression thread
Compression_AVC_EfficiencyPercentage	No	The efficiency percentage of the Compression Thread regarding AVC (Attribute Value Change) events
Compression_AVC_Received	No	The number of AVC (Attribute Value Change) events received
Compression_SC_Compressed	No	The number of SC (State Change) events compressed by the Compression thread
Compression_SC_EfficiencyPercentage	No	The efficiency percentage of the Compression Thread regarding SC (State Change) events
Compression_SC_Received	No	The number of SC (State Change) events received
Filter_DateLastRejectedEvent	No	The Date and Time of the last event that was rejected by the scenario filter

Filter_NumberOfPassedEventsSinceStartup	No	The number of events that passed the scenario filters since start-up
Filter_NumberOfRejectedEventsSinceStartup	No	The number of events rejected by the scenario filters since start-up
FlowPercentage	No	Percentage of events inserted into Working Memory compared to the total of events received by the Scenario
LogRules	Yes	Flag (true/false) indicating whether scenario specific Drools engine logging is enabled/disable for the scenario
Queue_CurrentSize	No	The current size (in number of events) of the scenario events queue
Queue_DateLastHighWaterMark	No	The date and time of the last high water mark of the Scenario events queue
Queue_DateLastPublish	No	Date and time of the last time an event was added to the Scenario events queue
Queue_DateLastSubscribe	No	Date and time of the last time an event was removed from the Scenario events queue to be processed
Queue_DateLastZeroed	No	The date and time of the last time the Scenario events queue was empty
Queue_HighWaterMark	No	The value of the high water mark of the Scenario events queue (in number of events)
Queue_HighWaterMarkStillIncreasing	No	Whether the high water mark of the Scenario events queue is still increasing or not
Queue_NumberZeroedSinceLastHighWaterMark	No	The number of times that the Scenario events queue was empty since the last high water mark
Queue_SizeHistory	No	The history of the Scenario events queue size
Queue_TotalObjects	No	The total number of "objects" that have been added to the Scenario events queue since start-up
Queue_TotalObjectsSinceLastHighWaterMark	No	The total number of "objects" that have been added to the Scenario events queue since the last high water mark
Status	No	The status of the Scenario, either: • Starting • Running • Degraded • Failed • Stopped • Unknown
StatusExplanation	No	An explanation for the status of the Scenario
StatusHistory	No	The full history of the Scenario statuses, since it was first started
WM_CurrentNumberOfFact	No	The current number of facts in the Drools Working Memory of the Scenario
WM_DateLastInjectedFact	No	Date and time of the last fact inserted into the Drools Working Memory of the Scenario
WM_DateLastRemovedFact	No	Date and time of the last fact removed from the Drools Working Memory of the Scenario

WM_DateLastUpdatedFact	No	Date and time of the last fact updated in the Drools Working Memory of the Scenario
WM_InsertUpdateRetractRate	No	The rate of operations (insert/update/retract fact) on the Drools Working Memory of the Scenario in operations per second
WM_MaxNumberOfFactsSinceStartup	No	The maximum number of facts in the Drools Working Memory of the Scenario since start-up
WM_MediationSynchronizationFlag	No	The value of the Mediation Synchronization Flag: True (i.e. the mediation flow is synchronized) False (i.e. the mediation flow is currently undergoing a synchronization)
WM_MediationSynchronizationHistory	No	The history of the synchronization status of the mediation flow
WM_NumberOfFactsSinceStartup	No	The number of facts that have been inserted into the Drools Working Memory of the Scenario since start-up
WM_NumberOfRemovedFactsSinceStartup	No	The number of facts that have been removed from the Drools Working Memory of the Scenario since start-up
WM_NumberOfUpdatedFactsSinceStartup	No	The number of facts that have been updated in the Drools Working Memory of the Scenario since start-up

The following screenshot shows the operations available for a Scenario sub-folder of a UCA for EBC Value Pack at the Java JMX Console:

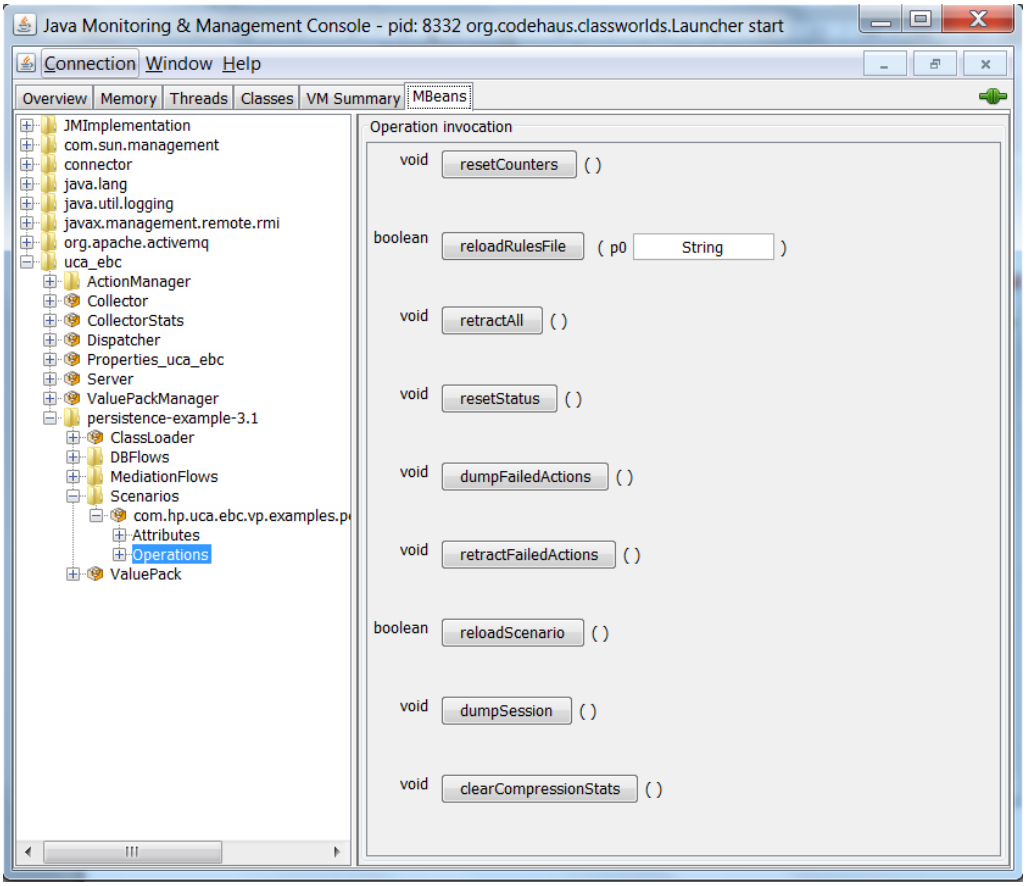


Figure 24: Java JMX Console: UCA for EBC Value Pack – Scenario - Operations

The following table lists the operations that can be executed on any Scenario of a UCA for EBC Value Pack using the Java JMX console:

Table 40: Java JMX Console: UCA for EBC Value Pack – Scenario - Operations

Operation name	Explanation
resetCounters()	Resets the statistics for the Scenario
reloadRulesFile(String)	Reload a specific Rules File of the Scenario Parameter 1: The name of the Rules File
retractAll()	Clears the Drools Working Memory of the Scenario
resetStatus()	Resets the status of the Scenario
dumpFailedActions()	Dump all failed actions for the Scenario
retractFailedActions()	Retracts all failed actions from the Drools Working Memory of the Scenario
reloadScenario()	Reloads all rules files of the Scenario
dumpSession()	Dumps the Drools Working Memory of the Scenario
clearCompressionStats()	Resets the statistics regarding Compression

Chapter 6
UCA for EBC Advanced Troubleshooting

6.1 UCA for EBC Logging Mechanism

The UCA for EBC logging feature is based on the log4j technology.

The main application logging mechanism is driven by the setting of the `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` log4j configuration file.

Some other (specific) logging levels can be activated by setting some properties in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file. These additional logging levels are:

- Scenario rule execution log:
  That allows logging scenarios rules execution in a dedicated file in order to help debugging.
- Collector log:
  That allows logging all alarms collected in a specific file.

The generated log files are located in the `${UCA_EBC_INSTANCE}/logs` directory.

NOTE: Changes to the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file require a restart of UCA for EBC Server in order for the changes to be taken into account.

Changes to the `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` file require either a reload of the Log4J configuration (through the uca-ebc-admin command-line tool, or the UCA for EBC User Interface) or a restart of UCA for EBC Server in order for the changes to be taken into account.

6.1.1 Standard application logging

Application logging is controlled by the `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` log4j configuration file.

The CONSOLE, FILE, and DB appenders are used for controlling application logging to the console, standard application log file or UCA for EBC User Interface. The standard application log file is the following (by default): `${UCA_EBC_INSTANCE}/logs/uca-ebc.log`.

The `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` can be modified to control:

- what kind of events get logged
- what is the trace level for each event type (event type are defined by Java package names)
- where the events are logged (what appenders are used)

The provided `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` file predefines a set of application classes for which the logging can be activated or not.

6.1.2 Collector logging

With UCA for EBC 3.4, events and alarms can still be collected through the OSS Open Mediation UCA for EBC Channel adapter, as in previous releases; but they can also be collected through the UCA for EBC UMB Adapter.

In both cases, UCA for EBC offers the possibility to log the collected alarms or events into a file exactly how they were received.

6.1.2.1 Events received through the OSS Open Mediation UCA for EBC Channel Adapter

The Collector raw logging feature is the possibility to log in a file the exact alarm list that is received by the collector through the UCA for EBC Channel Adapter

This logging feature can be enabled/disabled at application start-up by setting the collector.logger.enabled property to `true` or `false` in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file.

By setting this property to `true` all alarms going through the Collector will be dumped in either one of the following files before any other treatment if done on the received alarms:

- the `${UCA_EBC_INSTANCE}/logs/uca-ebc-collector.log` file for alarms that are not rejected by the Collector
- the `${UCA_EBC_INSTANCE}/logs/uca-ebc-collector-rejected.log` file for alarms that are rejected by the Collector

Alarms can be rejected by the Collector for either one of the following reasons:

- The JMS message containing the alarms does not have the proper body format: the expected JMS message body format expected by the Collector is Text
- The content of the JMS message cannot be converted to Alarm objects because the XML format of the alarms inside the JMS message is not compliant with the UCA for EBC Alarm format defined in the `${UCA_EBC_HOME}/schemas/uca-expert-alarm.xsd` file
- Collector message validation is turned on (the `collector.messages.validation` property is set to `true` in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file), and the alarms in the JMS message received by the Collector failed validation

Alarms are dumped directly in XML format in the `uca-ebc-collector.log` file. On the other hand, the `uca-ebc-collector-rejected.log` file has the format of a log file.

6.1.2.2 Events received through the UMB UCA Mediation Adapter

Events (including alarms), received by UCA for EBC through the UCA for EBC UMB Adapter will be logged in the following format:

```
<myEvent>
  <identifier>1</identifier>
  <value>23</value>
</myEvent>
```

Event as received by UCA through the UMB UCA mediation adapter

```
<EventBoxBase
xmlns="http://hp.com/uca/expert/event"
eventClassName="myEvent Class Name">
  <eventString><![CDATA[
    <myEvent>
      <identifier>1</identifier>
      <value>23</value>
    </myEvent>
  ]]>
  </eventString>
</EventBoxBase>
```

Event as logged by UCA to the uca-ebc-received-event.log

This logging feature can be enabled/disabled at application start-up by setting the **received.events.logger.enabled** property to `true` or `false` in the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file.

By setting this property to `true` all events going through the UCA for EBC UMB Adapter will be dumped in the following file before any other treatment is done on the received alarms:

`${UCA_EBC_INSTANCE}/logs/uca-ebc-received-events.log`

6.1.3 Scenario logging

6.1.3.1 Scenario logging

In order to be able to configure how log messages coming from the Scenario rule files (drl files) are processed (what trace level and appenders are used), a specific logger must be added to the `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` configuration file.

This logger is defined as follows:

```
<logger name="<scenario name>" additivity="false">
       <level value="INFO" />
       <appender-ref ref="CONSOLE" />
       <appender-ref ref="DB" />
 </logger>
```

Where `<scenario name>` is the name of the scenario for which you want to configure the logging. The `<scenario name>` has to be identical to the `<scenario name>` defined in the `ValuePackConfiguration.xml` file of your Value Pack.

The definition of your scenario specific logger can be added to the "Detailed Traces for Value Pack Scenarios" section of the `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` file. This section is identified by comments in the file.

The following screenshot shows an example of how to configure specific logging in the `uca-ebc-log4j.xml` file:

```
77     <!-- ######################################## -->
78     <!-- Detailed Traces for Value Pack Scenarios -->
79     <!-- ######################################## -->
80
81   <!--
82       Uncomment the following in order to show
83       detailed traces for your value pack scenarios.
84       You need to update the name="myScenario" attribute
85       with the actual name of your scenario
86   -->
87   <!--
88    <logger name="myScenario" additivity="false">
89          <level value="INFO" />
90          <appender-ref ref="CONSOLE" />
91          <appender-ref ref="DB" />
92    </logger>
93   -->
94
95    <logger name="myScenario" additivity="false">
96          <level value="INFO" />
97          <appender-ref ref="CONSOLE" />
98          <appender-ref ref="DB" />
99    </logger>
100
101 <!--
102     <logger name="com.hp.uca.expert.vp.pd.ProblemDetection" additivity="false">
103         <level value="DEBUG" />
104         <appender-ref ref="CONSOLE" />
```

Figure 25: Configuring scenario specific logging in the uca-ebc-log4j.xml file

6.1.3.2 Scenario exceptions logging

It is also possible to define a specific logger (one for each scenario) in the *${UCA_EBC_INSTANCE}*/conf/uca-ebc-log4j.xml configuration file for logging the exceptions thrown in the action part of the rules of a scenario.

By default, these exceptions are logged using the scenario logger as defined in the previous chapter: 6.1.3 "Scenario logging".

If you want exceptions log messages to be handled by a specific logger different from the scenario logger, you can define it in the uca-ebc-log4j.xml configuration file. The logger should be named "myScenario.exceptions" (change myScenario to the actual name of your scenario as per the ValuePackConfiguration.xml file).

The following screenshot shows an example of how to configure a specific scenario exception logger in the uca-ebc-log4j.xml file:

```
<!--
    Uncomment the following in order to show
    detailed traces for your value pack scenarios rules exceptions.
    You need to update the name="myScenario.exceptions" attribute
    with the actual name of your scenario + ".exceptions"
-->

 <logger name="myScenario.exceptions" additivity="false">
       <level value="INFO" />
       <appender-ref ref="CONSOLE" />
         <appender-ref ref="FILE" />
         <appender-ref ref="DB" />
 </logger>
```

Figure 26: Configuring scenario exceptions specific logging in the uca-ebc-log4j.xml file

In versions of UCA for EBC prior to UCA for EBC 3.4, these scenario exceptions were logged using either "com.hp.uca.expert.scenario.internal.ScenarioImpl" or "com.hp.uca.expert.watchdog.WatchdogThread" loggers depending on whether the Scenario Thread or Watchdog Thread was executing the rules when the exception occurred.

With to UCA for EBC 3.4 onward, these scenario exceptions are now logged to "myScenario.exceptions".

There's some commented XML code in the uca-ebc-log4j.xml file delivered with UCA for EBC 3.4 that can be used to easily create a "myScenario.exceptions" logger.

NOTICE:
Please refer to section 3.2.3 "uca-ebc-log4j.xml file configuration" to learn more about the configuration of the ${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml file.

6.1.3.3 Scenario rule execution logging

Rule execution can be logged per scenario in a dedicated log file. Logging can be enabled/disabled at application start-up by setting the **engine.logger.enabled** property to true/false in the ${UCA_EBC_INSTANCE}/conf/uca-ebc.properties file.

This property controls scenario specific rule execution logging for all scenarios.

Properties like **engine.logger.interval** (which controls the interval in milliseconds at which rule execution information is written to the log file) can also be set. These properties affect all scenario specific rule execution log files.

NOTICE:

Please refer to section 3.2.1 "uca-ebc.properties file configuration" especially Table 16 "Rule Engine logger properties in the uca-ebc.properties file" for more information on how to configure the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file.

Changes to the `${UCA_EBC_INSTANCE}/conf/uca-ebc.properties` file require a restart of UCA for EBC Server in order for the changes to be taken into account.

Scenario-specific rule execution log files are named `logEngine_<scenario name>.log` and are located in the `${UCA_EBC_INSTANCE}/logs` directory. Scenario-specific engine log files contain standard Drools engine log entries specific to a scenario.

At runtime, it is also possible to enable/disable scenario specific rule execution logging for just one scenario by using either the **uca-ebc-admin** command-line tool or the Java console.

Below is a screenshot showing how to enable/disable scenario specific rule execution logging for just one scenario by using the Java console:

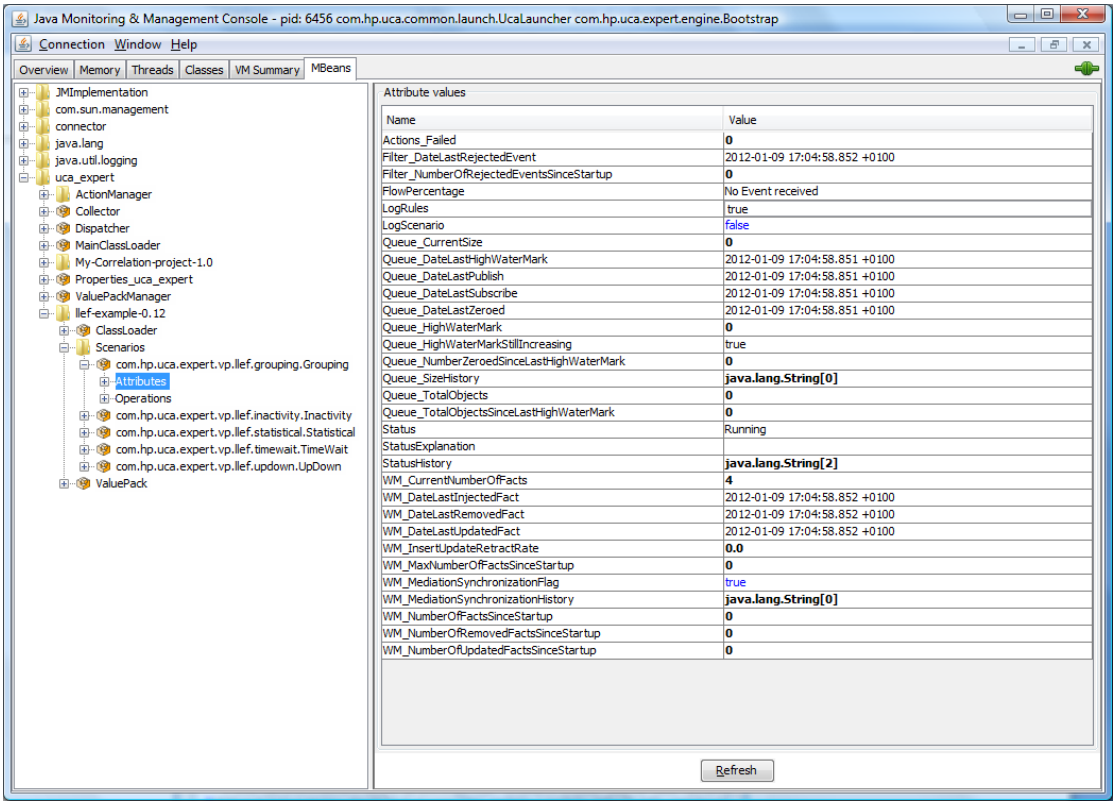


Figure 27: Java JMX Console: Enabling/Disabling scenario specific rule execution logging for one scenario

Scenario specific rule execution log files are compatible with the JBoss Rule Audit feature in Eclipse IDE.

The JBoss Rule Audit panel comes with the JBoss Drools Eclipse plugin. You can view this panel by selecting the JBoss Drools perspective in Eclipse IDE as shown below. The JBoss Rule Audit panel should be part of the JBoss Drools perspective unless it has been removed.

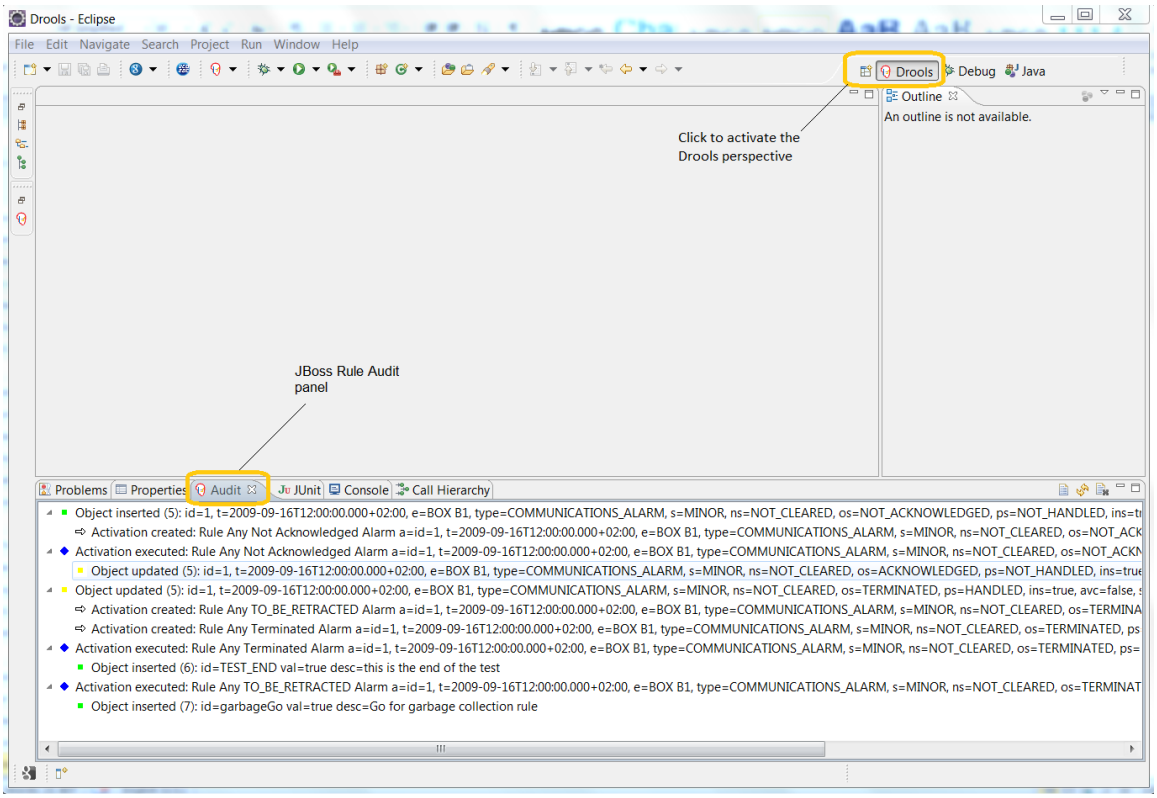
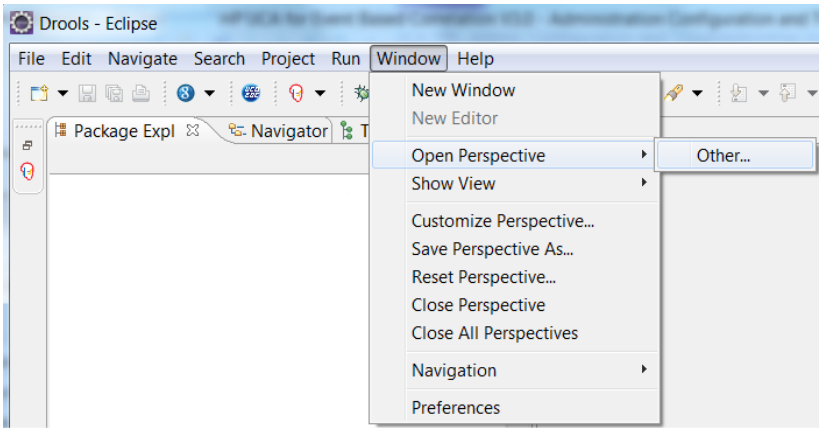


Figure 28: Selecting the JBoss Drools perspective in Eclipse IDE by clicking on the JBoss Drools perspective icon

Alternatively, you can switch to the JBoss Drools perspective by going to the "Window" -> "Open Perspective" Eclipse IDE top menu, and selecting the "Drools" perspective, as shown below.



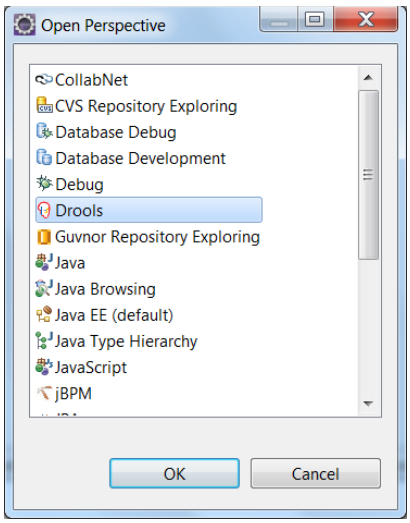


Figure 29: Selecting the JBoss Drools perspective in Eclipse IDE by using the Eclipse IDE menus

If the Drools Audit panel is not shown, you can select it by going to the "Window" -> "Show View" Eclipse IDE top menu, and selecting the "Audit" view from the Drools group.

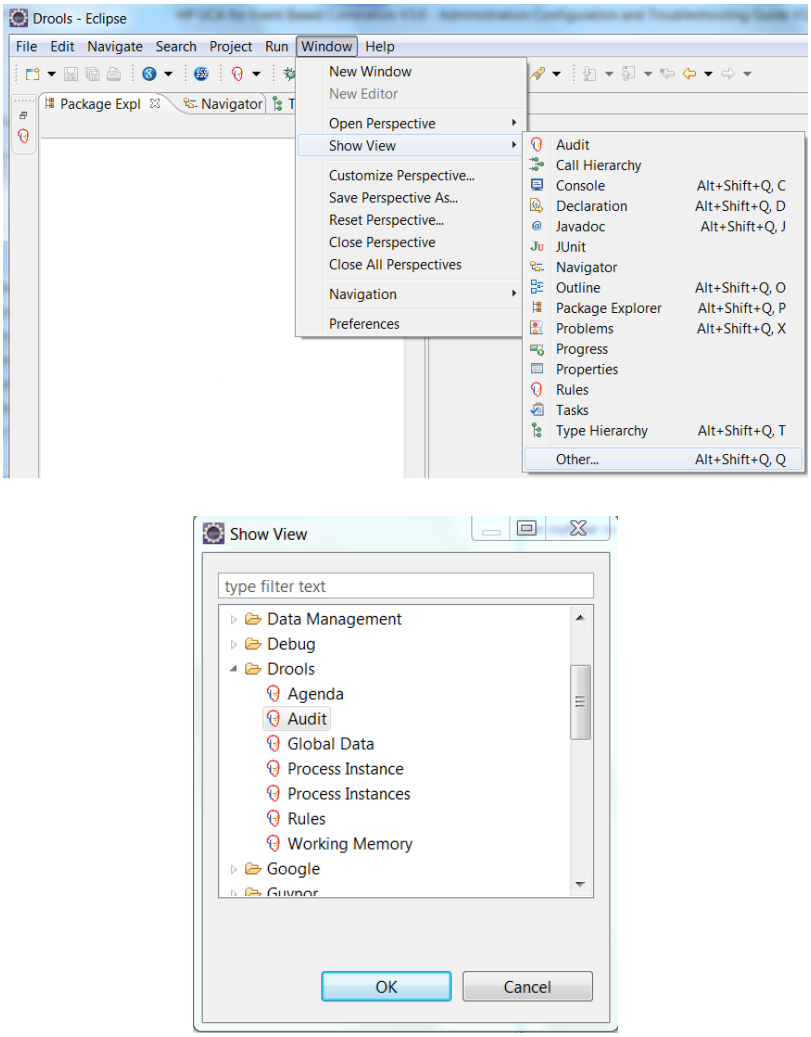


Figure 30: Showing the JBoss Drools Audit view in Eclipse IDE

To display the contents of a scenario specific rule execution log file using Eclipse IDE, you need to load the file inside the Audit panel.

You can open a `logEngine_<scenario name>.log` file in the Audit panel by using drap and drop of the file into the Audit panel as shown in the screenshot below.

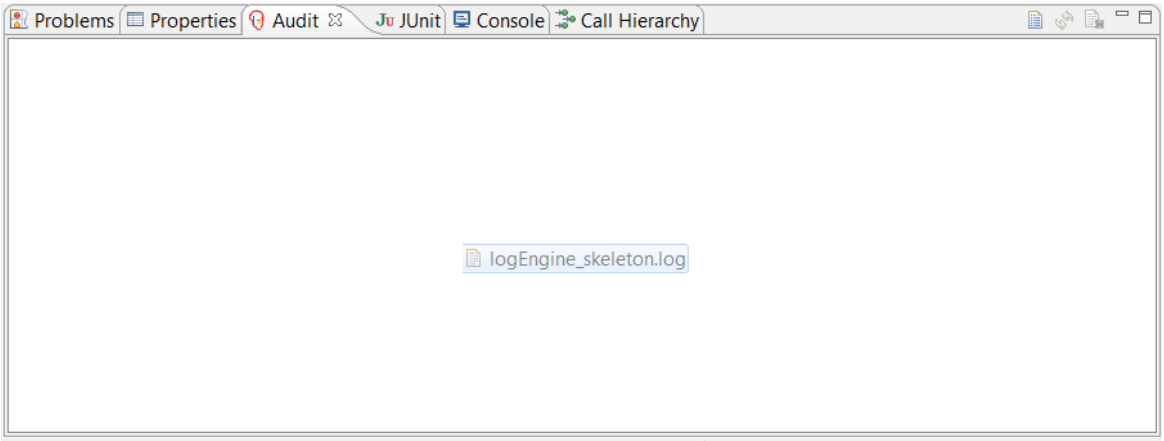


Figure 31: Eclipse IDE: Using drag and drop to open a Drools engine log file in the Drools Audit panel

Alternatively you can open a Drools engine log file in the Drools Audit panel by clicking on the "Open log" icon of the Drools Audit panel as show below:

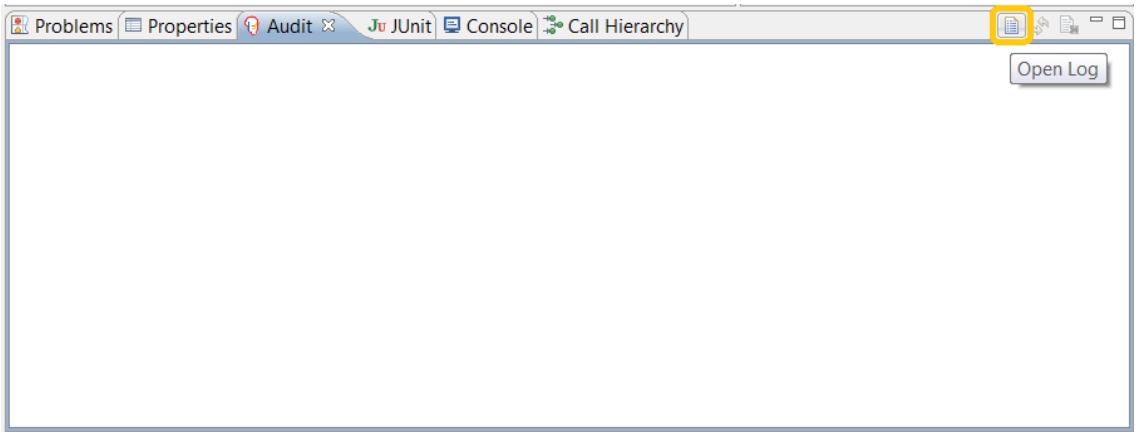


Figure 32**: Eclipse IDE: Using the "Open log" icon to open a Drools engine log file i**n the Drools Audit panel

The following screenshot shows an example of how contents of a scenario specific rule execution log file is displayed in the Audit panel of the Drools perspective in Eclipse IDE:

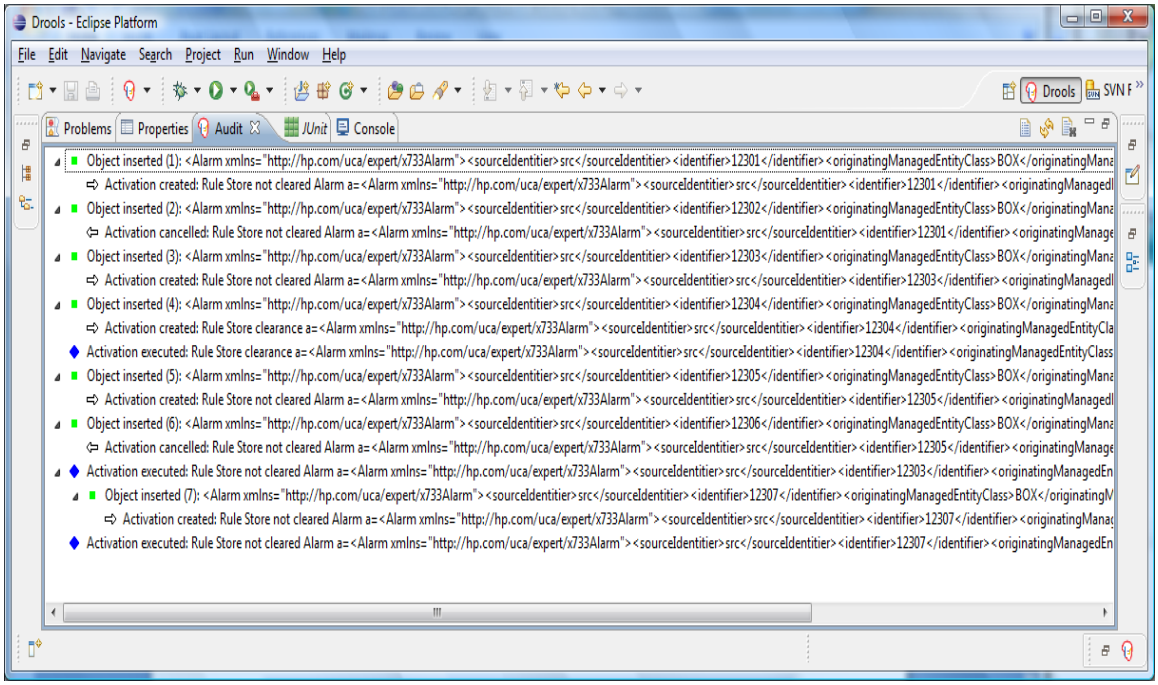


Figure 33: Eclipse IDE: Viewing scenario rule execution logs

Scenario specific rule execution log files contain Drools rule activation information in addition to the insertion/update/deletion of objects in Drools working memory.

Besides the Audit panel, the Drools perspective in Eclipse IDE also provides the Agenda and Working Memory panels which give information on the planned rule execution schedule (Agenda panel) and the list of all the objects in the Working Memory (Working Memory panel) of a Drools Engine.

You can select the Agenda or Working Memory panels by either switching to the Drools perspective or going to the "Window" -> "Show View" Eclipse IDE top menu, and selecting the "Agenda" or "Working Memory" view from the Drools group, as shown below.

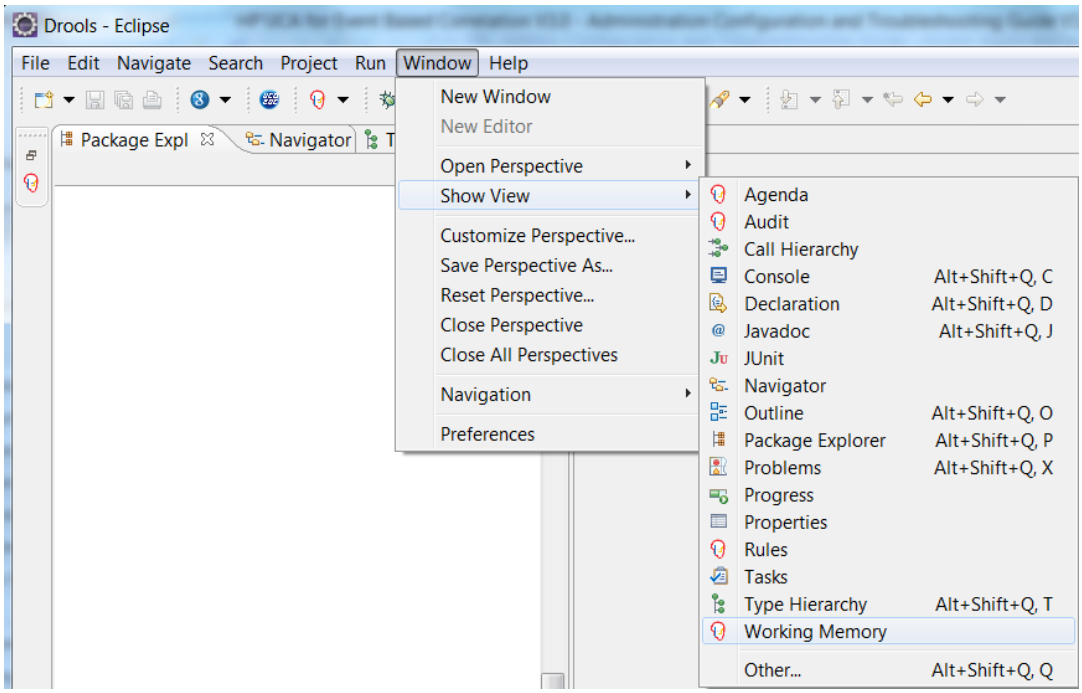


Figure 34: Showing the JBoss Drools Agenda or Working Memory view in Eclipse IDE

The Drools Agenda and Working Memory views are useful in debug mode in Eclipse, for example, when running the JUnit tests of a Value Pack in debug mode in Eclipse. You put breakpoints in either the rules or java code of a Value Pack (by double-clicking left of the line number of a line of rules or java code) then execute the JUnit tests of a Value Pack in debug mode by right-clicking on the JUnit test file and selecting the "Debug As" -> "Drools JUnit Test" context menu item, as shown below

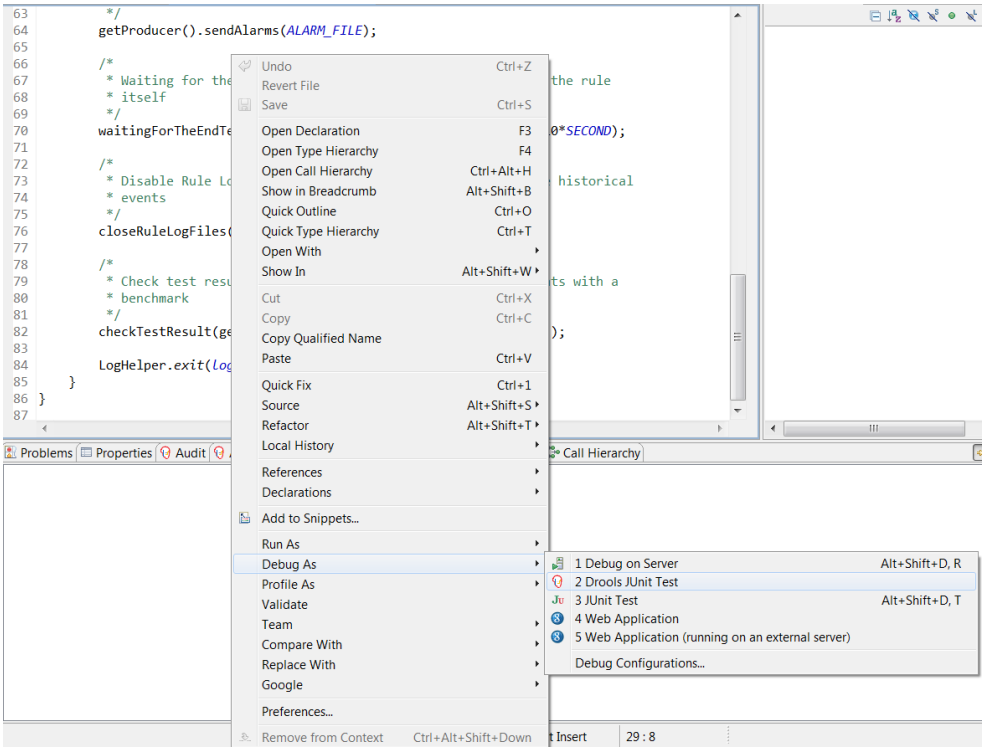


Figure 35: Running a JUnit Test of a Value Pack in debug mode in Eclipse IDE

The execution will pause once the first breakpoint is encountered. Once the execution is paused you can inspect the contents of the Drools Working Memory by looking at the Working Memory panel, as shown below:

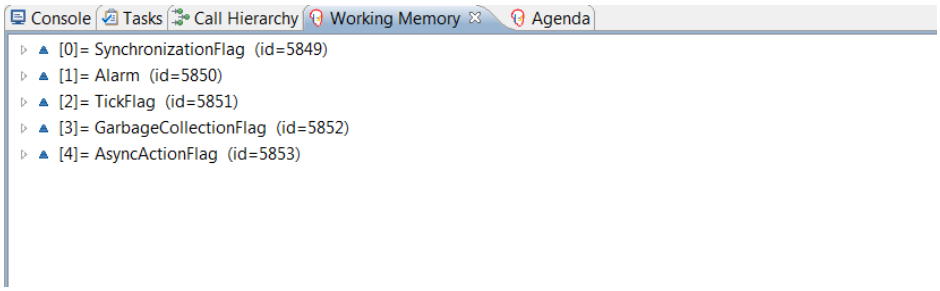


Figure 36: Sample view of the Drools Working Memory panel in Eclipse IDE

The Drools Working Memory panel gives information on the list of all the objects in Working Memory: Alarms, Flags, custom objects ...

You can also inspect the Drools Agenda by looking at the Agenda panel, as show below:

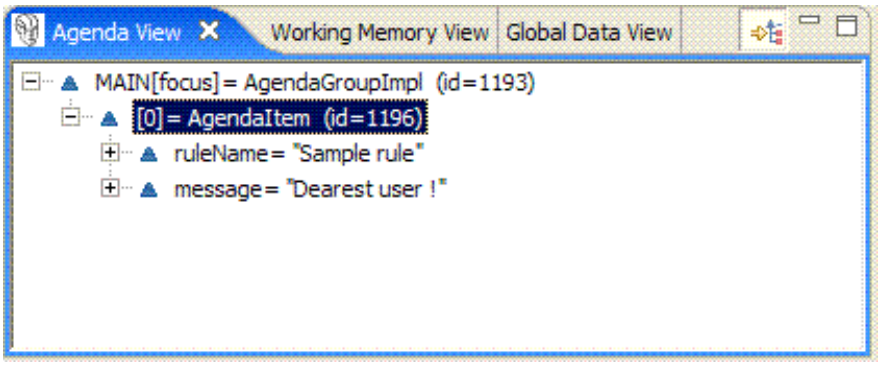


Figure 37: Sample view of the Drools Agenda panel in Eclipse IDE

The Drools Agenda panel gives information on the planned rule execution schedule.

NOTICE: The Drools perspective in Eclipse IDE is provided by Drools plug-in for Eclipse. For more information on how to install the Drools plug-in for Eclipse IDE please refer to: [R2] `HPE UCA for EBC Value Pack Development Guide`

6.1.4 Drools logging

6.1.4.1 Configuring the log for Working Memory Agenda and Event Listeners

In the `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` Log4J configuration file for UCA for EBC, you can configure the log level and appender references for two classes that monitor Drools Engine Agenda and Drools Working Memory for all the scenarios of all the Value Packs running on UCA for EBC.

You can configure the log for these two classes by updating the following section in the `${UCA_EBC_INSTANCE}/conf/uca-ebc-log4j.xml` Log4J configuration file:

```
268         <appender-ref ref="CONSOLE" />
269         <appender-ref ref="FILE" />
270         <appender-ref ref="DB" />
271     </logger>
272     <logger name="com.hp.uca.expert.engine.rulesession.WMAgendaEventListener"
273         additivity="false">
274         <level value="DEBUG" />
275         <appender-ref ref="CONSOLE" />
276         <appender-ref ref="FILE" />
277         <appender-ref ref="DB" />
278     </logger>
279
280     <logger name="com.hp.uca.expert.engine.rulesession.WMEventListener"
281         additivity="false">
282         <level value="DEBUG" />
283         <appender-ref ref="CONSOLE" />
284         <appender-ref ref="FILE" />
285         <appender-ref ref="DB" />
286     </logger>
287
288     <logger name="com.hp.uca.expert.lifecycle.LifeCycleAnalysis"
289         additivity="false">
290         <level value="DEBUG" />
291         <appender-ref ref="CONSOLE" />
```

Figure 38: Configuring the log for Working Memory Agenda and Event Listeners

Setting the log level to DEBUG for the WMAgendaEventListener will add log messages to the log(s) every time the Agenda of the Drools Engine of a Scenario is updated, i.e. when:

- Rule activations are created
- Rule activations are canceled
- Before rules are fired
- After rules are fired

Setting the log level to DEBUG for the WMEventListener will add log messages to the log(s) every time the Working Memory of the Drools Engine of a Scenario is updated, i.e. when:

- Objects are inserted into Working Memory
- Objects are updated in Working Memory
- Objects are retracted from Working Memory

NOTICE: Enabling these logs can be complementary to using the scenario specific Drools engine logs that are described in section: 6.1.2 “Collector logging”

6.2 Managing the Drools engine(s)

Each scenario has its own Drools rule engine for processing the Drools rules defined in the rules files of the scenario. The following operations can be performed on the working memory of a scenario, without having to restart either UCA for EBC or any Value Pack:

- Dumping the Working Memory
- Clearing the Working Memory
- Reloading the Rules

6.2.1 Dumping the Working Memory

Dumping the Working Memory of a scenario dumps the complete list of object (Facts) currently in the working memory of a Scenario to the log(s).

Dumping the Working Memory of a scenario can be performed using the Java JMX Console at the Scenario level by going to the "MBeans" tab of the Java Console and navigating to the "`uca_ebc/<value pack name>-<value pack version>/scenarios/<scenario name>/operations`" folder.

The following screenshot shows how to dump the working memory at the scenario level:

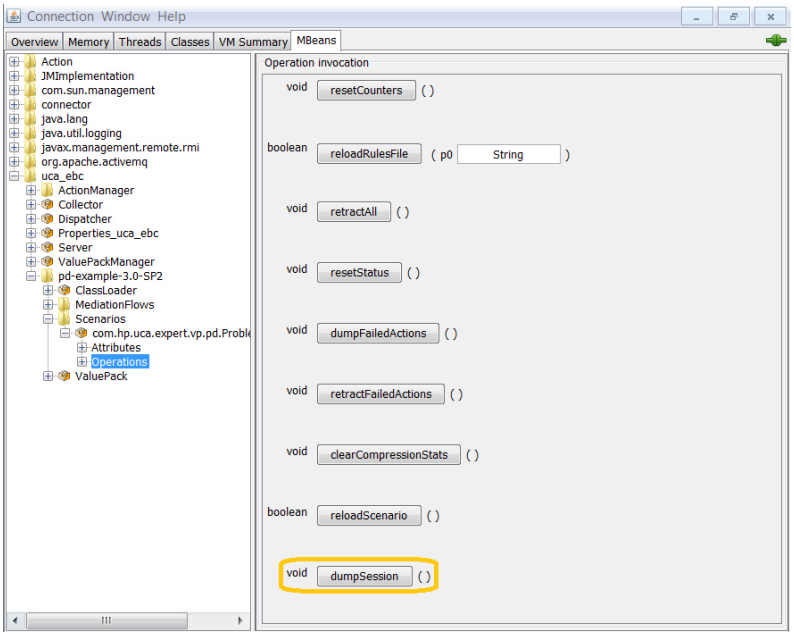


Figure 39: Java JMX Console: Dumping the working memory of a Scenario

Dumping the Working Memory of a scenario can also be performed at the UCA for EBC User Interface in the Scenario / Monitoring panel, as shown in the following screenshot:

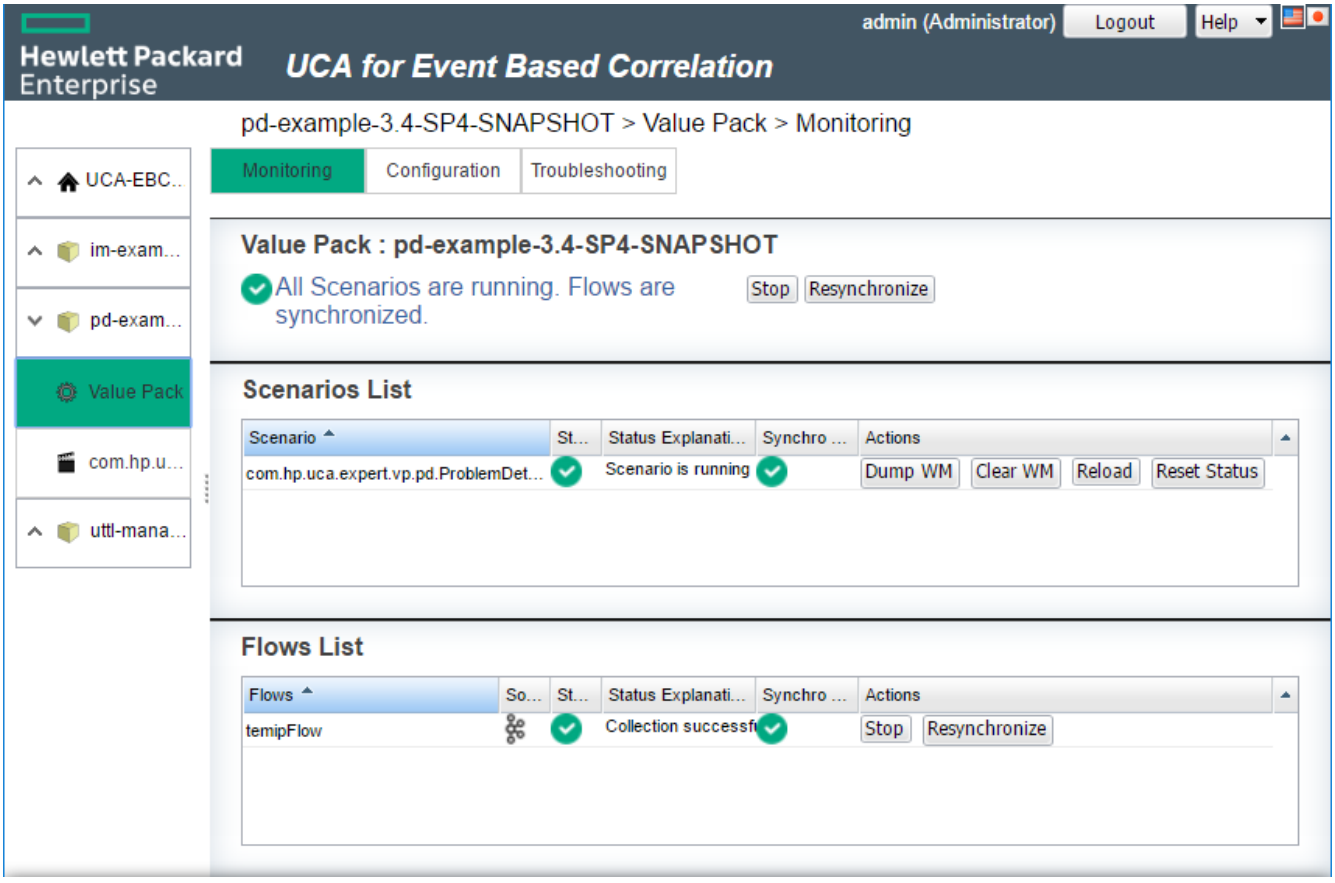


Figure 40: UCA for EBC User Interface: Dumping the working memory of a scenario

NOTICE:
For more information on the UCA for EBC User Interface, please refer to: [R3] `HPE UCA for EBC User Interface Guide`.
For more information on how to dump the working memory of a scenario using the Java JMX Console, please see the section: 5.1.3.3 "Monitoring UCA for EBC scenarios"

6.2.2 Clearing the Working Memory

Clearing the Working Memory of a scenario can be necessary at times when you want to start fresh with your scenario. This operation may or may not be followed by a resynchronization of the mediation flow of the Value Pack that the scenario belongs to, in case you need you scenario to receive the current list of events (Alarms) from the mediation layer or not.

Cleaning the Working Memory of a scenario can be performed using the Java JMX Console at the Scenario level by going to the "MBeans" tab of the Java Console and navigating to the "`uca_ebc/<value pack name>-<value pack version>/scenarios/<scenario name>/operations`" folder.

The following screenshot shows how to clear the working memory at the scenario level:

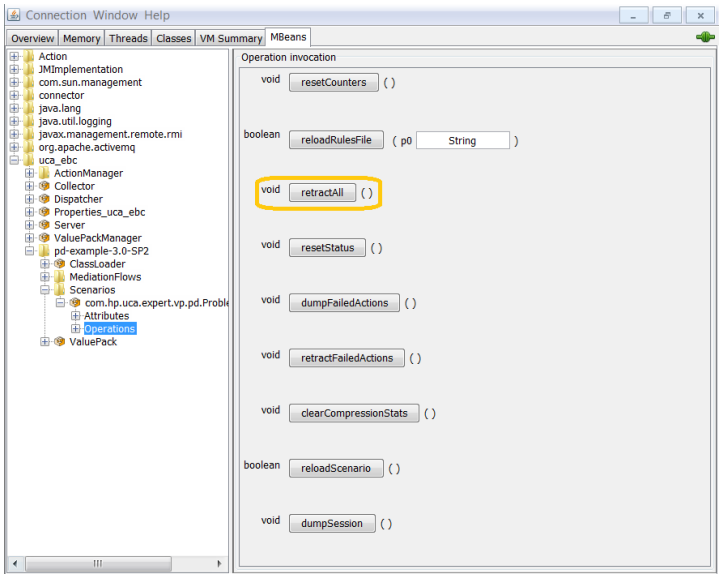


Figure 41: Java JMX Console: Clearing the working memory of a Scenario

Cleaning the Working Memory of a scenario can also be performed at the UCA for EBC User Interface in the Scenario / Monitoring panel, as shown in the following screenshot:

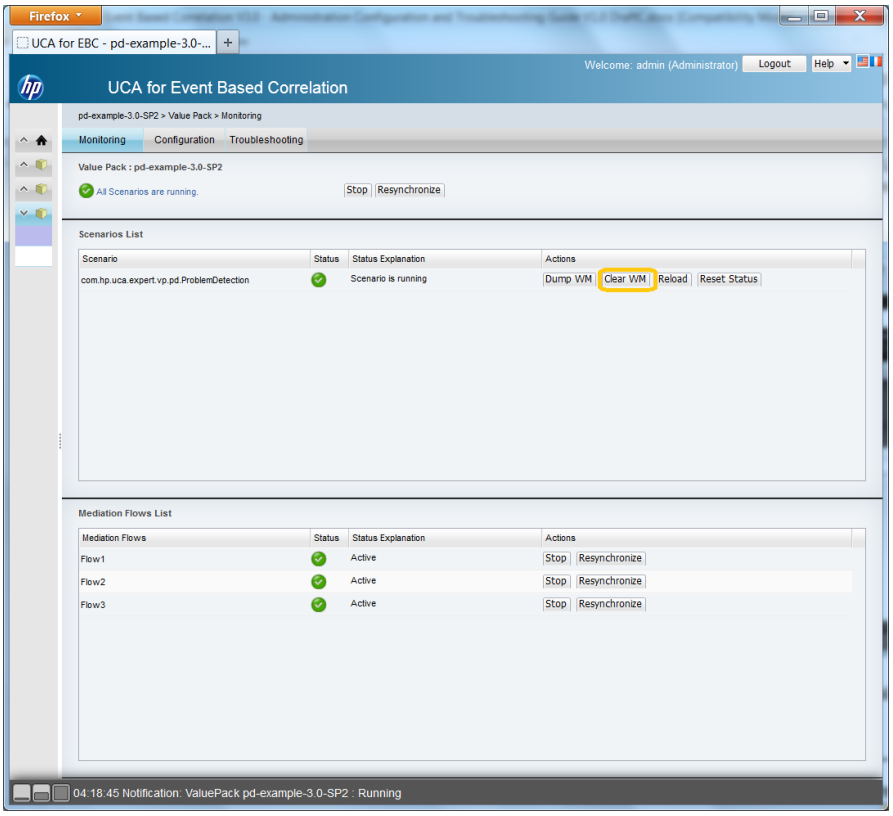


Figure 42: UCA for EBC User Interface: Clearing the working memory of a scenario

NOTICE:
For more information on the UCA for EBC User Interface, please refer to: [R3] `HPE UCA for EBC User Interface Guide`.
For more information on how to clear the working memory of a scenario using the Java JMX Console, please see the section: 5.1.3.3 "Monitoring UCA for EBC scenarios"

6.2.3 Reloading the rules

Each scenario of a Value Pack contains a list of Drools rules files or Drools template rules files (template rules file are similar to standard rules file but use an extra parameters file).

Each and all of the rules files (and template rules files) can be modified at runtime and reloaded without restarting UCA for EBC or any individual Value Pack so that the new rules files get used right away in the Drools engine of the scenario.

The process for reloading the rules files is the following:

- Update the rules files, template rules files, and template parameters files as you wish in the deployment directory of the Value Pack: `${UCA_EBC_INSTANCE}/deploy/<value pack name>-<value pack version>`
- Reload the rules of a scenario using either the uca-ebc-admin command-line tool (with the -r or --reload option), the Java JMX Console or UCA for EBC User Interface

Reloading the rules of a scenario can be performed using the Java JMX Console at the Scenario level by going to the "MBeans" tab of the Java Console and navigating to the "`uca_ebc/<value pack name>-<value pack version>/scenarios/<scenario name>/operations`" folder.

The following screenshot shows how to reload rules files at the scenario level:

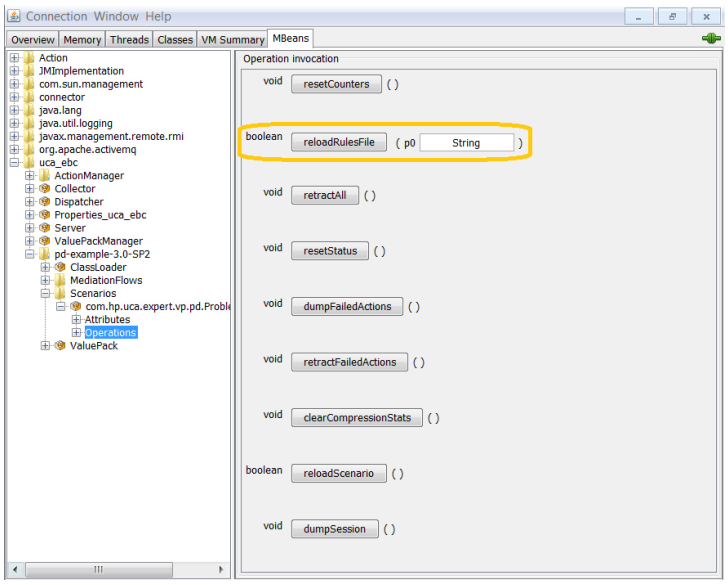


Figure 43: Java JMX Console: Reloading the rules of a Scenario

The same operation can be performed for all the rules files of all scenarios of one Value Pack, as shown in the following screenshot:

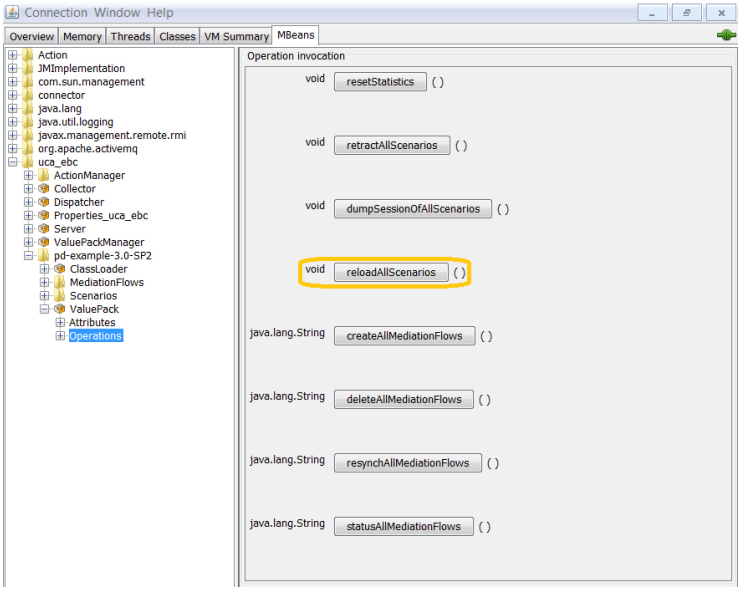


Figure 44: Java JMX Console: Reloading the rules of all Scenarios of a Value Pack

Reloading the rules of a scenario can also be performed at the UCA for EBC User Interface in the Scenario / Monitoring panel, as shown in the following screenshot:

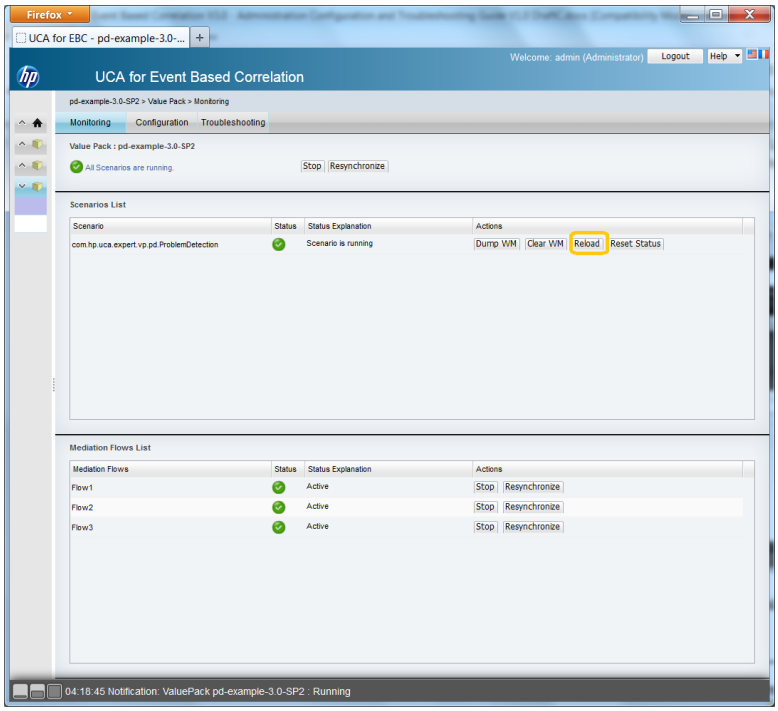


Figure 45: UCA for EBC User Interface: Reloading the rules of a Scenario

NOTICE:

For more information on the UCA for EBC User Interface, please refer to: [R3] `HPE UCA for EBC User Interface Guide`.

For more information on how to reload the rules of a scenario using the Java JMX Console, please see the section: 5.1.3.3 "Monitoring UCA for EBC scenarios"

For more information on how to reload the rules of a scenario using the uca-ebc-admin command-line tool, please see the section 2.2.3 "uca-ebc-admin".

6.3 Managing the flows and actions

6.3.1 Managing the DB flows

Each Value Pack can have one or more DB flows associated with it. Each DB flow represents a flow of events (Alarms) coming from a DB and going into the Value Pack and its scenarios.

DB flows are defined at the Value Pack level. All Scenarios of a Value Pack share the same DB flows.

6.3.1.1 Managing individual DB flows

The following operations can be performed on individual DB flows, without having to restart neither UCA for EBC nor the Value Pack (each operation only affects one DB flow):

- Start a DB flow (available in Java Console and UCA for EBC GUI)
- Stop a DB flow (available in Java Console and UCA for EBC GUI)
- Check the status of a DB flow (available in Java Console only)
- Resynchronize a DB flow (available in Java Console and UCA for EBC GUI)

The following screenshot shows how to perform these operations on individual DB flows using the Java console:

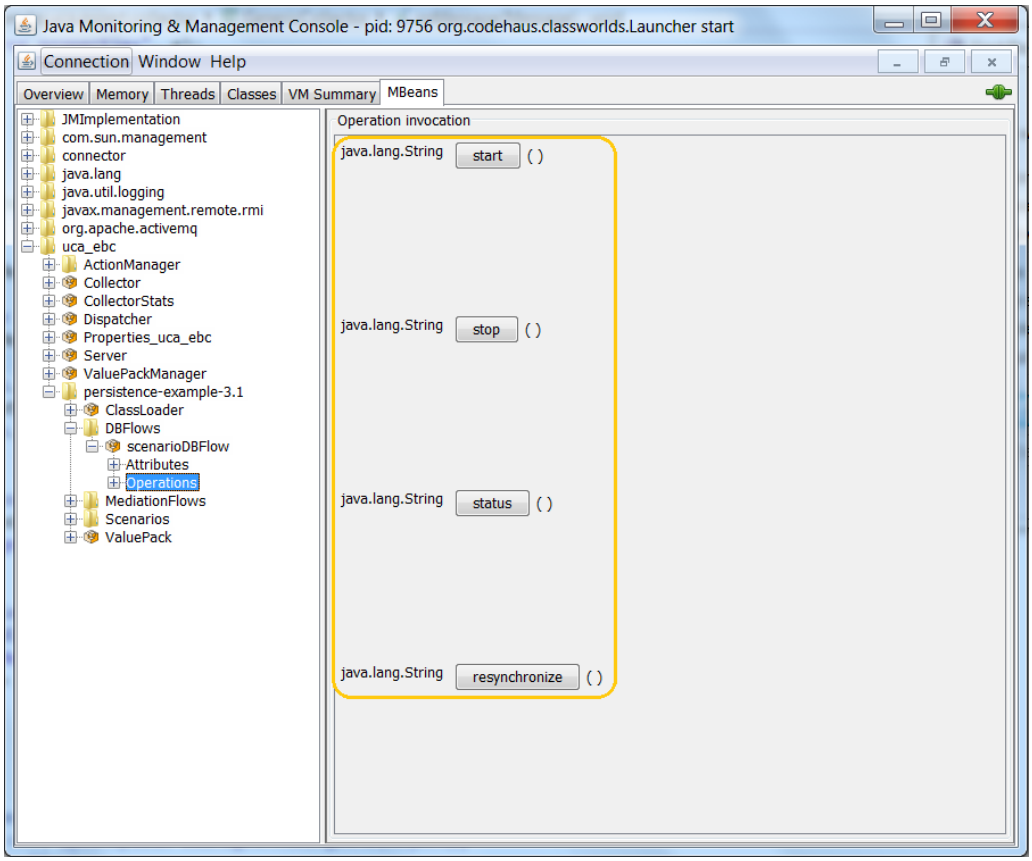


Figure 46: Java JMX Console: Performing operations on a single DB flow

It is possible to start, stop, and resynchronize DB flows using the UCA for EBC User Interface as shown in the following screenshot:

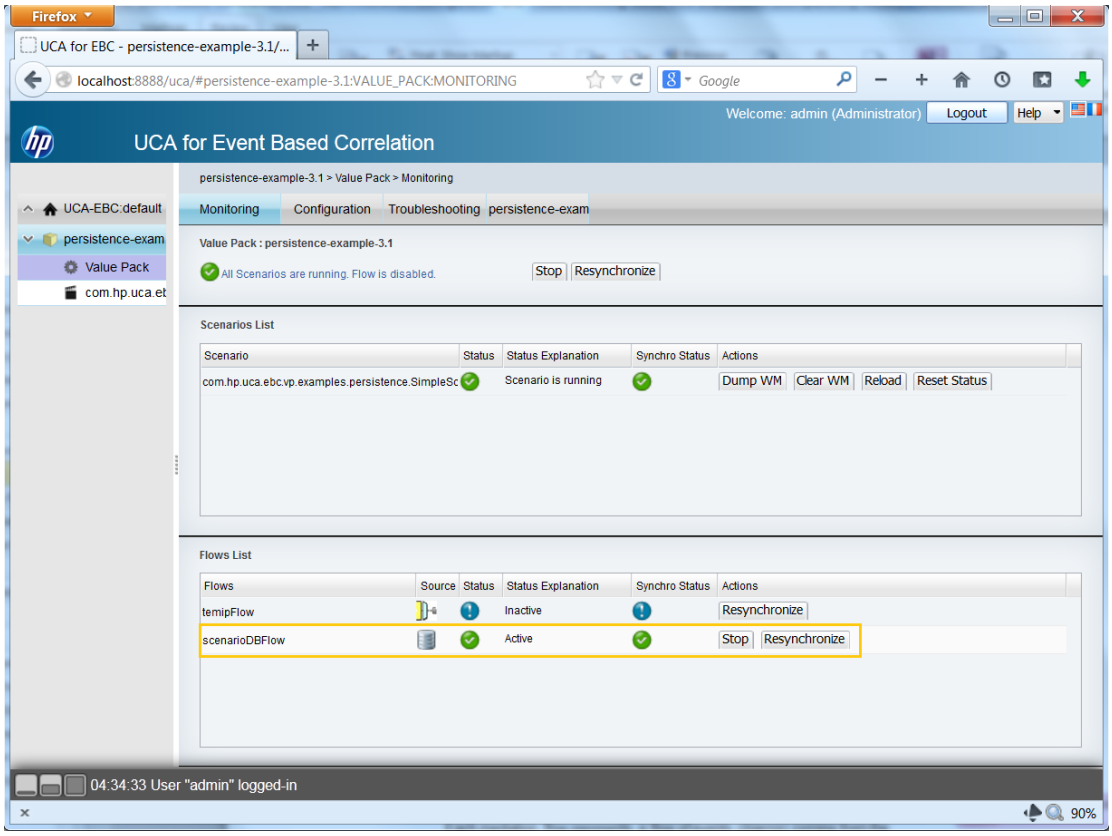


Figure 47: UCA for EBC User Interface: Performing operations on a single DB flow

6.3.2 Managing the mediation flows

Each Value Pack can have one or more mediation flows associated with it. Each mediation flow represents a flow of events (Alarms) coming from the mediation layer and going into the Value Pack and its scenarios.

Mediation flows are defined at the Value Pack level. All Scenarios of a Value Pack share the same mediation flows.

6.3.2.1 Managing the mediation flows at the value pack level

The following operations can be performed on the mediation flows of a Value Pack at the Value Pack level, without having to restart neither UCA for EBC nor the Value Pack (each operation affects all the mediation flows of the Value Pack at once):

- Create all the mediation flows (available in Java Console, and uca-ebc-admin tool)
- Delete all the mediation flows (available in Java Console, and uca-ebc-admin tool)
- Resynchronize all the mediation flows (available in Java Console, uca-ebc-admin tool and UCA for EBC GUI)
- Check the status of all the mediation flows (available in Java Console, and uca-ebc-admin tool)

The following screenshot shows how to perform these operation on the mediation flows at the value pack level using the Java console:

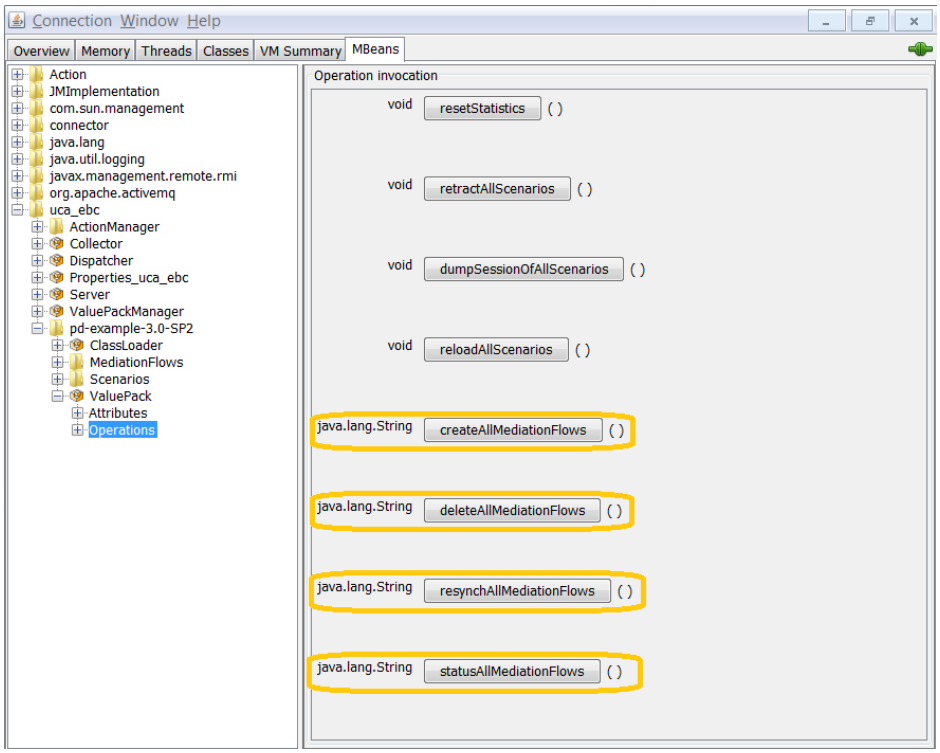


Figure 48: Java JMX Console: Performing operations on mediation flows at the Value Pack level

Resynchronizing the mediation flows is the only operation that can be performed at the value pack level on the mediation flows of a value pack using the UCA for EBC User Interface as shown in the following screenshot:

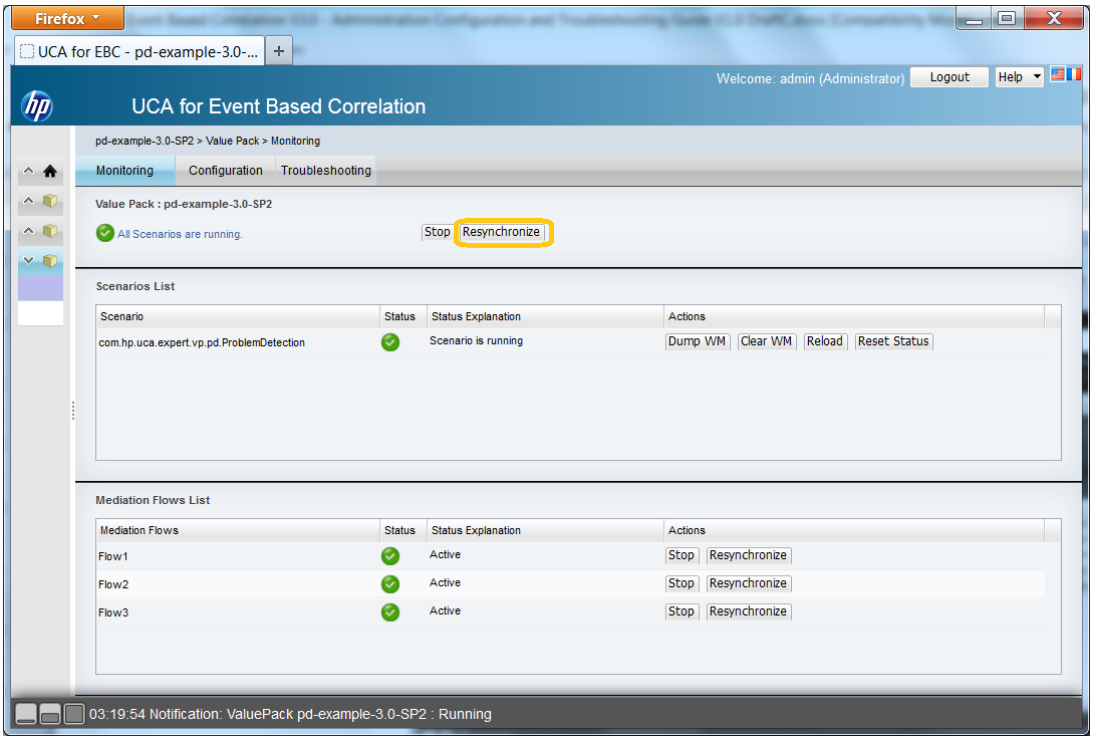


Figure 49: UCA for EBC User Interface: Resynchronizing the mediation flows of a Value Pack

Resynchronizing the mediation flows of a Value Pack can be necessary at times when you want to start fresh with your Value Pack and all its scenarios.

Mediation flows at defined at the Value Pack level in the *ValuePackConfiguration.xml* file of the Value Pack. Each Value Pack has its own mediation flows. As a consequence, resynchronizing the mediation flows of a Value Pack only affects the one Value Pack. All other Value Packs remain unaffected by the resynchronization.

When the mediation flows of a Value Pack are resynchronized, all the scenarios will receive the current list of events (Alarms) coming from the mediation layer. Usually, a resynchronization of the mediation flows is preceded by an operation to clear the Working Memory of all the scenarios of the Value Pack, so that:

- events (Alarms) are not duplicated in Working Memory, especially for scenarios that are in STREAM mode
- all scenarios can start fresh with both the complete current list of event from the mediation layer and an empty Working Memory

NOTICE:
For more information on the UCA for EBC User Interface, please refer to: [R3] `HPE UCA for EBC User Interface Guide`.
For more information on how to resynchronize the mediation flow for a value pack, please see the section 5.1.3.2 "Monitoring UCA for EBC value packs".

6.3.2.2 Managing individual mediation flows

The following operations can be performed on individual mediation flows, without having to restart neither UCA for EBC nor the Value Pack (each operation only affects one mediation flow):

- Start a mediation flow (available in Java Console, uca-ebc-admin tool and UCA for EBC GUI)
- Stop a mediation flow (available in Java Console, uca-ebc-admin tool and UCA for EBC GUI)
- Check the status of a mediation flow (available in Java Console, and uca-ebc-admin tool)
- Resynchronize a mediation flow (available in Java Console, uca-ebc-admin tool and UCA for EBC GUI)
- Display the configuration of the mediation flow (as XML text) (available only in Java Console)

- Display the status/output of the last action (either CreateFlow, DeleteFlow, StatusFlow or ResynchronizeFlow) performed on the mediation flow (available only in Java Console)
- Display the status/output of the last CreateFlow action performed on the mediation flow (available only in Java Console)
- Display the status/output of the last DeleteFlow action performed on the mediation flow (available only in Java Console)
- Display the status/output of the last StatusFlow action performed on the mediation flow (available only in Java Console)
- Display the status/output of the last ResynchronizeFlow action performed on the mediation flow (available only in Java Console)

The following screenshot shows how to perform these operations on individual mediation flows using the Java console:

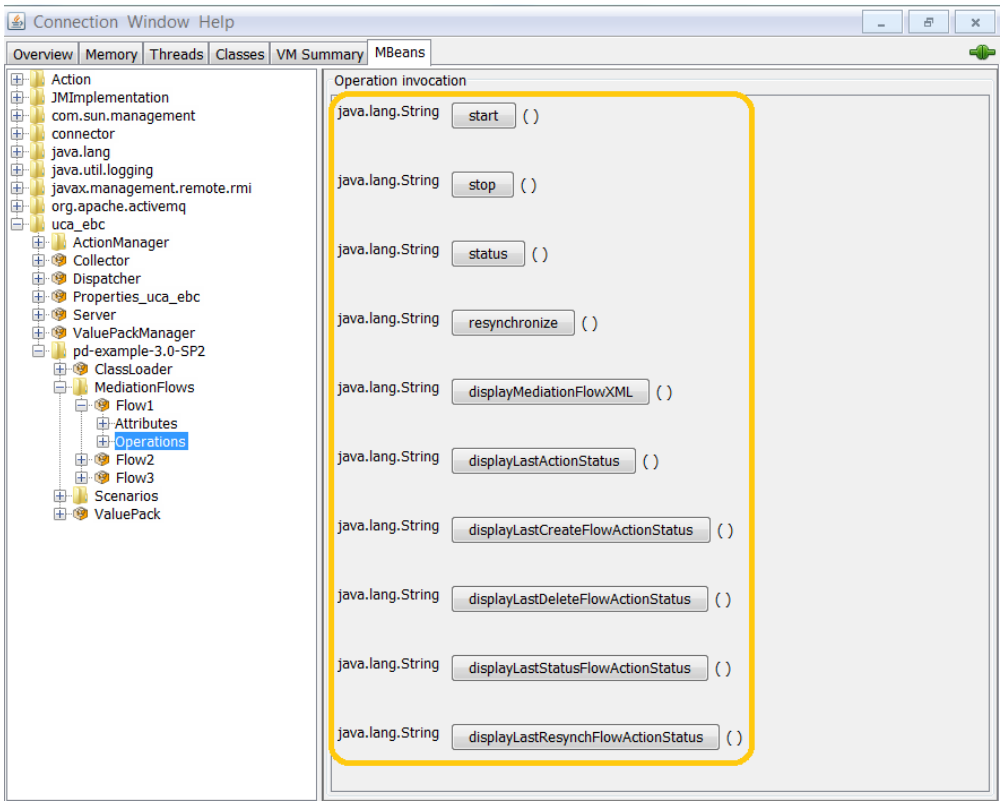


Figure 50: Java JMX Console: Performing operations on a single mediation flow

It is possible to start, stop, resynchronize, as well as view the status of individual mediation flows using the UCA for EBC User Interface as shown in the following screenshot:

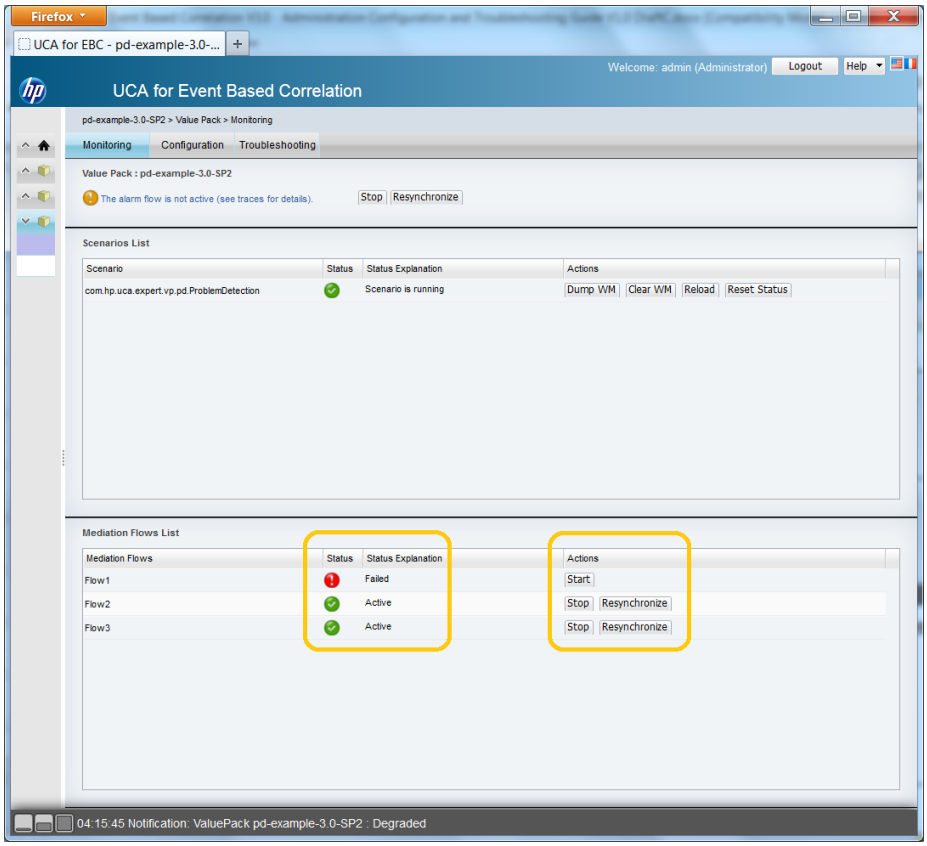


Figure 51: UCA for EBC User Interface: Performing operations on a single mediation flow

6.3.3 Managing actions

Actions are executed by the mediation layer. Each action is associated with the scenario that started the action.

6.3.3.1 Dumping Failed Actions

As actions are executed by the mediation layer, dumping the list of failed actions for a Scenario can be of great help while investigating issues regarding the mediation layer at the Scenario level.

The list of failed actions can be dumped in the log files (depending on your Log4J configuration). The log files can be viewed directly on the file system in the `${UCA_EBC_INSTANCE}/logs` directory using any text editor. The log files can also be viewed at the UCA for EBC User Interface in the Troubleshooting/Logs panel.

Dumping failed actions can only be performed using the Java JMX Console at the Scenario level by going to the "MBeans" tab of the Java Console and navigating to the "`uca_ebc/<value pack name>-<value pack version>/scenarios/<scenario name>/operations`" folder.

The following screenshot shows how to dump failed actions at the scenario level:

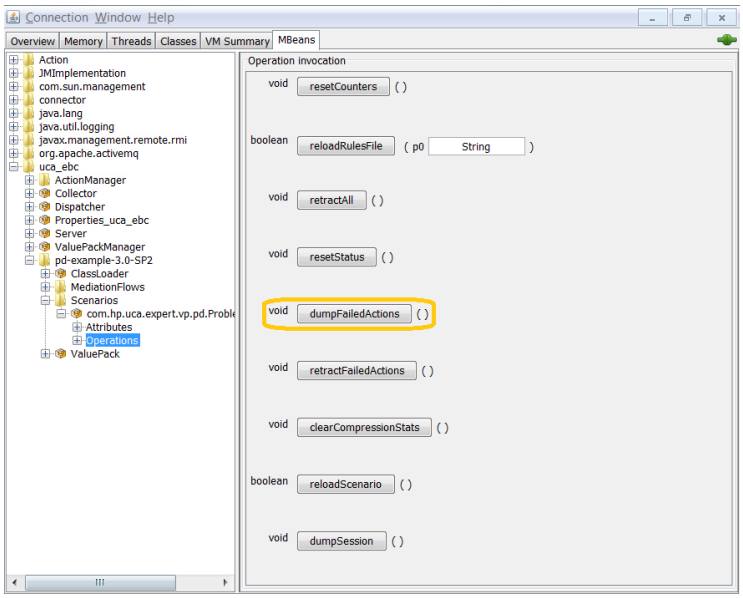


Figure 52: Java JMX Console: Dumping Failed Actions for a Scenario

NOTICE:
For more information on how to dump failed actions for a scenario, please see the section: 5.1.3.3 "Monitoring UCA for EBC scenarios"

6.4 UCA for EBC Performance analysis

Through the Java JMX interface, UCA for EBC provides event rate measurements that help when analyzing the performance of a UCA for EBC solution.

This "Dispatcher Rate" measure is the average event rate of UCA for EBC (in events per second) since start-up.

This measure is available by going to the "MBeans" tab of the Java Console and navigating to the "uca_ebc/Dispatcher/attributes" folder:

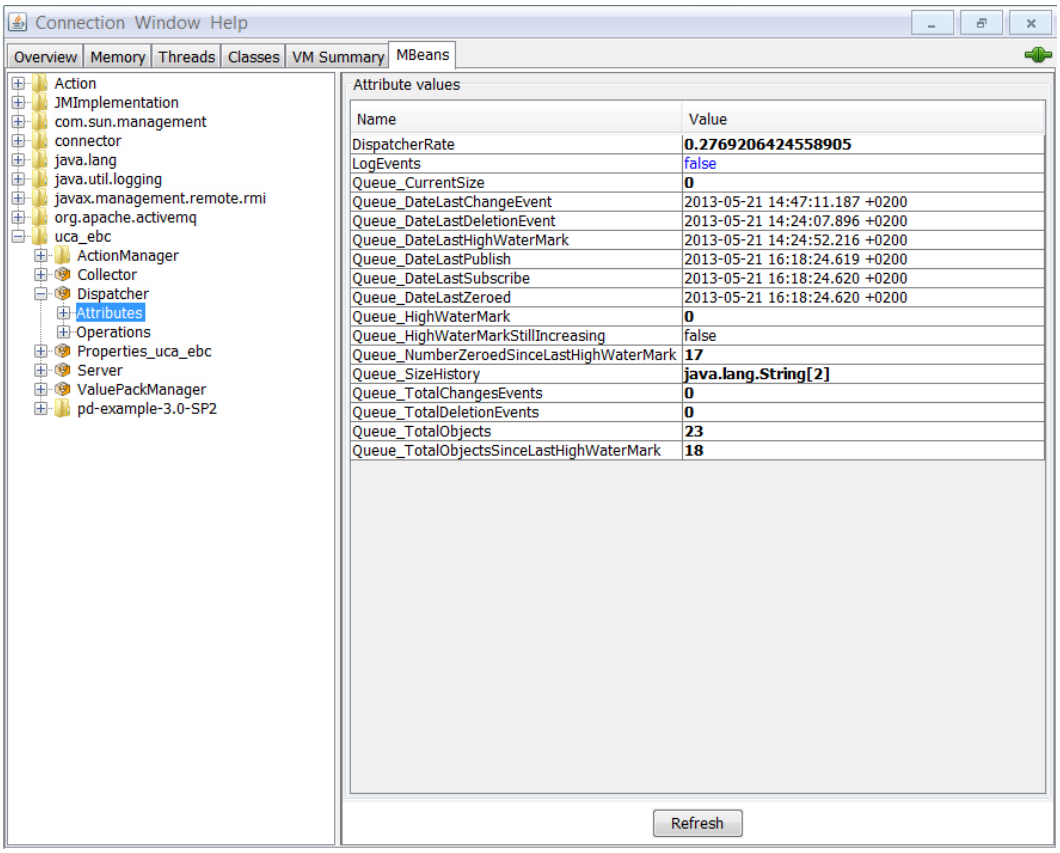


Figure 53: Java JMX Console: Monitoring performance of UCA for EBC Server

This measure and other measurement rates are available both at the Java JMX Console and also at the UCA for EBC User Interface in the Troubleshooting / Statistics panel.

NOTICE:

For more information on the Java JMX Console, please see the section 5.1.3 "JMX Console".

For more information on the UCA for EBC User Interface, please refer to: [R3] `HPE UCA for EBC User Interface Guide`.

Please see section 4.1 "Monitoring the alarm flow in real-time" for more information on how to monitor the alarm flow of UCA for EBC

Chapter 7
Frequent problems and solutions

Below is a list of known issues/ problems that you may encounter, along with a description of how to solve or work around the issue/problem.

7.1 Problems executing uca-ebc-admin

7.1.1 Cannot connect to UCA for EBC JMX connector

If you get an error stating "Cannot connect to UCA Expert JMX connector" while executing the uca-ebc-admin command-line tool, then you may want to perform the following verifications:

Table 41: uca-ebc-admin: Cannot connect to UCA for EBC JMX connector

Verification	Suggested solution/work-around
Verify that UCA for EBC Server is started	Start UCA for EBC Server if it is stopped

Below is an example of a command execution displaying this error:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc-admin <options>

ERROR - Cannot connect to UCA Expert JMX connector. Failed to retrieve RMISer
ver stub: javax.naming.ServiceUnavailableException [Root exception is java.rm
i.ConnectException: Connection refused to host: localhost; nested exception i
s:
        java.net.ConnectException: Connection refused (errno:239)]
```

7.1.2 FileNotFoundException: ${UCA_EBC_INSTANCE} /logs/uca-ebc-admin.log

If you get an error stating "FileNotFoundException: ${UCA_EBC_INSTANCE} /logs/uca-ebc-admin.log" while executing the uca-ebc-admin command-line tool, then you may want to perform the following verifications:

Table 42: uca-ebc-admin: FileNotFoundException

Verification	Suggested solution/work-around
Verify that the user trying to execute uca-ebc-admin has permission to write in the ${UCA_EBC_INSTANCE} directory	Use another user account or change the permissions on the ${UCA_EBC_INSTANCE} directory if this is not the case

Below is an example of a command execution displaying this error:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
```

```
$ uca-ebc-admin <options>

…
log4j:ERROR setFile(null,true) call failed.
java.io.FileNotFoundException:  /var/opt/UCA-EBC/logs/uca-ebc-
admin.log (Permission denied (errno:13))
        at java.io.FileOutputStream.openAppend(Native Method)….
```

7.2 Problems executing uca-ebc-injector

7.2.1 Cannot create connection

If you get an error stating "Cannot create connection on UCA Expert JMS queue" while executing the uca-ebc-injector command-line tool, then you may want to perform the following verifications:

Table 43: uca-ebc-injector: Cannot create connection

Verification	Suggested solution/work-around
Verify that UCA for EBC Server is started	Start UCA for EBC Server if it is stopped

Below is an example of a command execution displaying this error:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc-injector <options>

 /opt/UCA-EBC/bin>uca-ebc-injector -file ../alarms/Alarms.xml

ERROR - Command error: Cannot create connection on UCA Expert JMS queue
```

7.2.2 FileNotFoundException: ${UCA_EBC_INSTANCE} /logs/uca-ebc-injector.log

If you get an error stating "FileNotFoundException: ${UCA_EBC_INSTANCE} /logs/uca-ebc-injector.log" while executing the uca-ebc-injector command-line tool, then you may want to perform the following verifications:

Table 44: uca-ebc-injector: FileNotFoundException

Verification	Suggested solution/work-around
Verify that the user trying to execute uca-ebc-injector has permission to write in the ${UCA_EBC_INSTANCE} directory	Use another user account or change the permissions on the ${UCA_EBC_INSTANCE} directory if this is not the case

Below is an example of a command execution displaying this error:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc-injector <options>
…
log4j:ERROR setFile(null,true) call failed.
```

```
java.io.FileNotFoundException: /var/opt/UCA-EBC/logs/uca-ebc-
injector.log (Permission denied (errno:13))
        at java.io.FileOutputStream.openAppend(Native Method)….
```

7.3 Problems starting UCA for EBC

7.3.1 AlreadyBoundException

If you get an error stating "java.rmi.AlreadyBoundException: uca-ebc" while starting UCA for EBC, then you may want to perform the following verifications:

Table 45: uca-ebc: AlreadyBoundException

Verification	Suggested solution/work-around
Verify that there's no port number conflict between UCA for EBC RMI port number and the port numbers used by another process on the system	Update the UCA for EBC RMI port number in the *${UCA_EBC_INSTANCE}/conf/uca-ebc.properties* file to avoid the port number conflict if needed

Below is an example of a command execution displaying this error:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc <options>

…
INFO  - Unregistering JMX-exposed beans on shutdown
INFO  - Closing Hibernate SessionFactory
INFO  - closing
org.springframework.beans.factory.BeanCreationException: Error creating bean
with name 'serverConnector' defined in class path resource [main-context.xml]
: Invocation of init method failed; nested exception is java.io.IOException:
Cannot bind to URL [rmi://localhost:1100/uca-
ebc]: javax.naming.NameAlreadyBoundException: uca-
ebc [Root exception is java.rmi.AlreadyBoundException: uca-ebc]
```

7.3.2 ClassNotFoundException: javax.management.remote.rmi.RMIServerImpl_Stub

If you get an error stating "java.lang.ClassNotFoundException: javax.management.remote.rmi.RMIServerImpl_Stub" while starting UCA for EBC, then you may want to perform the following verifications:

Table 46: uca-ebc: ClassNotFoundException

Verification	Suggested solution/work-around
Verify that there's no port number conflict between UCA for EBC RMI port number and the port numbers used by another process on the system	Update the UCA for EBC RMI port number in the ${UCA_EBC_INSTANCE}/conf/uca-ebc.properties file to avoid the port number conflict if needed

Below is an example of a command execution displaying this error:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
```

```
$ uca-ebc <options>

…
                ... 30 more
Caused by: java.rmi.UnmarshalException: error unmarshalling arguments; nested
 exception is:
        java.lang.ClassNotFoundException: javax.management.remote.rmi.RMIServ
erImpl_Stub (no security manager: RMI class loader disabled)
Caused by: java.lang.ClassNotFoundException: javax.management.remote.rmi.RMIS
erverImpl_Stub (no security manager: RMI class loader disabled)
```

7.3.3 FileNotFoundException: ${UCA_EBC_INSTANCE} /logs/uca-ebc.log

If you get an error stating "`FileNotFoundException: ${UCA_EBC_INSTANCE} /logs/uca-ebc.log`" while starting UCA for EBC, then you may want to perform the following verifications:

Table 47: uca-ebc: FileNotFoundException

Verification	Suggested solution/work-around
Verify that the user trying to start UCA for EBC has permission to write in the ${UCA_EBC_INSTANCE} directory	Start UCA for EBC under the uca account if this is not the case

Below is an example of a command execution displaying this error:

On both HP-UX, and Linux:

```
$ cd ${UCA_EBC_HOME}/bin
$ uca-ebc <options>

…
log4j:ERROR setFile(null,true) call failed.
java.io.FileNotFoundException:  /var/opt/UCA-EBC/logs/uca-
ebc.log (Permission denied (errno:13))
        at java.io.FileOutputStream.openAppend(Native Method)
        at java.io.FileOutputStream.<init>(FileOutputStream.java:177)
….
```

Appendix A
Glossary

Table 48: Acronym table

Acronym	Description
CA	Channel Adapter for OSS Open Mediation V7.2
DB	Database
DRL	Drools Rule file
EBC	Event Based Correlation
EVP	UCA for EBC Value Pack
GUI	Graphical User Interface
Inference engine	Process that uses a Rete algorithm
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
NMS	Network Management System
SDK	Software Development Kit
TT	Trouble Ticket
UCA	Unified Correlation Analyzer
XML	Extensible Markup Language
XSD	Schema of an XML file, describing its structure. XSD stands for XML Schema Definition
X733	Standard describing the structure of an Alarm used in the telecommunications environment.