hp Unified Correlation Analyzer

Unified Correlation Analyzer for Event Based Correlation

Version 3.0

Value Pack Development Guide

Edition: 1.0

For Windows© and Linux (RHEL 5.8 & 6.3) Operating Systems

June 2013

Contents

Figures

Tables

Preface

This guide provides an overview of the Unified Correlated Analyzer for Event Based Correlation product and describes how to create Value Packs to target customer specific use cases.

Product Name: Unified Correlation Analyzer for Event Based Correlation

Product Version: V3.0

Intended Audience

Here are some recommendations based on possible reader profiles:

- Solution Developers
- Software Development Engineers

Software Versions

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

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

Product Version	Supported Operating systems
UCA for Event Based Correlation Software Development Kit V3.0	• Windows XP / Vista • Windows Server 2007 • Windows 7 • Red Hat Enterprise Linux Server release 5.8 & 6.3

Table 1 - Software versions

Typographical Conventions

`Courier` Font:

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

Italic Text:

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

Bold Text:

- To introduce new terms and to emphasize important words.

Associated Documents

The following documents contain useful reference information:

References

[R1] *HP UCA for Event Based Correlation – Installation Guide*

[R2] *HP UCA for Event Based Correlation – Reference Guide*

[R3] *HP UCA for Event Based Correlation – Administration, Configuration and Troubleshooting Guide*

[R4] *HP UCA for Event Based Correlation – Value Pack Examples*

[R5] *OSS Open Mediation V601 Functional Specification*

[R6] *OSS Open Mediation V601 Installation and Configuration Guide*

Support

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

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

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

Chapter 1

Introduction

This guide explains how to create a new correlation project, how to package it and deploy it on a Unified Correlated Analyzer for Event Based Correlation (UCA for EBC) Server in just a few minutes.

The following chapters will dive into the development of UCA for EBC Value Packs and explain how to create new scenarios, how to develop alarm/event correlation rules based on samples and how to customize UCA for EBC.

Note

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

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

Since UCA-EBC V2.0, on Linux and HP-UX systems, the `${UCA_EBC_DATA}` directory may contain multiple instances of UCA-EBC. In this document, we will use the value `${UCA_EBC_INSTANCE}` for referring to `${UCA_EBC_DATA}/instances/<instance-name>` directory on Linux/HP-UX systems and to `${UCA_EBC_DATA}` on Windows systems.
Note that at installation time on Linux/HP-UX, a single <instance-name> is configured: default.

Chapter 2

Getting started with UCA for EBC

This chapter prepares you to quickly build UCA for EBC Value Packs.

After validating some pre-requisites and installing both UCA for EBC (runtime) and UCA for EBC Development Kit products, you will learn how easy it is to create your own UCA for EBC Value Pack, create correlation rules that match your use case(s) and package your Value Pack.

Then you will see how to deploy your Value Pack on a UCA for EBC Server and test it live.

2.1 Software Pre-requisites

2.1.1 Operating system

The UCA for EBC Development Kit is provided (and supported) for:

- Windows operating systems.
  It has been validated on Windows XP, Windows Vista, Windows 7, and Windows Server 2007.
- Red Hat Enterprise Linux.
  It has been validated on Server Release 5.8 & 6.3.

2.1.2 Java JRE/JDK

The following table lists the Java JRE/JDK pre-requisites for UCA for EBC Development Kit:

Software	**Version**
Java JDK*	1.6.0.08 (or later)

Table 2 - Java JRE/JDK Prerequisites for UCA for EBC Development Kit

You can check whether Java is already installed on your system and which version of the Java JRE/JDK is installed by issuing the following commands:

On Windows XP, Windows Vista, Windows 7, and Windows Server 2007:

To check if you already have Java installed, open a command-line (Run… -> cmd.exe) and type:

```
C:\> java -version
```

You should get an output similar to the following:

```
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-
b04)
Java HotSpot(TM) Client VM (build 14.3-b01, mixed
mode, sharing)
```

Alternatively to using the command-line, you can check if you already have Java installed by going to the Control Panel and selecting the Java icon. In the Java tab, you will find information on the Java version installed on your system.

The latest JDK package for Windows XP, Windows Vista, Windows 7, and Windows Server 2007 can be downloaded (for free) from www.hp.com/go/java

On Linux:

To check if you already have Java installed:

```
$ rpm -qa | grep jdk
```

Red Hat Enterprise Linux Server comes with OpenJDK Java VM. You should get an output similar to the following (here 1.6.0 and 1.7.0 are installed):

```
java-1.6.0-openjdk-1.6.0.0-1.41.1.10.4.el6.x86_64
java-1.6.0-openjdk-devel-1.6.0.0-1.41.1.10.4.el6.x86_64
java-1.7.0-openjdk-1.7.0.9-2.3.4.1.el6_3.x86_64
java-1.7.0-openjdk-devel-1.7.0.9-2.3.4.1.el6_3.x86_64
```

You can also download (for free) the latest Java packages (HotSpot Java VM) from Oracle from http://java.com/en/download/manual.jsp. If this is installed (usually under /usr/java), you should get an output similar to the following:

```
jdk-1.6.0_23-fcs.x86_64
```

Note

* Java 1.6 JRE is enough for using the UCA for EBC Development Kit. However the JDK comes with some useful debugging tools (jconsole, jvisualvm, etc...) that may prove helpful for troubleshooting. It is therefore recommended to install the JDK.

2.1.3 Eclipse IDE

The UCA for EBC Development Kit has been designed for an easy integration with the Eclipse Integrated Development Environment (IDE) tool.

Before starting the development of any UCA for EBC value pack, it is necessary to download and install the Eclipse ™ application development environment.

The following table lists the Eclipse IDE pre-requisites for UCA for EBC Development Kit:

Software	Version
Eclipse IDE	3.7 (Indigo) or higher

Table 3 - Eclipse IDE Prerequisites for UCA for EBC Development Kit

The minimum version of Eclipse IDE required by the UCA for EBC Development Kit is version 3.4 but we recommended Eclipse IDE version 3.7 (Indigo) or higher.

If you already have Eclipse IDE installed on your system, you can either use this version with the UCA for EBC Development Kit (provided this version complies with the version requirement: version 3.4 or higher) or you can install a new version of Eclipse IDE.

If you want to install Eclipse IDE, please go to the following URL for downloading Eclipse IDE: http://www.eclipse.org/downloads/

At the time of writing, the Eclipse IDE version is *Juno 4.2*.

We recommend you to download either (other choices may also be valid):

- Eclipse IDE for Java Developers, or
- Eclipse IDE for Java EE Developers

Then you need to choose to install either the 32-bit or 64-bit version of Eclipse IDE depending on whether you have a 32-bit or 64-bit operating system.

Once Eclipse IDE is installed on your system, and in order to get the full benefit of the Drools development environment in Eclipse, it is also necessary to download and install the Drools plug-in for Eclipse.

Before downloading the Drools plug-in for Eclipse IDE, please make sure that the Drools plug-in you plan to download has the same version number as the version of Drools used by UCA for EBC.

UCA for EBC currently uses Drools version 5.5.0.Final. The download URL for this version of the plug-in is the following:

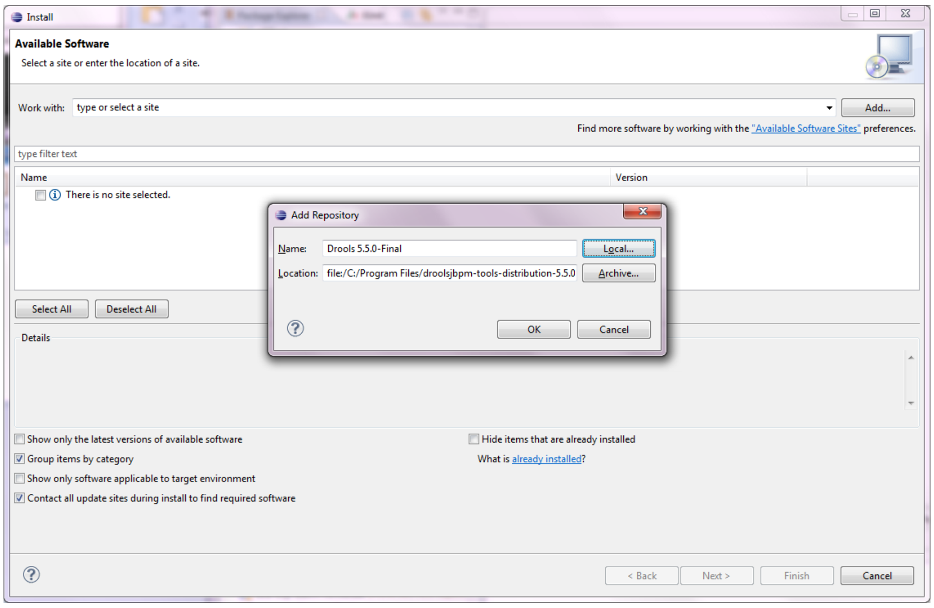
https://repository.jboss.org/nexus/content/repositories/releases/org/drools/org.drools.updatesite/5.5.0.Final/org.drools.updatesite-5.5.0.Final-assembly.zip

2.1.3.1 Drools plug-in for Eclipse IDE installation instructions

Download and save the ZIP file of the Drools plug-in for Eclipse IDE in a temporary directory, for example: *C:\Temp*.

From the Eclipse 'Help' menu, choose 'Install new software' and then click on the Add... button.

Select the downloaded file using the Archive... button and give it the name "jboss drools tools 5.5.0.Final" as shown in the picture below:


Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☐ There is no site selected.	

Add Repository

Name: Drools 5.5.0-Final

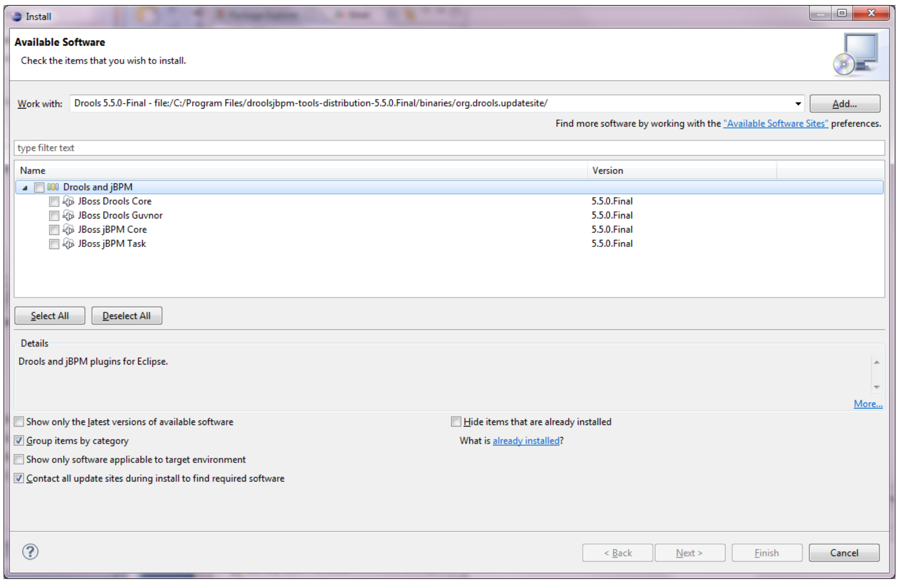
Location: file:/C:/Program Files/droolsjbpm-tools-distribution-5.5.0

☐ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☐ Hide items that are already installed
What is already installed?

Figure 1 - Drools plug-in for Eclipse IDE: Installation step 1

Then click on the OK button.

The screen should then display the archive content as follow:


Install

Available Software

Check the items that you wish to install.

Work with: Drools 5.5.0-Final - file:/C:/Program Files/droolsjbpm-tools-distribution-5.5.0.Final/binaries/org.drools.updatesite/

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☐ Drools and jBPM	
☐ JBoss Drools Core	5.5.0.Final
☐ JBoss Drools Guvnor	5.5.0.Final
☐ JBoss jBPM Core	5.5.0.Final
☐ JBoss jBPM Task	5.5.0.Final

Details

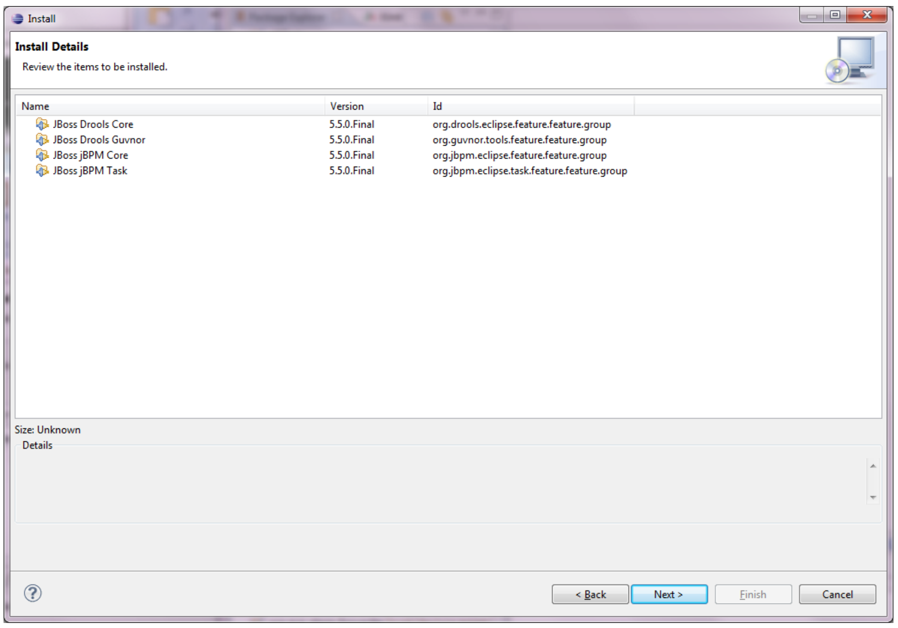
Drools and jBPM plugins for Eclipse.

☐ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☐ Hide items that are already installed
What is already installed?

Figure 2 - Drools plug-in for Eclipse IDE: Installation step 2

Check the “Drools and jBPM” checkbox and then click on the Next > button.

The following screen is displayed:



Install Details

Review the items to be installed.

Name	Version	Id
JBoss Drools Core	5.5.0.Final	org.drools.eclipse.feature.feature.group
JBoss Drools Guvnor	5.5.0.Final	org.guvnor.tools.feature.feature.group
JBoss jBPM Core	5.5.0.Final	org.jbpm.eclipse.feature.feature.group
JBoss jBPM Task	5.5.0.Final	org.jbpm.eclipse.task.feature.feature.group

Size: Unknown

Details

Figure 3 - Drools plug-in for Eclipse IDE: Installation step 3

Click on the Next > button for installing the plug-in after accepting the license terms.

The plug-in installation requires a restart of your Eclipse IDE environment.

2.1.4 Installing UCA for EBC and UCA for EBC Development Kit

Detailed information on how to install UCA for EBC and UCA for EBC Development Kit is provided in the [R1] *HP UCA for Event Based Correlation – Installation Guide*

2.1.5 Post-install Environment Setup

2.1.5.1 The UCA_EBC_DEV_HOME Variable

The UCA for EBC Development Kit installation procedure adds the %UCA_EBC_DEV_HOME% environment variable to your user environment.

This variable is necessary for various development phases of a UCA for EBC value pack development, especially the build and packaging phases.

To verify that this variable is correctly set after the UCA for EBC Development Kit has been installed, open a command-line (Run... -> cmd.exe) and type:

On Windows:

```
C:\> echo %UCA_EBC_DEV_HOME%
```

You should get an output similar to the following:

```
C:\UCA-EBC-DEV\
```

Note

On Windows 7, you should log out and log back in again for the new environment variable to be taken into account after installation of the UCA for EBC Development Kit.

On Linux:

```
$ echo ${UCA_EBC_DEV_HOME}
```

You should get an output similar to the following:

```
/opt/UCA-EBC-DEV
```

Note

On Linux this Variable must be manually set in the user's environment, as specified in the UCA for EBC Installation Guide.

2.1.5.2 Ant Configuration

The UCA for EBC value pack packaging is based on the use of the Apache Ant tool. This tool requires a specific version and specific settings. Be sure to use the Apache Ant tool provided with UCA for EBC in the *%UCA_EBC_DEV_HOME%\3pp\ant* directory (${UCA_EBC_DEV_HOME}/3pp/ant on Linux).

Be sure that you don't have the ANT_HOME environment variable set to the path of another version of Apache Ant, which would create conflicts with the version of Apache Ant in the *3pp\ant\bin* folder. If you do, you should either clear the ANT_HOME environment variable:

```
C:\> set ANT_HOME=
```

Or set it to the directory of the Apache Ant version that comes with the UCA for EBC development kit:

```
C:\> set ANT_HOME=%UCA_EBC_DEV_HOME%\3pp\ant

$ANT_HOME/bin/ant -version
Apache Ant(TM) version 1.8.2 compiled on December 20 2010
```

The delivered Apache Ant version that comes with the UCA for EBC development kit is:

```
# $ANT_HOME/bin/ant -version
Apache Ant(TM) version 1.8.2 compiled on December 20 2010
```

2.1.6 UCA for EBC Eclipse plug-in installation instructions

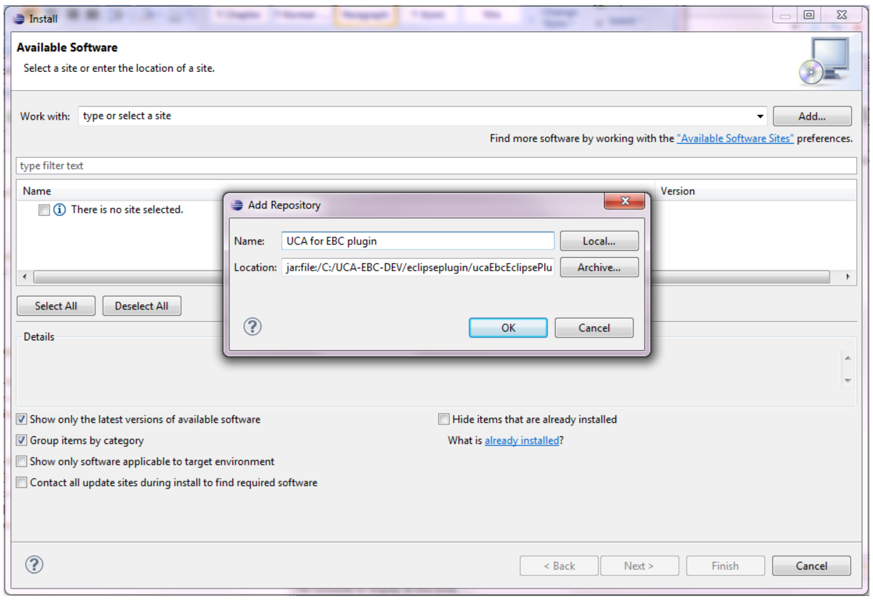
The UCA for EBC Development Kit delivers an Eclipse plug-in that eases UCA for EBC value pack project creation under eclipse.

This plugin is delivered in the %UCA_EBC_DEV_HOME%\eclipseplugin\ucaEbcEclipsePluginSite-3.0.2-assembly.zip file.

The installation of this plug-in is made as follows:

From the Eclipse ‘Help’ menu, choose ‘Install new software’ and then click on the Add... button.

Select the UCA for EBC eclipse plug-in ZIP file using the Archive... button and give it the name “UCA for EBC plug-in” as shown in the picture below:


Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Available Software Sites" preferences.

type filter text

There is no site selected.

Add Repository

Name: UCA for EBC plugin

Location: jar:file:/C:/UCA-EBC-DEV/eclipseplugin/ucaEbcEclipsePlu

[✓] Show only the latest versions of available software

[✓] Group items by category

[ ] Show only software applicable to target environment

[ ] Contact all update sites during install to find required software

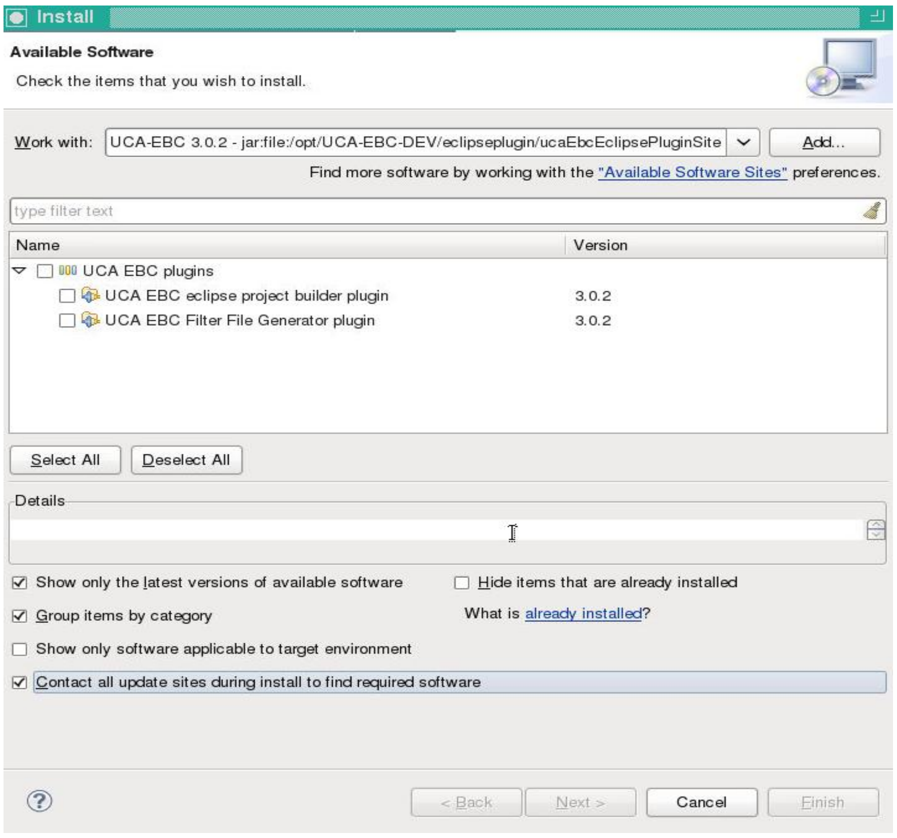
[ ] Hide items that are already installed

What is already installed?

Figure 4 - UCA for EBC Eclipse plug-in: Installation step 1

Then click on the OK button.

The screen should then display the archive content as follow:


Install

Available Software

Check the items that you wish to install.

Work with: UCA-EBC 3.0.2 - jar:file:/opt/UCA-EBC-DEV/eclipseplugin/ucaEbcEclipsePluginSite

Find more software by working with the "Available Software Sites" preferences.

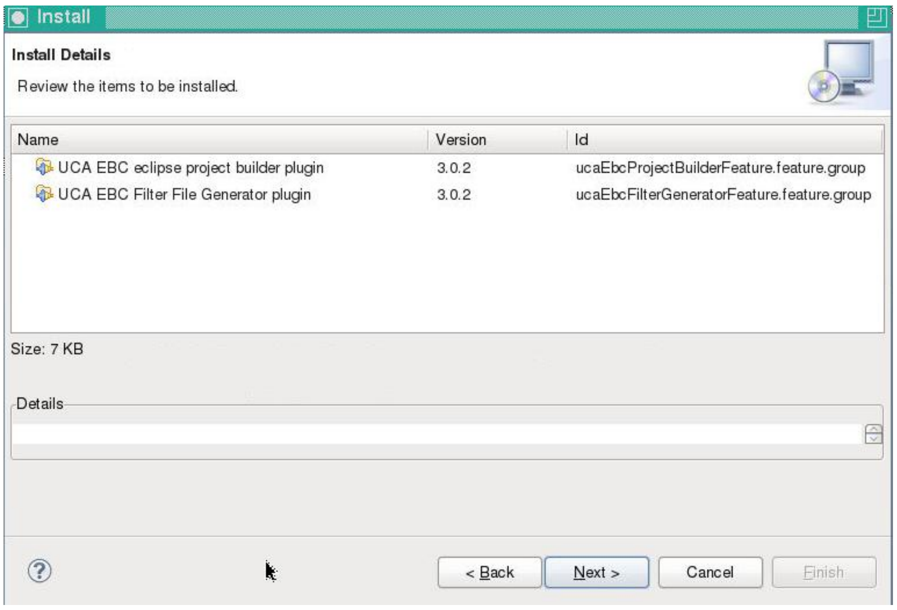
Name	Version
☐ UCA EBC plugins	
☐ UCA EBC eclipse project builder plugin	3.0.2
☐ UCA EBC Filter File Generator plugin	3.0.2

- ☑ Show only the latest versions of available software
- ☐ Hide items that are already installed
- ☑ Group items by category
- What is already installed?
- ☐ Show only software applicable to target environment
- ☑ Contact all update sites during install to find required software

Figure 5 - UCA for EBC Eclipse plug-in: Installation step 2

Check the "UCA EBC plugins" checkbox, uncheck the "Contact all update sites...", and then click on the Next > button.

The following screen is displayed:



Name	Version	Id
UCA EBC eclipse project builder plugin	3.0.2	ucaEbcProjectBuilderFeature.feature.group
UCA EBC Filter File Generator plugin	3.0.2	ucaEbcFilterGeneratorFeature.feature.group

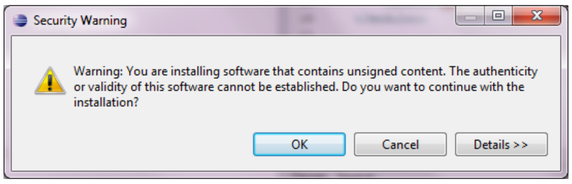
Size: 7 KB

Figure 6 - UCA for EBC Eclipse plug-in: Installation step 3

Click on the Next > button for installing the plug-ins after accepting the license terms.

Note

The following message may appear during the installation. This is normal as the provided jar files are not signed in the current version.



Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the installation?

Click OK to continue the installation.

The plug-in installation requires a restart of your Eclipse IDE environment. Please restart eclipse before any attempt to create a UCA for EBC project.

2.2 Creating a new UCA for EBC Value Pack

UCA for EBC can be seen as an application container in which so called UCA for EBC “Value Packs" are deployed.

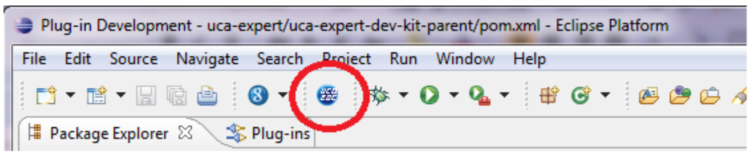
A Value Pack represents a set of features (scenarios) that are grouped together to implement one or more correlation use cases.

A UCA for EBC value pack thus includes for example: event filtering, event based rules, customized java code and possibly configuration files for each of these scenarios.

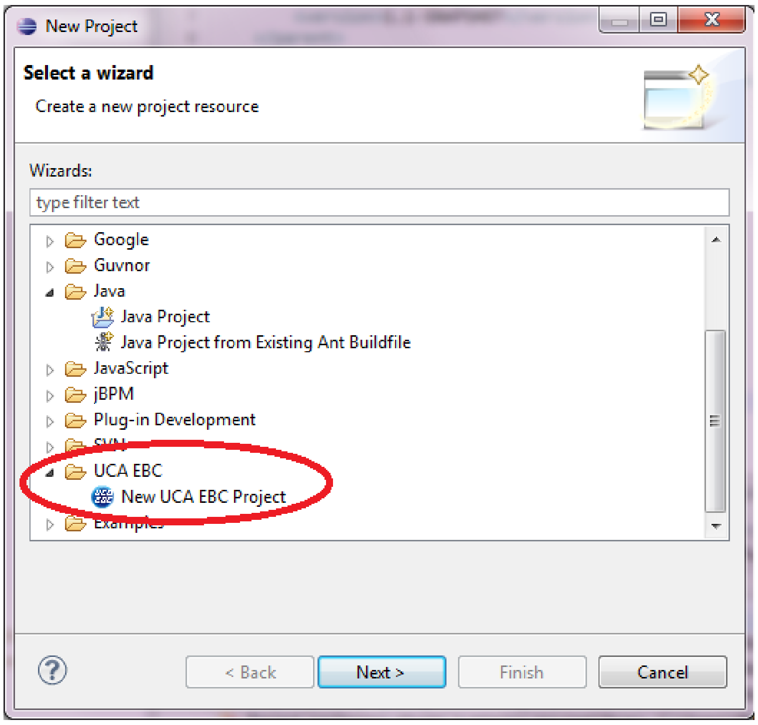
2.2.1 Creating a value pack project within Eclipse

The UCA for EBC eclipse plug-in provides a project creation wizard allowing the creation of a new value pack project in just a few clicks and dialog boxes.

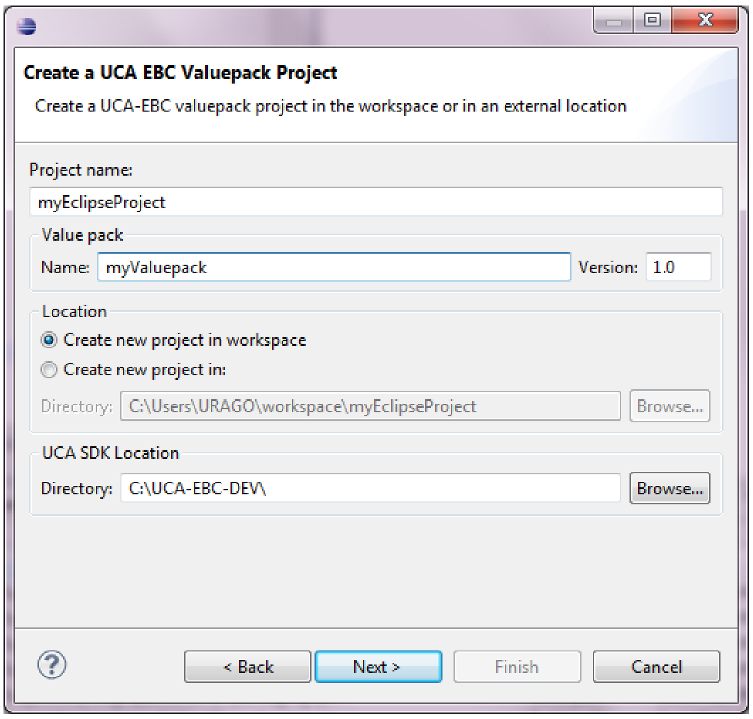
This wizard can be launched from the eclipse main toolbar by clicking on the UCA/EBC icon:


Plug-in Development - uca-expert/uca-expert-dev-kit-parent/pom.xml - Eclipse Platform
File
Edit
Source
Navigate
Search
Project
Run
Window
Help
Package Explorer
Plug-ins

Or from the Eclipse "New Project" Menu as follow:



This launches the UCA EBC value pack wizard:


Create a UCA EBC Valuepack Project

Create a UCA-EBC valuepack project in the workspace or in an external location

Project name: myEclipseProject

Value pack

Name: myValuepack Version: 1.0

Location

(●) Create new project in workspace

( ) Create new project in:

Directory: C:\Users\URAGO\workspace\myEclipseProject

UCA SDK Location

Directory: C:\UCA-EBC-DEV\

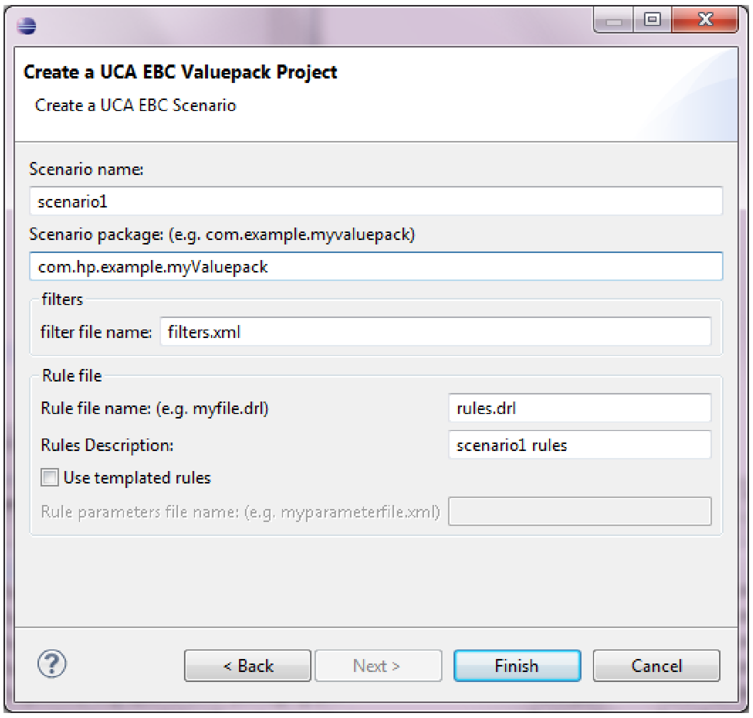
Figure 7 - Value pack project creation wizard Step1

From this panel you can set the project and value pack configuration:

- On the first line you must enter the name of the eclipse project to be created.
- On the second line you need to give the value pack name and its version
- Then the 'location' panel allows specifying the location of the created project. It can be in the current workspace or in an external directory of your choice.
- Finally the UCA SDK Location allows specifying the home directory of the UCA for EBC Development kit. The default value is obtained from the %UCA_EBC_DEV_HOME% environment variable.

Then Click on the Next > button for getting the next wizard step.

This is the scenario panel configuration. Note that the project creation wizard allows creating a single initial scenario per value pack. The creation of additional scenarios for a given value pack must be done manually by editing the various value pack configuration files.



Create a UCA EBC Valuepack Project

Create a UCA EBC Scenario

Scenario name: scenario1

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

filters

filter file name: filters.xml

Rule file

Rule file name: (e.g. myfile.drl) rules.drl

Rules Description: scenario1 rules

☐ Use templated rules

Rule parameters file name: (e.g. myparameterfile.xml)

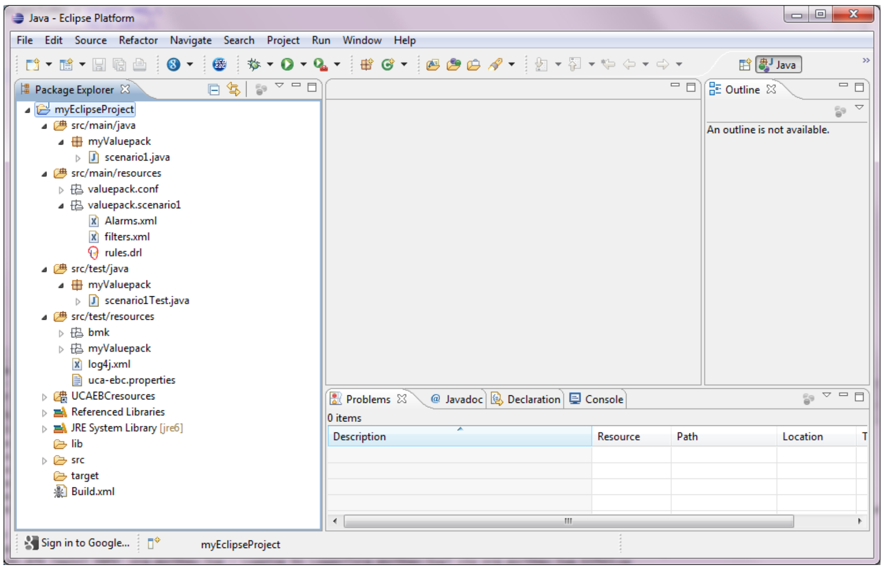
Figure 8 - Value pack project creation wizard Step2

At this step you can set the scenario parameters:

- On the first line you must enter the scenario name.
- On the second line you need to give the scenario package name. This package name will be used for all the scenario's java source code files.
- In the filter panel you have to enter the name of the filter file for this scenario. As this is an XML file, the '.xml' suffix is mandatory.
- Then the rule panel allows you specifying the rule file name (and a description) and also specify if this scenario will use template rules file or not (this is done by checking the 'Use template rule' box.
- Then Click on the Finish button for creating the Project.

Note: for creating "topology based" Value Pack project, please refer to the UCA Topology Extension user guide.

This project creation wizard execution leads to the creation of an Eclipse project skeleton. It exhibits a basic correlation scenario that can compile and unit test successfully. From this example, developers can extend it to build their own Value Packs.



```
myEclipseProject
  src/main/java
    myValuepack
      scenario1.java
  src/main/resources
    valuepack.conf
    valuepack.scenario1
      Alarms.xml
      filters.xml
      rules.drl
  src/test/java
    myValuepack
      scenario1Test.java
  src/test/resources
    bmk
    myValuepack
    log4j.xml
    uca-ebc.properties
  UCAEBCresources
  Referenced Libraries
  JRE System Library [jre6]
  lib
  src
  target
  Build.xml
```

An outline is not available.

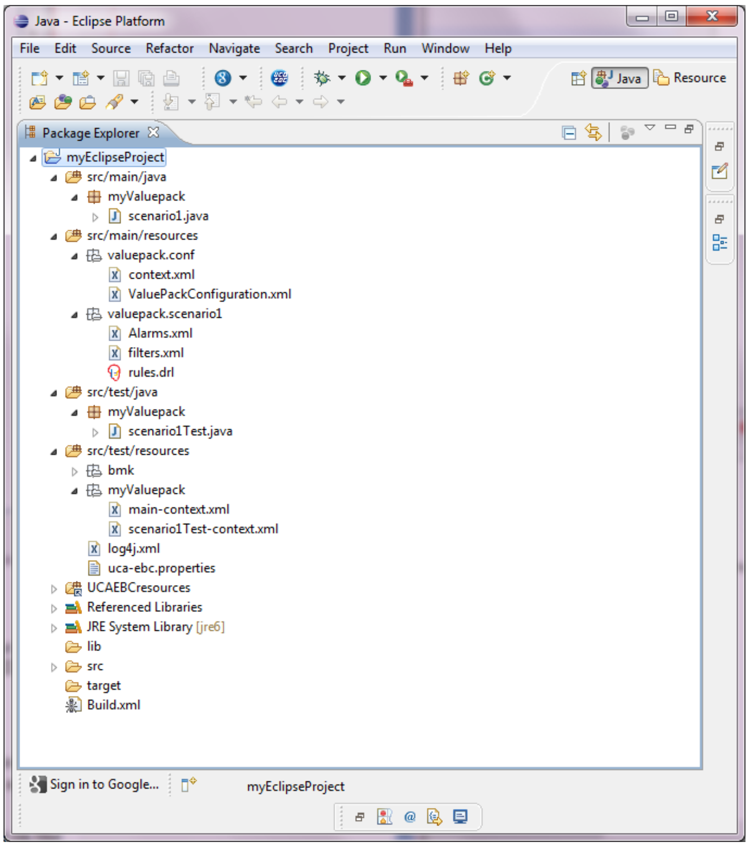
0 items

Figure 9 - Created Value pack

2.3 Anatomy of the created project

Using Eclipse IDE, you can browse through the different directories that compose the created “Skeleton” project.

Please see below for a glimpse at the folder structure of the created project:



```
myEclipseProject
  src/main/java
    myValuepack
      scenario1.java
  src/main/resources
    valuepack.conf
      context.xml
      ValuePackConfiguration.xml
    valuepack.scenario1
      Alarms.xml
      filters.xml
      rules.drl
  src/test/java
    myValuepack
      scenario1Test.java
  src/test/resources
    bmk
    myValuepack
      main-context.xml
      scenario1Test-context.xml
    log4j.xml
    uca-ebc.properties
  UCAEBCresources
  Referenced Libraries
  JRE System Library [jre6]
  lib
  src
  target
  Build.xml
```

Figure 10 - Folder structure of the created project

The created “Skeleton” project also comes with an Apache Ant build.xml file that is used for building and packaging the value pack outside of the Eclipse IDE.

2.4 Validation of the created project

The created project contains predefined test classes that automatically load/compiles the value pack resources (scenario definitions, filters and rules files) and validate them (at least syntactically).

JUnit tests can be run either directly from eclipse, by right-clicking on the test package and choosing “Run As > JUnit Test” as shown in the following screen shot:

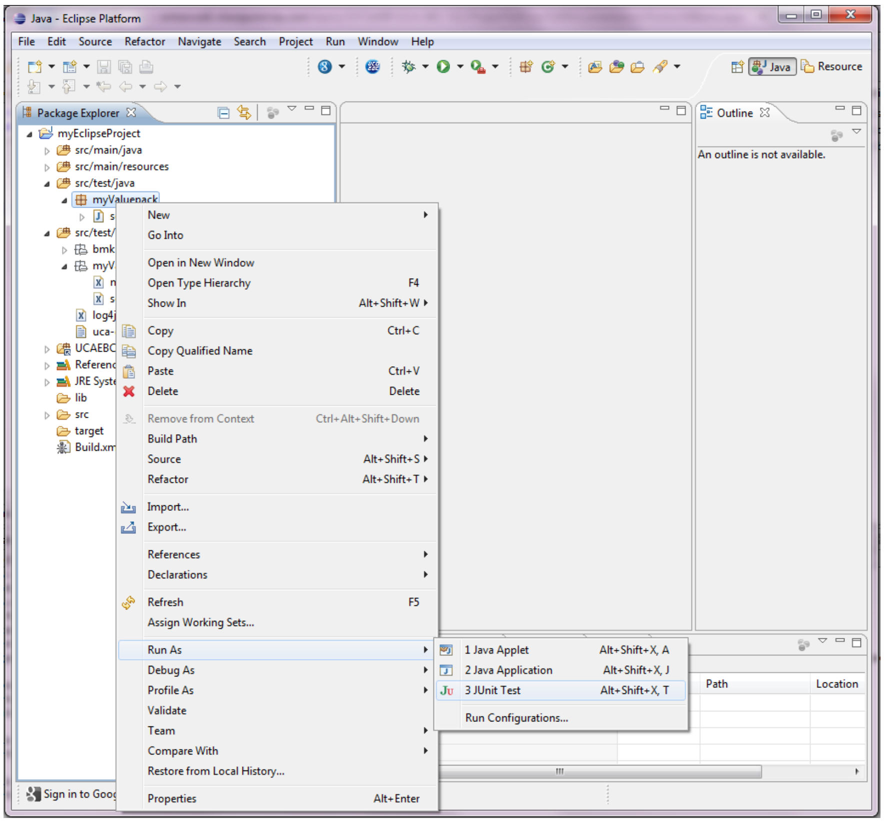
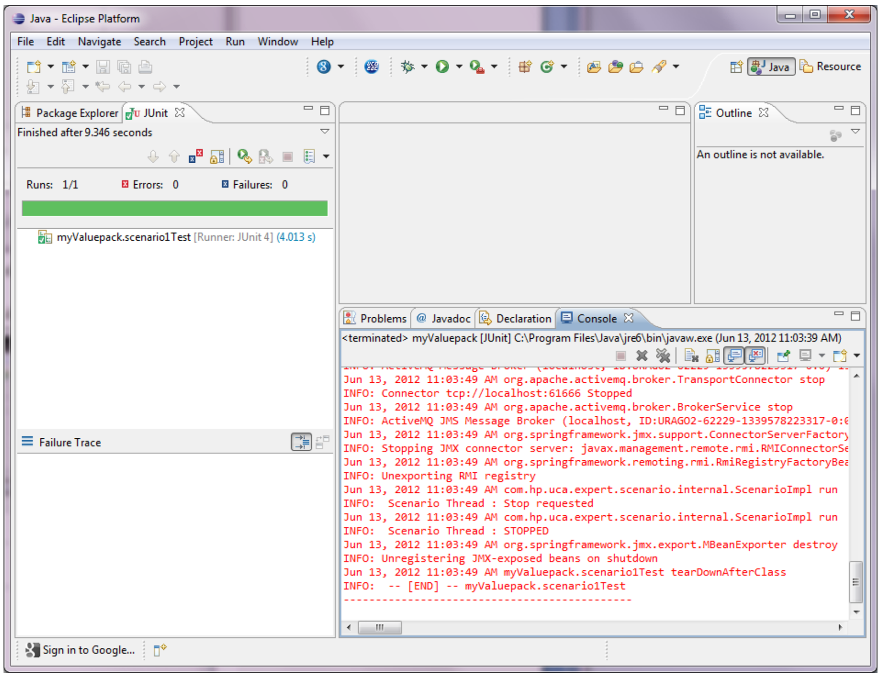


Figure 11- Running JUnit tests on the created project in Eclipse IDE

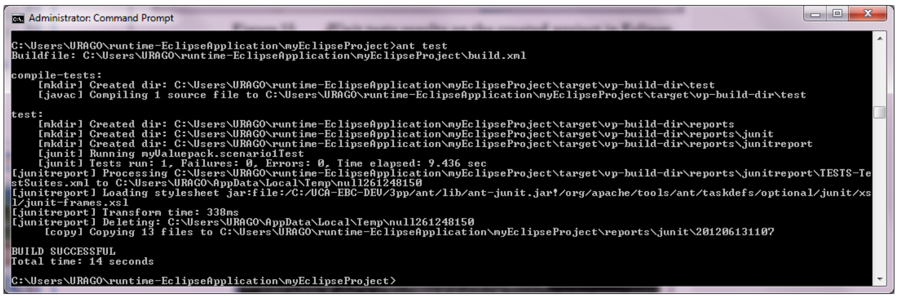
In which case the test results can be seen directly in Eclipse IDE:


Java - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Java
Resource
Package Explorer
JUnit
Finished after 9.346 seconds
Runs: 1/1
Errors: 0
Failures: 0
myValuepack.scenario1Test [Runner: JUnit 4] (4.013 s)
Failure Trace
Outline
An outline is not available.
Problems
Javadoc
Declaration
Console
Sign in to Google...

```
<terminated> myValuepack [JUnit] C:\Program Files\Java\jre6\bin\javaw.exe (Jun 13, 2012 11:03:39 AM)
Jun 13, 2012 11:03:49 AM org.apache.activemq.broker.TransportConnector stop
INFO: Connector tcp://localhost:61666 Stopped
Jun 13, 2012 11:03:49 AM org.apache.activemq.broker.BrokerService stop
INFO: ActiveMQ JMS Message Broker (localhost, ID:URAGO2-62229-1339578223317-0:(
Jun 13, 2012 11:03:49 AM org.springframework.jmx.support.ConnectorServerFactory
INFO: Stopping JMX connector server: javax.management.remote.rmi.RMIConnectorSe
Jun 13, 2012 11:03:49 AM org.springframework.remoting.rmi.RmiRegistryFactoryBea
INFO: Unexporting RMI registry
Jun 13, 2012 11:03:49 AM com.hp.uca.expert.scenario.internal.ScenarioImpl run
INFO:  Scenario Thread : Stop requested
Jun 13, 2012 11:03:49 AM com.hp.uca.expert.scenario.internal.ScenarioImpl run
INFO:  Scenario Thread : STOPPED
Jun 13, 2012 11:03:49 AM org.springframework.jmx.export.MBeanExporter destroy
INFO: Unregistering JMX-exposed beans on shutdown
Jun 13, 2012 11:03:49 AM myValuepack.scenario1Test tearDownAfterClass
INFO:  -- [END] -- myValuepack.scenario1Test
--------------------------------------------
```

Figure 12 - JUnit tests results on the created project in Eclipse IDE

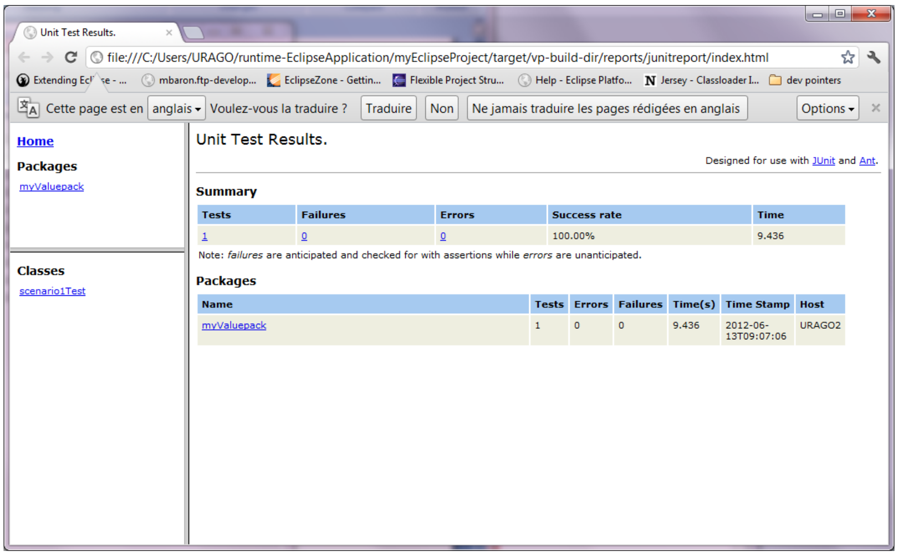
Or from the command line by executing the Apache Ant tool and selecting the "test" Ant target (You need to run the "**ant test**" command from the root directory of your project workspace) as shown in the following screen shot:



```
C:\Users\URAGO\runtime-EclipseApplication\myEclipseProject>ant test
Buildfile: C:\Users\URAGO\runtime-EclipseApplication\myEclipseProject\build.xml

compile-tests:
    [mkdir] Created dir: C:\Users\URAGO\runtime-EclipseApplication\myEclipseProject\target\vp-build-dir\test
    [javac] Compiling 1 source file to C:\Users\URAGO\runtime-EclipseApplication\myEclipseProject\target\vp-build-dir\test

test:
    [mkdir] Created dir: C:\Users\URAGO\runtime-EclipseApplication\myEclipseProject\target\vp-build-dir\reports
    [mkdir] Created dir: C:\Users\URAGO\runtime-EclipseApplication\myEclipseProject\target\vp-build-dir\reports\junit
    [mkdir] Created dir: C:\Users\URAGO\runtime-EclipseApplication\myEclipseProject\target\vp-build-dir\reports\junitreport
    [junit] Running myValuepack.scenario1Test
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 9.436 sec
[junitreport] Processing C:\Users\URAGO\runtime-EclipseApplication\myEclipseProject\target\vp-build-dir\reports\junitreport\TESTS-Te
stSuites.xml to C:\Users\URAGO\AppData\Local\Temp\null261248150
[junitreport] Loading stylesheet jar:file:/C:/UCA-EBC-DEV/3pp/ant/lib/ant-junit.jar!/org/apache/tools/ant/taskdefs/optional/junit/xs
l/junit-frames.xsl
[junitreport] Transform time: 338ms
[junitreport] Deleting: C:\Users\URAGO\AppData\Local\Temp\null261248150
     [copy] Copying 13 files to C:\Users\URAGO\runtime-EclipseApplication\myEclipseProject\reports\junit\201206131107

BUILD SUCCESSFUL
Total time: 14 seconds

C:\Users\URAGO\runtime-EclipseApplication\myEclipseProject>
```

Figure 13 - Running JUnit tests on the created project at the command-line using Ant

In which case the results can be shown in your preferred Web browser by opening the *index.html* file in the ***target\vp-build-dir\reports\junitreport*** directory of your project workspace:



Unit Test Results.

Designed for use with JUnit and Ant.

Summary

Tests	Failures	Errors	Success rate	Time
1	0	0	100.00%	9.436

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

Packages

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
myValuepack	1	0	0	9.436	2012-06-13T09:07:06	URAGO2

Figure 14 - JUnit tests results on the created project viewed using a Web browser

2.5 Customizing the created 'skeleton' Value Pack project

The project generated by the UCA for EBC project builder eclipse plug-in provides a simple scenario implementing some basic alarm statistics that is just here for validating the project structure.

Of course you have to turn the created 'skeleton' project into your new Correlation-project value pack. For this you have to customize

- The Value pack configuration files
- The scenario filter file
- The scenario rule files
- The Associated Java code files.

Note

☞ For additional information about Value Pack and Scenario configuration parameters, please refer to: [R2] *HP UCA for Event Based Correlation – Reference Guide*

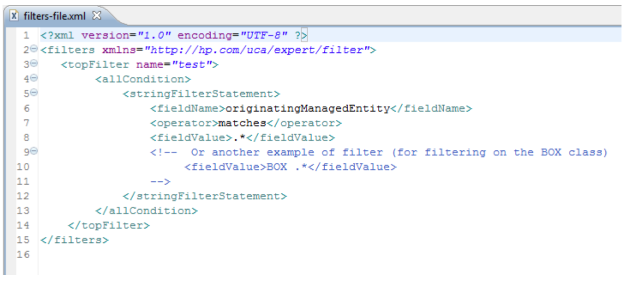
2.5.1 Updating the scenario filters

There is a filter file named `filters.xml` that is associated with the scenario of the created value pack.

The goal of this file is to define the passing filter for Alarms that will be consumed by the current scenario. Then, all alarms entering UCA for EBC will be evaluated against the filter file of each scenario, to decide if they should be forwarded to the scenario or not.

If the properties of an alarm match the passing filter(s) defined in the filters file then the alarm is forwarded to the scenario. On the other hand, if the properties of an alarm don't match the passing filter(s) of the filters file then the alarm is not forwarded to the scenario.

The default generated filter allows any alarm to be forwarded to the scenario.



```
<?xml version="1.0" encoding="UTF-8" ?>
<filters xmlns="http://hp.com/uca/expert/filter">
   <topFilter name="test">
        <allCondition>
            <stringFilterStatement>
                <fieldName>originatingManagedEntity</fieldName>
                <operator>matches</operator>
                <fieldValue>.*</fieldValue>
                <!--  Or another example of filter (for filtering on the BOX class)
                     <fieldValue>BOX .*</fieldValue>
                -->
            </stringFilterStatement>
        </allCondition>
    </topFilter>
</filters>
```

Figure 15 - The default "catch all" project's `filters.xml` file

Note

☞ In case you want to change the filter for the scenario or for additional information about scenario filter files, please refer to: [R2] *HP UCA for Event Based Correlation – Reference Guide*

2.5.2 Updating the correlation rules file

By default, the generated rules file defines a single rule implementing a basic statistic use case. This rule is just for demoing and testing. It is just an example, which must be changed to something relevant.

Note

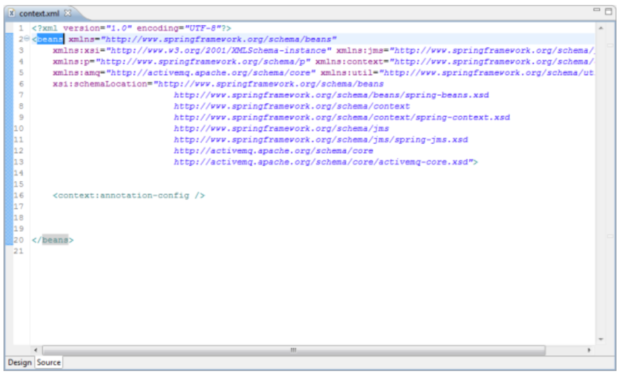
☞ For further information on the rule content (collect, retract, …), please refer to section "How to implement Alarm enrichment" of this document.

2.6 Advanced feature: Spring Framework integration

A Spring Framework *context.xml* file is provided in the src/main/resources/valuepack/conf folder. This file is defined for the whole "skeleton" value pack, i.e. it is common for all scenarios of the value pack.

All the Spring beans defined in this file will be available to each rule file of each scenario of the value pack.

By default the *context.xml* file is empty:



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/
    xmlns:p="http://www.springframework.org/schema/p" xmlns:context="http://www.springframework.org/schema/
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:util="http://www.springframework.org/schema/ut
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.xsd
                        http://www.springframework.org/schema/context
                        http://www.springframework.org/schema/context/spring-context.xsd
                        http://www.springframework.org/schema/jms
                        http://www.springframework.org/schema/jms/spring-jms.xsd
                        http://activemq.apache.org/schema/core
                        http://activemq.apache.org/schema/core/activemq-core.xsd">


    <context:annotation-config />



</beans>

```

Figure 16 - The default project's empty `context.xml` file

You can define any number of Spring beans in the context.xml file. These beans will be accessible from within the rules files through global variables defined in your rules files provided you follow the instructions explained in the following sections.

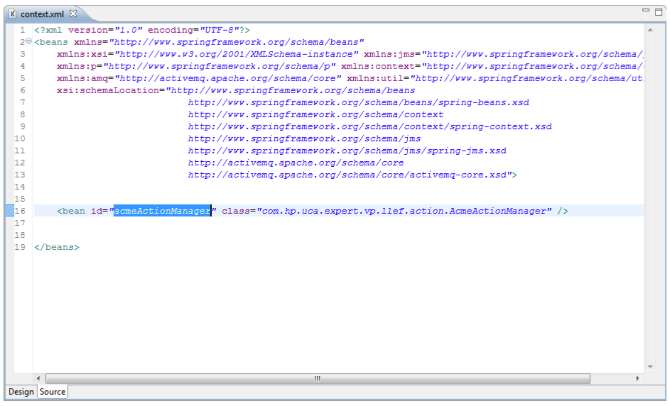
2.6.1 How to define and use Spring Beans inside rule files using global variables

The Spring "dependency injection" framework is useful for defining global variables (already initialized) in rules files. In a normal Drools environment, this is done through some Java code. As UCA hides the Drools session object, global variables are "injected" with Spring, from a XML definition (context.xml).

Comment [BO1]: It could be useful to precise that it is necessary to add this bean definition tag in the test context XML.

First you need to define your Spring beans in the *context.xml* file (the following sample file comes from the Low Level Event Filtering value pack and is described in the UCA for EBC Value Packs Examples guide)

The Spring beans that you define in the *context.xml* file are defined at the Value Pack level, and thus are global to all scenarios of the Value Pack:



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/
    xmlns:p="http://www.springframework.org/schema/p" xmlns:context="http://www.springframework.org/schema/
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:util="http://www.springframework.org/schema/ut
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.xsd
                        http://www.springframework.org/schema/context
                        http://www.springframework.org/schema/context/spring-context.xsd
                        http://www.springframework.org/schema/jms
                        http://www.springframework.org/schema/jms/spring-jms.xsd
                        http://activemq.apache.org/schema/core
                        http://activemq.apache.org/schema/core/activemq-core.xsd">


    <bean id="acmeActionManager" class="com.hp.uca.expert.vp.llef.action.AcmeActionManager" />


</beans>
```

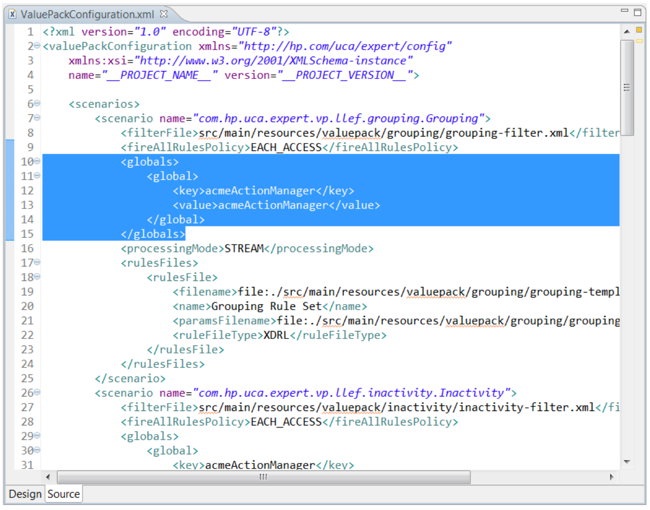
Figure 17 - The "Low Level Event Filtering" Value Pack's `context.xml` file

In the above screenshot, we define a Spring bean called acmeActionManager. This is just an example; with any other Spring bean, the process explained in the following paragraphs would have been the same.

Next we need to associate the Spring beans with global variables defined in your scenario. This is done in the *ValuePackConfiguration.xml* file that defines the configuration for all the scenarios of your value pack.

Note

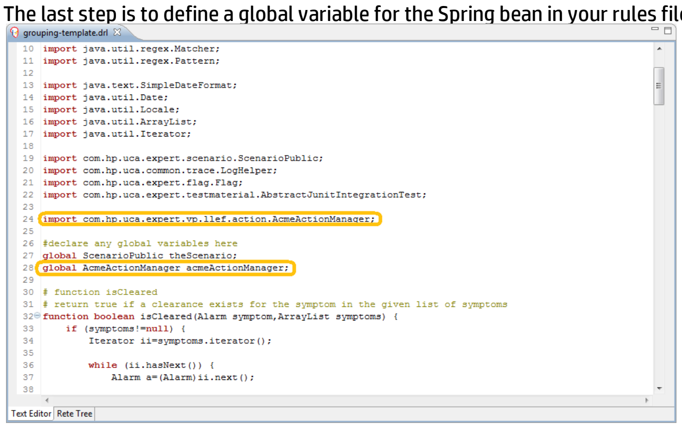
Although Spring beans are defined at the Value Pack level, global variables are defined at the scenario level. If you need a Spring bean to be global to all scenarios of your Value Pack, you need to configure the Spring bean as a global variable for each scenario of the Value Pack in the *ValuePackConfiguration.xml* file.



```
<?xml version="1.0" encoding="UTF-8"?>
<valuePackConfiguration xmlns="http://hp.com/uca/expert/config"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    name="__PROJECT_NAME__" version="__PROJECT_VERSION__">

    <scenarios>
        <scenario name="com.hp.uca.expert.vp.llef.grouping.Grouping">
            <filterFile>src/main/resources/valuepack/grouping/grouping-filter.xml</filter
            <fireAllRulesPolicy>EACH_ACCESS</fireAllRulesPolicy>
            <globals>
                <global>
                    <key>acmeActionManager</key>
                    <value>acmeActionManager</value>
                </global>
            </globals>
            <processingMode>STREAM</processingMode>
            <rulesFiles>
                <rulesFile>
                    <filename>file:./src/main/resources/valuepack/grouping/grouping-templ
                    <name>Grouping Rule Set</name>
                    <paramsFilename>file:./src/main/resources/valuepack/grouping/grouping
                    <ruleFileType>XDRL</ruleFileType>
                </rulesFile>
            </rulesFiles>
        </scenario>
        <scenario name="com.hp.uca.expert.vp.llef.inactivity.Inactivity">
            <filterFile>src/main/resources/valuepack/inactivity/inactivity-filter.xml</fi
            <fireAllRulesPolicy>EACH_ACCESS</fireAllRulesPolicy>
            <globals>
                <global>
                    <key>acmeActionManager</key>
```

Figure 18 - Defining global variables in the `ValuePackConfiguration.xml` file

The last step is to define a global variable for the Spring bean in your rules file:



```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.util.ArrayList;
import java.util.Iterator;

import com.hp.uca.expert.scenario.ScenarioPublic;
import com.hp.uca.common.trace.LogHelper;
import com.hp.uca.expert.flag.Flag;
import com.hp.uca.expert.testmaterial.AbstractJunitIntegrationTest;

import com.hp.uca.expert.vp.llef.action.AcmeActionManager;

#declare any global variables here
global ScenarioPublic theScenario;
global AcmeActionManager acmeActionManager;

# function isCleared
# return true if a clearance exists for the symptom in the given list of symptoms
function boolean isCleared(Alarm symptom,ArrayList symptoms) {
    if (symptoms!=null) {
        Iterator ii=symptoms.iterator();

        while (ii.hasNext()) {
            Alarm a=(Alarm)ii.next();

```

Figure 19 - Defining global variables in rules files

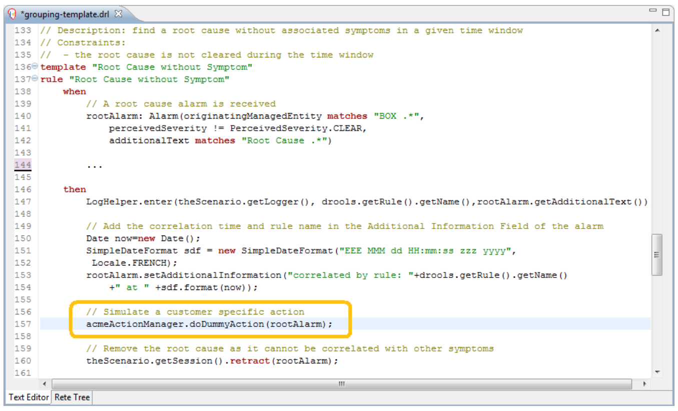
In the import section of your rules file, you need to add an “*import*” statement for the Java class of your Spring Bean:

```
import
com.hp.uca.expert.vp.llef.action.AcmeActionManager;
```

Then you need to add a “global” statement creating a global variable for your Spring Bean:

```
global AcmeActionManager acmeActionManager;
```

Then you can use the global variable in your rules:



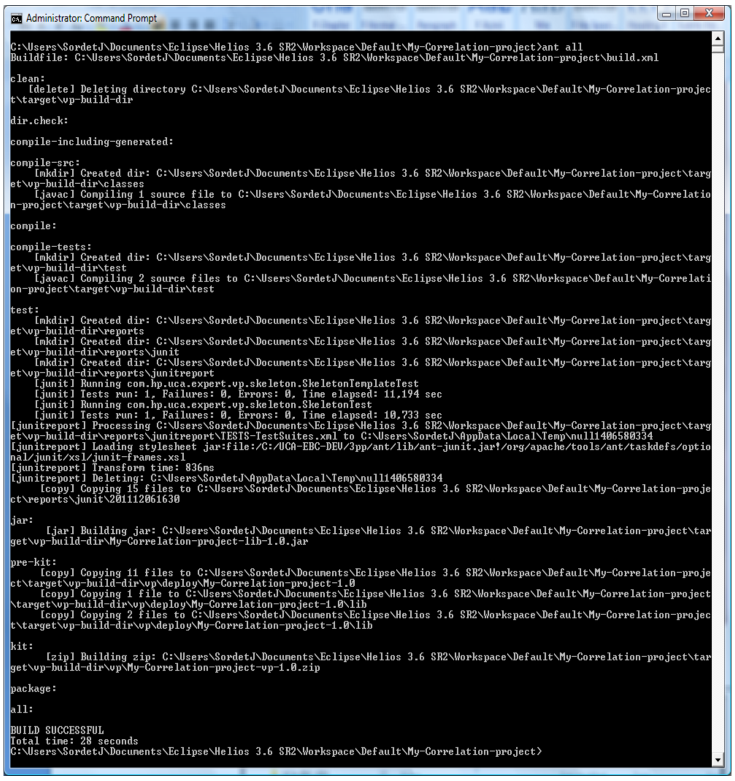
```
// Description: find a root cause without associated symptoms in a given time window
// Constraints:
//  - the root cause is not cleared during the time window
template "Root Cause without Symptom"
rule "Root Cause without Symptom"
    when
        // A root cause alarm is received
        rootAlarm: Alarm(originatingManagedEntity matches "BOX .*",
            perceivedSeverity != PerceivedSeverity.CLEAR,
            additionalText matches "Root Cause .*")

        ...

    then
        LogHelper.enter(theScenario.getLogger(), drools.getRule().getName(),rootAlarm.getAdditionalText())

        // Add the correlation time and rule name in the Additional Information Field of the alarm
        Date now=new Date();
        SimpleDateFormat sdf = new SimpleDateFormat("EEE MMM dd HH:mm:ss zzz yyyy",
         Locale.FRENCH);
        rootAlarm.setAdditionalInformation("correlated by rule: "+drools.getRule().getName()
            +" at " +sdf.format(now));

        // Simulate a customer specific action
        acmeActionManager.doDummyAction(rootAlarm);

        // Remove the root cause as it cannot be correlated with other symptoms
        theScenario.getSession().retract(rootAlarm);

```

Figure 20 - Using global variables in rules files

2.7 Generating the Value Pack kit

Once your project has been updated, it is necessary to generate the kit associated with it so that it can be deployed on UCA for EBC (this is the packaging phase). To do this, you just need to execute the following commands:

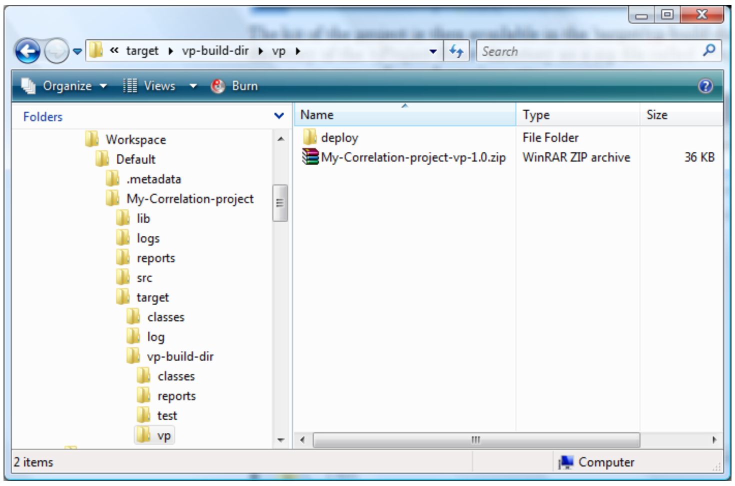
```
C:\> cd <Project Base>
...> ant all
```



```
C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-project>ant all
Buildfile: C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-project\build.xml

clean:
   [delete] Deleting directory C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-projec
t\target\vp-build-dir

dir.check:

compile-including-generated:

compile-src:
    [mkdir] Created dir: C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-project\targ
et\vp-build-dir\classes
    [javac] Compiling 1 source file to C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlatio
n-project\target\vp-build-dir\classes

compile:

compile-tests:
    [mkdir] Created dir: C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-project\targ
et\vp-build-dir\test
    [javac] Compiling 2 source files to C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlati
on-project\target\vp-build-dir\test

test:
    [mkdir] Created dir: C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-project\targ
et\vp-build-dir\reports
    [mkdir] Created dir: C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-project\targ
et\vp-build-dir\reports\junit
    [mkdir] Created dir: C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-project\targ
et\vp-build-dir\reports\junitreport
    [junit] Running com.hp.uca.expert.vp.skeleton.SkeletonTemplateTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 11,194 sec
    [junit] Running com.hp.uca.expert.vp.skeleton.SkeletonTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 10,733 sec
[junitreport] Processing C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-project\targ
et\vp-build-dir\reports\junitreport\TESTS-TestSuites.xml to C:\Users\SordetJ\AppData\Local\Temp\null1406580334
[junitreport] Loading stylesheet jar:file:/C:/UCA-EBC-DEV/3pp/ant/lib/ant-junit.jar!/org/apache/tools/ant/taskdefs/optio
nal/junit/xsl/junit-frames.xsl
[junitreport] Transform time: 836ms
[junitreport] Deleting: C:\Users\SordetJ\AppData\Local\Temp\null1406580334
     [copy] Copying 15 files to C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-proje
ct\reports\junit\201112061630

jar:
      [jar] Building jar: C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-project\tar
get\vp-build-dir\My-Correlation-project-lib-1.0.jar

pre-kit:
     [copy] Copying 11 files to C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-proje
ct\target\vp-build-dir\vp\deploy\My-Correlation-project-1.0
     [copy] Copying 1 file to C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-project
\target\vp-build-dir\vp\deploy\My-Correlation-project-1.0\lib
     [copy] Copying 2 files to C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-projec
t\target\vp-build-dir\vp\deploy\My-Correlation-project-1.0\lib

kit:
      [zip] Building zip: C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-project\tar
get\vp-build-dir\vp\My-Correlation-project-vp-1.0.zip

package:

all:

BUILD SUCCESSFUL
Total time: 28 seconds
C:\Users\SordetJ\Documents\Eclipse\Helios 3.6 SR2\Workspace\Default\My-Correlation-project>
```

Figure 21 - Building the kit of your customized Value Pack

The kit of the project is then generated in the `target/vp-build-dir/vp` directory of the <*Project Base*> directory as a zip file called *`<my valuepack name>-vp-<my valuepack version>.zip`*:



Name	Type	Size
deploy	File Folder	
My-Correlation-project-vp-1.0.zip	WinRAR ZIP archive	36 KB

Figure 22 - The kit of your customized Value Pack

The ZIP file of your customized Value Pack contains the following information:

- The Configuration (*`conf/`*) directory that contains:
    - The Value Pack Spring beans file: *`context.xml`*
    - The Value Pack configuration file: *`ValuePackConfiguration.xml`*
- The Library (*`lib/`*) directory that contains:
    - The JAR file of the Value Pack containing the compiled Java code that you developed for your Value Pack in addition to the rules files
    - Any custom JAR files that you need to run this Value Pack
- The Scenario (<*`your-scenario-name>/`*) directory that contains:
    - The filters file(s)
    - The external parameters file(s), if your Value Pack contains rules files that are template-based
    - The rule file(s)

```
$ unzip -l target/vp-build-dir/vp/myVP1-vp-1.0.zip
Archive:  target/vp-build-dir/vp/myVP1-vp-1.0.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  05-30-2013 17:46   myVP1-1.0/
        0  05-30-2013 17:46   myVP1-1.0/conf/
        0  05-30-2013 17:46   myVP1-1.0/lib/
        0  05-30-2013 17:46   myVP1-1.0/myScenario1/
     2726  05-30-2013 17:46   myVP1-
1.0/conf/ValuePackConfiguration.xml
     1100  05-30-2013 17:46   myVP1-1.0/conf/context.xml
     6423  05-30-2013 17:46   myVP1-1.0/lib/myVP1-lib-1.0.jar
     2596  05-30-2013 17:46   myVP1-1.0/myScenario1/Alarms.xml
      626  05-30-2013 17:46   myVP1-1.0/myScenario1/filters.xml
      420  05-30-2013 17:46   myVP1-
1.0/myScenario1/filtersTags.xml
     3299  05-30-2013 17:46   myVP1-1.0/myScenario1/rules.drl
---------                     -------
    17190                     11 files
```

Figure 23 - Contents of the ZIP file of your customized Value Pack

2.8 Deploying the Value Pack kit on UCA for EBC

To deploy your value pack in the UCA server, the following three steps are necessary:

1. Install the Value Pack ZIP file on UCA for EBC Server
2. Deploy the Value Pack on UCA for EBC Server
3. Start the Value Pack on UCA for EBC Server

The following paragraphs explain these three steps in detail:

1. Install the Value Pack package (ZIP file) on an HP Itanium or Linux system running UCA for EBC Server.

   Copy your Value Pack package (the ZIP file located at: *target/vp/<my value pack name>vp-<my value pack version>.zip*) to the *${UCA_EBC_INSTANCE}/valuepacks* directory on the UCA for EBC system, for example:

```
$ cp  target/vp-build-dir/vp/myVP1-vp-1.0.zip
${UCA_EBC_DATA}/instances/default/valuepacks/
```

2. Deploy the Value Pack in the *${UCA_EBC_INSTANCE}/deploy* directory using the "--deploy" option of the **uca-ebc-admin** administration tool (executed as **uca** user):

```
> cd ${UCA_EBC_HOME}/bin
> uca-ebc-admin --deploy -vpn <my value pack name> -vpv <my
value pack version>
```

You should get an output similar to the following:

```
UCA for EBC Home directory set to: /opt/UCA-EBC
UCA for EBC Data directory set to: /var/opt/UCA-EBC
INFO  - Value Pack name: <my value pack name> version: <my
value pack version> has been successfully deployed
INFO  - Exiting...
```

Note

☞ Alternatively, you can also deploy the value pack from the UCA for EBC GUI.

3. Start the Value Pack on UCA for EBC Server:

Two different ways are available to you to start value packs deployed on UCA for EBC depending on whether UCA for EBC is started or not.

You can check whether UCA for EBC is running or not by issuing the following command:

```
> ${UCA_EBC_HOME}/bin/uca-ebc show
```

If UCA for EBC is stopped, restarting UCA for EBC will load all value packs deployed in the `${UCA_EBC_INSTANCE}/deploy` folder including your value pack.

If UCA for EBC is running, use the "--start" option of the **uca-ebc-admin** administration tool (executed as **uca** user) to start your value pack:

```
> cd ${UCA EBC HOME}/bin
> uca-ebc-admin --start -vpn <my value pack name> -vpv
<my value pack version>
```

You should get an output similar to the following:

```
UCA for EBC Home directory set to: /opt/UCA-EBC
UCA for EBC Data directory set to: /var/opt/UCA-EBC
INFO  - Exiting...
```

Note

☞ Alternatively, you can also start the value pack from the UCA for EBC GUI.

You can get the list of running value packs on UCA for EBC using the “--list” option of the **uca-ebc-admin** command-line administration tool:

```
> cd ${UCA EBC HOME}/bin
> uca-ebc-admin --list
```

Note

☞ For additional information about the uca-ebc-admin command-line administration tool, please refer to**:** [R3] *HP UCA for Event Based Correlation – Administration, Configuration and Troubleshooting Guide*

[R4] *HP UCA for Event Based Correlation – Value Pack Examples*

2.9 Testing the Value Pack in real-time

Now that both UCA for EBC and your value pack are up and running, the UCA for EBC application implements the ‘Statistic circuit’ correlation package and is ready to listen to incoming alarms.

In order to provide an easy way to test the global solution, a simple tool is provided that lets you inject a set of alarms (defined in a XML file) into UCA for EBC.

As the action provided in the properties file is to “log” information to a log file (in “append” mode), it is easily possible to test the circuit in real-time.

A sample `Alarms.xml` input file containing sample alarms to use with your value pack is provided in the `${UCA_EBC_INSTANCE}/deploy/<your value pack name>-<your value pack version>/skeleton` folder. The output log file named `output.xml` is located in the `${UCA_EBC_HOME}` root folder.

Following is an example of how to use the **uca-ebc-injector** command-line tool to inject Alarms into UCA for EBC in order to test your Value Pack in real conditions:

```
>${UCA_EBC_HOME}/bin/uca-ebc-injector -file
${UCA_EBC_INSTANCE}/deploy/skeleton-project-
1.0/mypackage/Alarms.xml
>tail -f ${UCA_EBC_HOME}/output.xml &
```

You should get an output similar to the following:

```
### STATISTICAL ALARM: 2 Alarms received ###
```

Note

☞ For additional information about the uca-ebc-injector command-line tool, please refer to**:** [R3] *HP UCA for Event Based Correlation – Administration, Configuration and Troubleshooting Guide*

[R4] *HP UCA for Event Based Correlation – Value Pack Examples*

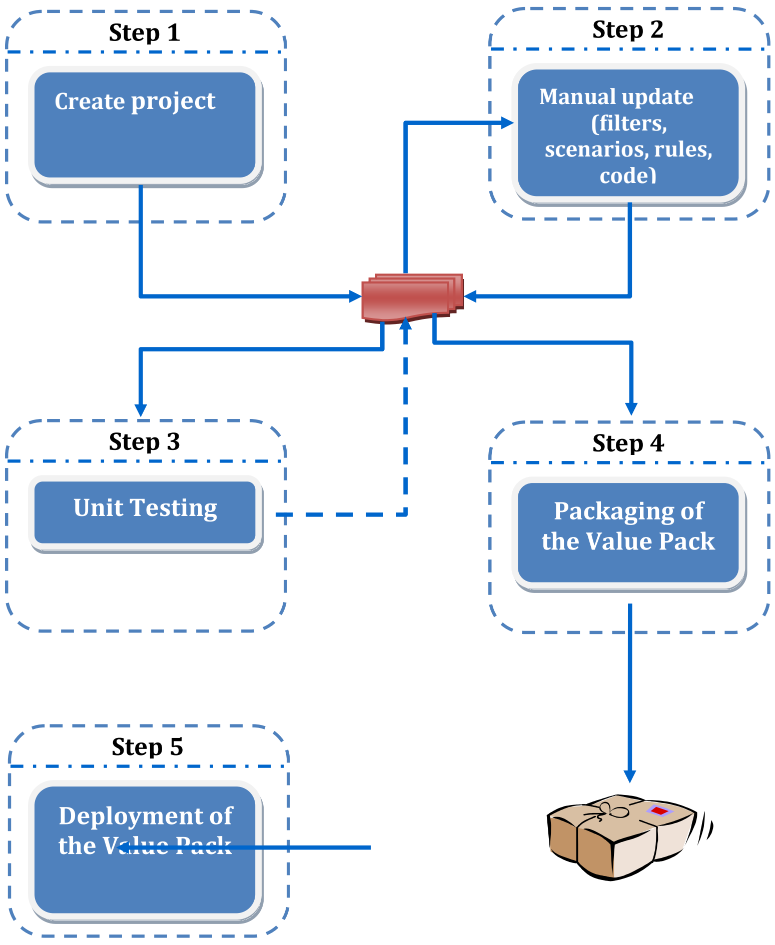
Developing a UCA for EBC Value Pack

3.1 Methodology

The following steps are needed to create a UCA for EBC Value Pack:

1. Create a new UCA for EBC Value Pack project based Thanks to the UCA for EBC project builder eclipse plug-in.
2. Update the UCA for EBC Value Pack project: scenario configuration, filters, and correlation rules.
3. Possibly, develop correlation rules can require developing (and testing) Java code. This third step is optional.
4. When all resources, rules and code are correctly updated and tested, the fourth step is the generation of the UCA for EBC Value Pack package (ZIP file).
5. The fifth and final step is the installation of this package on UCA for EBC Server. This phase is also called Value Pack deployment.

The process of creating a UCA for EBC Value Pack is described by the following figure:



Step 1
Create project
Step 2
Manual update (filters, scenarios, rules, code)
Step 3
Unit Testing
Step 4
Packaging of the Value Pack
Step 5
Deployment of the Value Pack

Figure 24 - The 5 steps to create a UCA for EBC Value Pack

For step 1 “Create a new UCA for EBC Value Pack project”, use the UCA for EBC project builder eclipse plug-in.

Step 2 “Update the UCA for EBC Value Pack project” is the main step when creating new UCA for EBC Value Packs. This part is explained in details in the next paragraphs and sections.

Step 3 “Develop correlation rules” is also a main step when creating new UCA for EBC Value Packs.

Step 4 is performed automatically using Apache Ant. The `build.xml` file has all necessary targets to compile, test, and generate a ZIP file for your Value Pack.

Step 5 involves copying your Value Pack zip file to the *`${UCA_EBC_INSTANCE}/valuepacks`* folder on a UCA for EBC Server, as mentioned in Chapter 2 “Getting started with UCA for EBC” of this document.

Developing correlation features involves creating one or more correlation scenarios for your Value Pack, each scenario using its own filter and implementing its own rules.

The following chapters will go into detail on the following subjects:

- **How to define a Value Pack**
- **How to define a Scenario**
- **How to define rules**
- **How to define and use template rules**

Then we will focus on “pure” development/coding:

- **How to use customized external actions**

And finally we will explain how to test your Value Pack:

- **How to create JUnit Tests**
- **How to inject events for integration and systems tests of your Value Pack**

3.2 How to define a Value Pack

Creating a Value Pack can be seen as implementing a “Correlation” bundle for managing a special correlation use case. The following are example of such correlation use cases:

- a Low Level Filtering use case
- a domain-specific correlation use case like IP MPLS or L2 Metro Ethernet
- a simple ‘operator’ use case that groups/correlates alarms based on specific rules

A Value Pack is a “functional container” that contains one or more scenarios, each scenario implementing a part of the whole correlation use case targeted by the Value Pack.

Scenarios can be cascaded so that the output of one scenario can be the input of another scenario.

Note

☞ For additional information about Value Pack and Scenario configuration parameters, please refer to: [R2] *HP UCA for Event Based Correlation – Reference Guide*

3.3 How to define a Scenario

A scenario is fully defined by implementing the following steps:

1. Defining the properties of the scenario
2. Defining the filter of the scenario (this will determine what type of alarms will enter the scenario)

3. Implementing Alarm enrichment processing (optional)
4. Implementing scenario rules

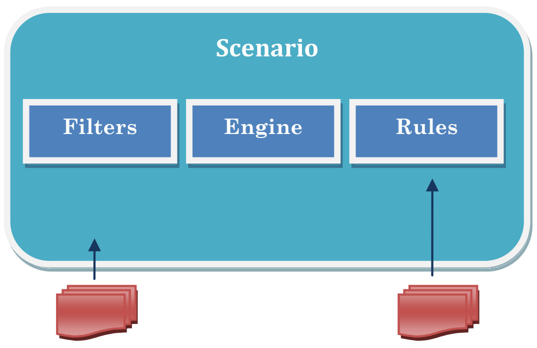


Figure 25 - The 3 steps to create a Scenario of a UCA for EBC Value Pack

Note

☞ The first two steps “Scenario definition file” and “Filter definition file” are described in the following document: [R2] *HP UCA for Event Based Correlation – Reference Guide*

The following sections will focus on step 3 “How to implement Alarm enrichment” and step 4 “How to implement the scenario rules”

3.4 How to implement Alarm enrichment

Alarm enrichment processing is called by the UCA for EBC framework after the alarm passed the scenario filter and before it is inserted in the scenario Working Memory.

The enrichment is implemented by performing the following steps:

1. **extend the UCA `com.hp.uca.expert.lifecycle.LifeCycleAnalysis` Java class and override the following methods:**
   - `onAlarmCreationProcess(Alarm alarm)`: to extend alarm creation objects
   - `onAlarmDeletionProcess(AlarmDeletion alarm)`: to extend alarm deletion objects
   - `onAlarmStateChangeProcess(AlarmStateChange alarm)`: to extend alarm state change objects
   - `onAlarmAttributeValueChangeProcess(AlarmAttributeValueChange alarm)`: to extend alarm attribute value change objects

Example of LifeCycleAnalysis Extension:

```
package com.hp.uca.ebc.enrichmentexample;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.hp.uca.common.trace.LogHelper;
import com.hp.uca.expert.alarm.Alarm;
import com.hp.uca.expert.alarm.AlarmCommon;
import com.hp.uca.expert.lifecycle.LifeCycleAnalysis;
import com.hp.uca.expert.scenario.Scenario;

public class ExtendedLifeCycle extends LifeCycleAnalysis {

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

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

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

            //  put the Alarm Enrichment Code here !!
            //  (standard alarm fields or LocalVariable)


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

In this example, the enrichment is performed only in the case of an alarm creation event.

2. **Declare the ExtendedLifeCycle class at the scenario definition Level :**

   This is done by using the <customLifeCycleClass> in the Scenario Definition section of the ValuepackConfiguration.xml file.

Example :

```
<scenarios>
 <scenario name="com.hp.uca.ebc.enrichmentexample.myscenario">
  <alarmEligibilityPolicy>
     NetworkState!=&quot;CLEARED&quot;
  </alarmEligibilityPolicy>
  <filterFile>
   src/main/resources/valuepack/myscenario/filters.xml
  </filterFile>
  <fireAllRulesPolicy>EACH_ACCESS</fireAllRulesPolicy>
  <globals></globals>
  <processingMode>CLOUD</processingMode>
  <rulesFiles>
      <rulesFile>
        <filename>
          file:./src/main/resources/valuepack/myscenario/rules.drl
        </filename>
        <name>my scenario rules</name>
        <ruleFileType>DRL</ruleFileType>
      </rulesFile>
  </rulesFiles>
  <customLifeCycleClass>
     com.hp.uca.ebc.enrichmentexample.ExtendedLifeCycle
  </customLifeCycleClass>
 </scenario>
</scenarios>
```

3. **Extend the Alarm object if necessary**

   In order to ease the rule writing, it may be easier to store the enrichment information in some dedicated alarm object attributes.

   In such case the Alarm objects (Alarm, AlarmDeletion, AlarmAttributeValueChange and AlarmStateChange) can be extended.

Example of Alarm extension :

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

@XmlRootElement
public class EnrichedAlarm extends Alarm {

    /**
     * New Alarm field
     */
    private String location;

    public EnrichedAlarm() {
            super();
    }

    public EnrichedAlarm (Alarm alarm) {
            super(alarm);
    }
    @Override
    public EnrichedAlarm clone() throws CloneNotSupportedException
    {
            EnrichedAlarm newAlarm = (EnrichedAlarm) super.clone();
            newAlarm.location = this.location;
            return newAlarm;
    }

    public String getLocation() {
            return location;
    }

    public void setLocation(String location) {
            this.location = location;
    }

    @Override
    public String toFormattedString() {
            StringBuffer toStringBuffer=
AlarmHelper.toFormattedStringBuffer(this);

            AlarmHelper.addFormatedItem(toStringBuffer, “Location:”,
getLocation());

            return toStringBuffer.toString();
    }

}
```

Example of LifeCycleAnalysis Extension using Alarm extension:

```
package com.hp.uca.ebc.enrichmentexample;

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

public class ExtendedLifeCycle extends LifeCycleAnalysis {

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

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

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

            EnrichedAlarm enrichedAlarm = new EnrichedAlarm (alarm);

            //  put the Alarm Enrichment Code here !!
            // enrichedAlarm.setLocation(“a location”);

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

```
        }
}
```

3.5 How to implement the scenario rules

Rules files are files that contain correlation rules that can be interpreted by the Drools inference engine of the scenario.

The Drool Expert engine used in UCA for EBC has its own rule language.

☞ Please refer to *Drools Expert guide, Chapter 5 The Rule Language* for a description of the language: http://www.jboss.org/drools/documentation

Important note

Drools keywords MUST NOT be used directly when developping UCA-EBC rules. This is for working memory integrity, and due to locking mechanism implemented within UCA-EBC framework.

In particular, all timer based keywords should be avoided: **duration, timer, calendar**.

On top of the basic rule language syntax, additional operators are available to deal with time constraints:

- Temporal operator: see *Drools Fusion guide, Chapter 2.4. Temporal Reasoning*
- Sliding Time Window Feature see *Drools Fusion guide, Chapter 2.6. Sliding Time Window*

☞ See http://www.jboss.org/drools/documentation for more information on how to create rules that deal with time constraints.

Note

To use the sliding time window feature, objects in working memory must be declared as Event (and not as Fact).

☞ Please see *Drools Fusion guide, Chapter 2.1. Events semantics* at URL http://docs.jboss.org/drools/release/5.3.0.Final/drools-fusion-docs/html/ch02.html#d0e184, for more information on what events are compared to facts and how to declare them.

3.5.1 Basics

Any rules file contains one or multiple rules, and has a '.drl' extension.

Here are the different parts composing a rule file:

```
package package-name

imports

globals
```

```
functions

queries

rules
```

Package

The package name is optional, but it is recommended to partition your rules in different packages for clarity.

Imports

The "imports" part, allows you to import Java classes that can be used in the Action or Condition parts of a rule.

Important note

In UCA for EBC, importing the Alarm Java class (com.hp.uca.expert.alarm.Alarm) is necessary in order to be able to use alarm attributes in rule conditions.

Globals

The "globals" part is used to define variables that have a global scope (across rules). The global variables have to be initialized by the application.

Functions

Functions let you define functions that let you avoid repeating the same lines of code over the entire rules file.

Queries

UCA for EBC does not currently provide support for queries.

Rules

The rules define the behavior of the expert system.

☞ Please refer to *Drools Expert guide,* for a full description of rule files: http://www.jboss.org/drools/documentation

3.5.2 Sample rules on Alarm facts in CLOUD mode

In CLOUD mode, the UCA for EBC system inserts Alarm facts in Working Memory and these facts remain infinitely in working memory unless they are specifically removed in the rules (using the retract statement). This retract statement is generally done in the right end side part of rules.

UCA for EBC contains an Alarm Java class (com.hp.uca.expert.alarm.Alarm) which represents a "generic" Alarm as a fact. Rules can rely on attributes and services of the Alarm object. For instance, testing a specific value of an attribute in the condition part or setting a specific attribute of the Alarm in the action part.

To use the CLOUD mode, the scenario processing mode must be set to "CLOUD" in the `ValuePackConfiguration.xml` file:

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

```
<valuePackConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 name="myValuepackName" version="myValuepackVersion">
 <scenarios>
     <scenario name="myScenario">
        <filterFile>${uca.home}/myValuePack/myScenario/myScenario-
filter.xml</filterFile>
        <fireAllRulesPolicy>WATCHDOG</fireAllRulesPolicy>
        <globals>
        </globals>
        <processingMode>CLOUD</processingMode>
        <rulesFiles>
            <rulesFile>
 <filename>file:${uca.home}/myValuePack/myScenario/myScenarioRules.drl</
filename>
                <name>myRules</name>
                <ruleFileType>DRL</ruleFileType>
            </rulesFile>
        </rulesFiles>
    </scenario>
 </scenarios>
</valuePackConfiguration>
```

Here is a simple example that identifies “Similar alarms” (i.e. Alarms that have the same alarm type, managed object and probable cause as another Alarm). This example illustrates a case where the UCA for EBC engine is in CLOUD processing mode.

The rule file called `myScenarioRules.drl` contains a rule, the “Similar Alarm” rule, which performs the following processing:

- When an alarm ‘a’ is found in Working Memory (with a severity different from ‘clear’) and if there is another not cleared (severity different from ‘clear’) alarm (this !=a) with the same attribute values for the originatingManageEntity, alarmType and probableCause properties then display a text.

```
package scenario.sample;

import com.hp.uca.expert.alarm.Alarm;
import com.hp.uca.expert.x733alarm.PerceivedSeverity;

rule "Similar Alarm"
when
  a: Alarm(perceivedSeverity != PerceivedSeverity.CLEAR)
  a1: Alarm(
      this != a &&
      perceivedSeverity != PerceivedSeverity.CLEAR &&
      originatingManagedEntity == a.originatingManagedEntity &&
      alarmType == a.alarmType &&
      probableCause == a.probableCause)

then
  System.out.println("Executing: "+drools.getRule().getName());
  System.out.println(a1.getIdentifier() + "similar to "+
a.getIdentifier());
end
```

Important note

Importing the Alarm Java class (com.hp.uca.expert.alarm.Alarm) is necessary. Declaring the Alarm class as a Fact in the "declare" section of the rules file is not mandatory however. By default, if they are not declared at all, objects are understood to be Facts in Working Memory.

Another rule, the "Clear Alarm" rule focuses on cleared alarms:

```
rule "Clear Alarm"
when
  a: Alarm(perceivedSeverity != PerceivedSeverity.CLEAR)
  a1: Alarm(
      perceivedSeverity == PerceivedSeverity.CLEAR &&
      originatingManagedEntity == a.originatingManagedEntity &&
      alarmType == a.alarmType &&
      probableCause == a.probableCause &&
      timeInMilliseconds > a.timeInMilliseconds)

then
  System.out.println("Executing: "+drools.getRule().getName());
  System.out.println(a1.getIdentifier() + " clears "+
a.getIdentifier());
end
```

Note

The **drools** object in the sample rule code above is a predefined Drools java object that you can use in the Action part of a rule to get information on the rule itself among other things. In our example, the method `drools.getRule().getName()`, called from a rule's Action part, returns the name of the rule. See http://docs.jboss.org/drools/release/5.3.0.Final/drools-expert-docs/html_single/#d0e5989 for more information on the **drools** predefined object.

3.5.3 Sample rules on Alarm events in STREAM mode

In STREAM mode, UCA for EBC inserts Alarm events in Working Memory only for a period of time. After that, Alarm events are automatically removed from working memory.

To use the STREAM mode, the scenario processing mode must be set to "STREAM" in the `ValuePackConfiguration.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<valuePackConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 name="myValuepackName" version="myValuepackVersion">
 <scenarios>
    <scenario name="myScenario">
         <filterFile>${uca.home}/myValuePack/myScenario/myScenario-
filter.xml</filterFile>
         <fireAllRulesPolicy>EACH_ACCESS</fireAllRulesPolicy>
         <globals>
         </globals>
         <processingMode>STREAM</processingMode>
         <rulesFiles>
             <rulesFile>

 <filename>file:${uca.home}/myValuePack/myScenario/myScenarioRules.drl</
filename>
```

```
                <name>myRules</name>
                <ruleFileType>DRL</ruleFileType>
             </rulesFile>
         </rulesFiles>
      </scenario>
  </scenarios>
</valuePackConfiguration>
```

Important note

Importing the Alarm Java class (com.hp.uca.expert.alarm.Alarm) is necessary. Declaring the Alarm class as an Event in the “declare” section of the rules file is also mandatory.

By default, if they are not declared at all, objects are understood to be Facts in Working Memory. So, declaring Alarms as Events is mandatory.

☞ Please see *Drools Fusion guide, Chapter 2.1. Events semantics* at URL http://docs.jboss.org/drools/release/5.5.0.Final/drools-fusion-docs/html/ch02.html#d0e184, for more information on what events are compared to facts and how to declare them.

```
import com.hp.uca.expert.alarm.Alarm;
import com.hp.uca.expert.x733alarm.PerceivedSeverity;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;


declare Alarm
    @role( event )
    @timestamp( timeInMilliseconds )
    @expires( 30m )
end
```

The above “Alarm” declaration specifies that:

1. Alarms should be treated as Events in Working Memory, not Facts
2. The timeInMilliseconds attribute (i.e. the EventTime attribute of the Alarm) is used as the timestamp of the Alarm instead of the time when the Alarm Event is actually inserted into working memory, which is the default timestamp for Events in Working Memory. The timestamp of the Alarm Event plays a role when time constraints are used in rules.
3. Alarm Events expiration time is 30 minutes: the Alarm Events will be removed from working memory automatically after 30 minutes.

Generally, rules in STREAM mode are used to identify patterns of Events (Events that occurs in a specific order) during a specific time window.

The “Store not cleared Alarm” rule is an example of such a rule in STREAM mode. It performs the following rules:

- When an alarm 'a' is in Working Memory (an alarm on a "BOX" item with a severity different from 'clear') and if there are no other alarms (matching specific criterias) received within 2 seconds of alarm 'a' then the AdditionalInformation attribute of alarm 'a' is updated

```
rule "Store not cleared Alarm"
when
        a: Alarm(originatingManagedEntity matches "BOX .*" &&
                perceivedSeverity != PerceivedSeverity.CLEAR)

        not Alarm(originatingManagedEntity ==
a.originatingManagedEntity &&
                perceivedSeverity == PerceivedSeverity.CLEAR &&
                this after[ 0s, 2s ] a)

then
        System.out.println("Executing rule:
"+drools.getRule().getName()+" on " + a.getAdditionalText());

        // Add the correlation time and rule name in the Additional
Information Field of the alarm
        Date now=new Date();
        SimpleDateFormat sdf = new SimpleDateFormat("EEE MMM dd
HH:mm:ss zzz yyyy",
         Locale.FRENCH);
        a.setAdditionalInformation("correlated by rule:
"+drools.getRule().getName()
                +" at " +sdf.format(now));

        // Store the alarm
        acmeActionManager.doDummyAction(a);
end
```

Note

The JBoss Drools documentation contains a lot of other examples of rules in both STREAM (Drools Fusion) and CLOUD (Drools Expert) modes. As writing the correlations rules is the major undertaking of creating a correlation project, it is highly recommended to constantly refer to the Drools documentation on how to write Rules.

☞ Please see http://www.jboss.org/drools/documentation for documentation on how to write rules for Drools Expert and Drools Fusion.

3.6 How to define and use rule templates

☞ For information about rule templates, please refer to: [R2] *HP UCA for Event Based Correlation – Reference Guide*

3.7 How to use Java code in the rules

Drools rules files natively support Java code in the consequence part of the rules (after the "then" keyword). All you have to do is import the packages/classes that you need in the import section of the rules files and then write Java code referencing these classes.

For example, you declare the java.util.Date class in the rules file:

```
template header
timeslot

package com.hp.uca.expert.vp.llef.grouping;

#list any import classes here.
import com.hp.uca.expert.alarm.Alarm;
import com.hp.uca.expert.x733alarm.PerceivedSeverity;

import java.util.regex.Matcher;
import java.util.regex.Pattern;
import com.hp.uca.expert.example.hibernate.AlarmDao;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.util.ArrayList;
import java.util.Iterator;

import com.hp.uca.expert.scenario.ScenarioPublic;
import com.hp.uca.common.trace.LogHelper;
import com.hp.uca.expert.flag.Flag;
import com.hp.uca.expert.testmaterial.AbstractJunitIntegrationTest;

#declare any global variables here
global AlarmDao alarmDAO;
global ScenarioPublic theScenario;
```

Then you can create and use java.util.Date objects in the consequence part (after the “then” keyword) of your rules:

```
// Description: find a root cause and the associated symptoms in a
given time window
// Constraints:
//      - the root cause is not cleared during the time window
template "Update Root Cause with Symptoms no clearance received"
rule "Update Root Cause with Symptoms no clearance received"

       when
               […]

       then
               LogHelper.enter(theScenario.getLogger(),
drools.getRule().getName(),rootAlarm.getOriginatingManagedEntity()+" -
"+ rootAlarm.getAdditionalText());

               // Add the correlation time and rule name in the
Additional Information Field of the alarm
               Date now=new Date();
               SimpleDateFormat sdf = new SimpleDateFormat("EEE MMM dd
HH:mm:ss zzz yyyy",
         Locale.FRENCH);
        String addInfo="correlated by rule:
"+drools.getRule().getName()
                      +" at " +sdf.format(now) + "\nAssociated
sympthoms:\n";
```

The java.util.Date objects that you create are not stored in Working Memory unless you do so explicitly using the “insert” statement.

Note

☞ For more information, please see the Drools documentation: http://www.jboss.org/drools/documentation

3.8 How to use external actions in rules

External actions in rules are basically any action that either uses OSS Open Mediation V6.2 framework services or external Java services.

There are two categories of external actions that we will describe in the following sections:

- **Standard external actions** that use the Action class, defined by the UCA for EBC framework, to execute actions on the OSS Open Mediation V6.2 framework
- **Customized external actions** that use external Java services (i.e. Java libraries) not provided by the core Java language packages. Spring beans (corresponding to the external Java services that you want to use) are defined in the `context.xml` of your Value Pack and global variables that reference these Spring beans are defined in your scenario(s) and used in your rule file(s).

3.8.1 Standard external actions

Standard external actions are defined as actions that are to be executed by the OSS Open Mediation V6.2 framework.

The UCA for EBC framework defines a Java class named Action that you can use to perform standard external actions in rules, like for example executing a shell script or a TeMIP directive on a TeMIP director.

In order to be able to use the methods of the Action class, you have to import the class in the "import" part of the rule file:

```
package com.hp.uca.expert.action;


#list any import classes here.
import com.hp.uca.expert.alarm.Alarm;
import com.hp.uca.expert.alarm.WMObjectStatus;
import com.hp.uca.expert.x733alarm.CustomFields;
import com.hp.uca.expert.x733alarm.CustomField;
import com.hp.uca.expert.x733alarm.PerceivedSeverity;
import com.hp.uca.expert.x733alarm.NetworkState;        // NOT_CLEARED,
CLEARED
import com.hp.uca.expert.x733alarm.OperatorState;       // NOT_ACKNOWLEDGED,
ACKNOWLEDGED, TERMINATED
import com.hp.uca.expert.x733alarm.ProblemState;        // NOT_HANDLED,
HANDLED, CLOSED
import com.hp.uca.mediation.action.client.Action;
import com.hp.uca.mediation.action.jaxws.ActionResponseItem;
import java.util.ArrayList;
```

Then you can create Action objects in the "then" part of a rule as described in the example below:

```
# Display properties of any new alarm

rule "Any Not Acknowledged Alarm (Action)"
 when
     a: Alarm(operatorState == OperatorState.NOT_ACKNOWLEDGED)
 then
     System.out.println("[RULE " + drools.getRule().getName() + "] Found not
acknowledged alarm: identifier = " + a.getIdentifier() + ", wmStatus = " +
a.getWmStatus() + ":");
     System.out.println(a.toFormattedString());

  // Acknowledging the Alarm
  Action action = new Action("TeMIP_AO_Directives_localhost ");
  action.addCommand("directiveName", "ACKNOWLEDGE");
  action.addCommand("entityName", a.getIdentifier());
  action.addCommand("UserId", "UCA Expert");
  theScenario.addAction(action); // Associate the action with the scenario
     System.out.println("Executing synchronous ACKNOWLEDGE directive on
alarm: " + a.getIdentifier());
  action.executeSync();
     System.out.println("Done:");
     System.out.println("    - ActionId = " + action.getActionId());
     System.out.println("    - ActionStatus = " + action.getActionStatus());
     System.out.println("    - ActionStatusExplanation = " +
action.getActionStatusExplanation());
     if (!action.getListActionResponseItem().isEmpty()) {
                    System.out.println("    - ActionResponseItems = ");
                    // Loop through all action response items
            for (ActionResponseItem item :
action.getListActionResponseItem()) {
                 if (!item.getOutput().getEntry().isEmpty()) {
                    // Loop through all output entries
                              for (ActionResponseItem.Output.Entry entry :
item.getOutput().getEntry()) {
                                   System.out.println("             -> " +
entry.getKey() + " = " + entry.getValue());
                              }
                          }
            }
     }
     else {
            System.out.println("    - ActionResponseItems = none");
     }
     System.out.println("    - RawText = " + action.getRawTextAsString());
 end
```

Basically you need to write the following code in your rule:

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

This will create a new Action object. There are 2 ways to create a new Action object:

- Either with the Action class constructor that takes an Action Reference parameter. The value of this parameter must match an Action Reference defined in `${UCA_EBC_INSTANCE}/conf/ActionRegistry.xml` file
- Or with the Action class constructor that takes the NMS Name, Service Name, Mvp Name and Mvp Version parameters. The Mvp Name and Version must match a Mediation Value Pack MvpName and MvpVersion attributes in the `${UCA_EBC_INSTANCE}/conf/ActionRegistry.xml` file

Here's the content of a sample `ActionRegistry.xml` file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ActionRegistryXML xmlns="http://registry.action.mediation.uca.hp.com/">

       <MediationValuePack MvpName="temip"
                           MvpVersion="1.0"

       url=http://localhost:26700/uca/mediation/action/ActionService?WSDL
brokerURL=" failover://tcp://localhost:10000">

              <Action actionReference="TeMIP_AO_Directives_localhost">
                     <ServiceName>aoDirective</ServiceName>
                     <NmsName>localTeMIP</NmsName>
              </Action>
              <Action actionReference="TeMIP_TT_Directives_localhost">
                     <ServiceName>ttDirective</ServiceName>
                     <NmsName>localTeMIP</NmsName>
              </Action>
              <Action actionReference="TeMIP_FlowManagement">
```

```
                        <ServiceName>subscriptionManagement</ServiceName>
                        <NmsName>localTeMIP</NmsName>
                </Action>
        </MediationValuePack>

        <MediationValuePack MvpName="exec"
                             MvpVersion="1.0"
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>

</ActionRegistryXML>
```

☞ Please refer to [R2] *HP UCA for Event Based Correlation – Reference Guide*

for more information on how to use the `Action` class or configure the `ActionRegistry.xml` file.

☞ Please refer to [R6] *OSS Open Mediation V601 Installation and Configuration Guide* for more information on how to configure OSS Open Mediation V6.2 to support the execution of Actions.

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

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

Using the addCommand() method you can specify the key/value pairs to use as parameters to the Action object. These parameters depend on the type of Action to perform.

For acknowledging a TeMIP Alarm, you need to specify the key/value pairs as shown above: specifying the UserId of the user acknowledging the alarm is optional, just like in TeMIP.

Then, you need to associate the Action to the current Scenario so that the Action can be properly processed:

```
theScenario.addAction(action);
```

Then, you need to execute the Action. Both synchronous and asynchronous actions are possible. Only one of the following lines of code is necessary, depending on whether you want to execute a synchronous or asynchronous action:

```
action.executeSync();

action.executeAsync(AODirectiveKey.ENTITY_NAME);
```

Synchronous actions are “blocking”. The action.executeSync() call will block the execution of the rule until the action is completed. The whole rule engine for the scenario is blocked while the action is being executed.

Asynchronous actions are “non blocking”. This is the reason why they are the recommended method for executing actions. The action.executeAsync(…) call doesn’t block the execution of the rule. The rules continue to be executed.

There’s a mandatory parameter to the action.executeAsync(…) method: the synchronizationKey. This key indicates the name of the action command key that will be used to synchronize asynchronous actions so that the order of asynchronous actions referring to the same action command key/value pair is preserved.

The synchronizationKey parameter enables you to preserve some kind of order among all the asynchronous actions triggered by your rules. By default (if you specify Action.NO_SYNCHRONIZATION_KEY as the synchronization key) there is no order. All asynchronous actions are executed in parallel by a pool of threads. There is no guarantee that the asynchronous actions will be executed in the order in which they were requested.

If you do not need asynchronous actions to be executed in any specific order, then you can use Action.NO_SYNCHRONIZATION_KEY as the synchronization key when calling the action.executeAsync(…) method.

On the other hand, if you need all asynchronous actions to be executed in the order they are requested, you need to use a command key (specified with the action.addCommand(key, value) method) that has the same value for all asynchronous actions as the synchronization key.

If you need only groups of asynchronous actions to be executed in the order they are requested, you need to use a command key (specified with the action.addCommand(key, value) method) that has the same value for all asynchronous actions of the same group as the synchronization key.

For example, for executing TeMIP AO Directives you can use the `AODirectiveKey.ENTITY_NAME` as synchronization key:

```
…
Action action = new
Action(“TeMIP_AO_Directives_localhost”);
action.addCommand(AODirectiveKey.DIRECTIVE_NAME,
AODirective.SET);
action.addCommand(AODirectiveKey.ENTITY_NAME,
“OPERATION_CONTEXT OC1 ALARM_OBJECT 155”);
action.addCommand(AODirectiveKey.ADDITIONAL_TEXT, “my
text”);
theScenario.addAction(action)
…
action.executeAsync(AODirectiveKey.ENTITY_NAME);
…
```

In the example above, as long as you execute TeMIP AO Directives using the `action.executeAsync(AODirectiveKey.ENTITY_NAME)` syntax, all TeMIP AO Directives actions on the same entity will be executed in the order that they are called.

If you do not want to use the synchronization key feature, you can pass null or Action.NO_SYNCHRONIZATION_KEY to the executeAsync(…) method:

```
…
```

```
action.executeAsync(Action.NO_SYNCHRONIZATION_KEY);
…
```

Once the action has been performed on the Network Management System the result of the execution of the action can be retrieved using the following methods:

```
action.getActionStatus();
action.getActionStatusExplanation();
```

Other methods of the Action class provide even more detailed information on the result of the execution of the action. See the *Java Documentation* for the Action class for more information.

3.8.1.1 How to write rules for the OSS Open Mediation TeMIP Value Pack

The delivered value pack examples come with a `lib/` directory containing the TeMIP mapper jar file:

```
lib/uca-mediation-temip-mvp-mapper-keys-3.0.jar
```

This will allow you to benefit from java classes that have been designed to help you write rules that execute TeMIP Alarm Object (AO) directives or TeMIP Trouble Ticket (TT) directives (provided the OSS Open Mediation V6.2 TeMIP Value Pack is deployed).

To do so, the first step is to add the following import statement in your rules file:

```
import com.hp.uca.temip.mvp.mapper.*;
```

Below is the list of classes that you can use to help you write rules (all AO classes are defined in the com.hp.uca.temip.mvp.aodirective.mapper package, while TT classes are defined in the com.hp.uca.temip.mvp.ttdirective.mapper package).

There are 2 sets of classes. The first set contains classes that define constants that should be used in the “key” part when using the Action.addCommand(key, value) method:

Class name	Class description
AODirectiveKey in com.hp.uca.temip.mvp.aodirective.mapper package	Contains string constants that list all the possible values for keys when using the Action.addCommand(key, value) method on AO Directives
TTDirectiveKey in com.hp.uca.temip.mvp.ttdirective.mapper package	Contains string constants that list all the possible values for keys when using the Action.addCommand(key, value) method on TT Directives

Table 4 - Java helper classes for OSS Open Mediation TeMIP Value Pack

The most important constant in the AODirectiveKey class is the AODirectiveKey.*DIRECTIVE_NAME* (or the TTDirectiveKey.*DIRECTIVE_NAME* in the TTDirectiveKey class depending on whether you want to execute AO or TT directives).

Using this constant, you can define the name of the TeMIP Alarm Object (or Trouble Ticket) directive that you wish to execute:

```
…
```

```
Action action = new
Action("TeMIP_AO_Directives_localhost");
action.addCommand(AODirectiveKey.DIRECTIVE_NAME,
AODirective.SET);
…
theScenario.addAction(action);
…
action.executeAsync(AODirectiveKey.ENTITY_NAME);
…
```

The other constants define the names of AO (or TT) Directive parameters or attributes that you can use. For example:

```
…
Action action = new
Action("TeMIP_AO_Directives_localhost");
action.addCommand(AODirectiveKey.DIRECTIVE_NAME,
AODirective.SET);
action.addCommand(AODirectiveKey.ENTITY_NAME,
"OPERATION_CONTEXT OC1 ALARM_OBJECT 155");
action.addCommand(AODirectiveKey.ADDITIONAL_TEXT, "my
text");
theScenario.addAction(action);
…
action.executeSync();
…
```

The second set contains classes that define constants that should be used in the "value" part when using the Action.addCommand(key, value) method.

Below is the list of such classes for Alarm Object directives (besides the AODirectiveKey class that is explained above):

Class name	Class description
AlarmClassType	Contains string constants that list all the possible values for the Alarm_Class attribute (of the SET directive for example). These constants should be used in the value part when using the Action.addCommand(key, value) method
AlarmObjectProblemStatus	Contains string constants that list all the possible values for the Problem_Status attribute (of the DUMP or SET directives for example)
AlarmObjectState	Contains string constants that list all the possible values for the State attribute (of the DUMP or SET directives for example) and the Previous_State attribute (of the SET directive for example)
AlarmOriginType	Contains string constants that list all the possible values for the Alarm_Origin attribute (of the SET directive for example)
AlarmType	Contains string constants that list all the possible values for the

	Alarm_Type attribute (of the CREATE, DUMP or SET directives for example)
AODirective	Contains string constants that list all the possible values for Alarm Object directive names (ACKNOWLEDGE, ADDPARENT, ARCHIVE, … for example)
AutomaticOperationsSeverity	Contains string constants that list all the possible values for the Automatic_Terminate_On_Close attribute (of the SET directive for example)
DeleteCondition	Contains string constants that list all the possible values for the State attribute (of the DELETE directive for example)
EntityScope	Contains string constants that list all the possible values for the entityScope attribute (of any directive)
EventID	Contains string constants that list all the possible values for the EventID attribute (of the GETEVENT directive for example)
Partition	Contains string constants that list all the possible values for the Partition attribute (of any directive)
ProbableCause	Contains string constants that list all the possible values for the Probable_Cause attribute (of the CREATE, DUMP or SET directives for example)
SecurityAlarmCause	Contains string constants that list all the possible values for the Security_Alarm_Cause attribute (of the CREATE, DUMP or SET directives for example)
Severity	Contains string constants that list all the possible values for the Severity (of the ARCHIVE directive for example), Perceived_Severity (of the CREATE, DELETE, DUMP, or SET directives for example), or Original_Severity (of the SET directive for example) attributes
SummarizeScope	Contains string constants that list all the possible values for the Scope attribute (of the DUMP directive for example)
TrendIndication	Contains string constants that list all the possible values for the Trend_Indication attribute (of the CREATE or SET directives for example)

Table 5 - AO directives helper classes

Below is the list of such classes for Trouble Ticket (TT_SERVER) directives (besides the TTDirectiveKey class that is explained above):

Class name	Class description
AttributeId	Contains string constants that list all the possible values for the AttributeId attribute (of the SHOW directive). These constants should be used in the value part when using the Action.addCommand(key, value) method
AutoResponseType	Contains string constants that list all the possible values for the Type attribute (of the ASSOCIATETT, CANCELTT, CLOSETT, CREATETT or DISSOCIATETT directives)
Partition	Contains string constants that list all the possible values for the Partition attribute (of any directive)
RegisterOperationType	Contains string constants that list all the possible values for the Operation attribute (of the REGISTER directive)

TTDirective	Contains string constants that list all the possible values for Trouble Ticket directive names (ASSOCIATETT, CANCELTT, CLEARALL, CLOSETT, CREATE … for example)

Table 6 - TT directives helper classes

The most important class in this set is the AODirective class (or the TTDirective class of Trouble Ticket directives) that lists all possible Alarm Object directive names (ACKNOWLEDGE, ADDPARENT, ARCHIVE, … for example):

```
…
Action action = new
Action("TeMIP_AO_Directives_localhost");
action.addCommand(AODirectiveKey.DIRECTIVE_NAME,
AODirective.SET);
…
theScenario.addAction(action);
…
action.executeAsync(AODirectiveKey.ENTITY_NAME);
…
```

The other classes contain constants that define the list of possible value for AO Directive (or TT Directive) parameters or attributes.

```
…
Action action = new
Action("TeMIP_AO_Directives_localhost");
action.addCommand(AODirectiveKey.DIRECTIVE_NAME,
AODirective.SET);
action.addCommand(AODirectiveKey.ENTITY_NAME,
"OPERATION_CONTEXT OC1 ALARM_OBJECT 155");
action.addCommand(AODirectiveKey.TREND_INDICATION,
TrendIndication.LESSSEVERE);
action.addCommand(AODirectiveKey.PROBABLE_CAUSE,
ProbableCause.LOSSOFSIGNAL);
theScenario.addAction(action);
…
action.executeSync();
…
```

You can use Eclipse IDE's automatic completion feature (the keyboard shortcut for this feature is: CTRL+<Space>) to discover the constants defined in each of the classes mentioned above.

3.8.1.2 How to write rules for the OSS Open Mediation Exec Value Pack

The delivered value pack examples come with a lib directory containing the TeMIP mapper jar file:

```
lib/uca-mediation-exec-mvp-mapper-keys-3.0.jar
```

To create an Exec Action for the OSS Open Mediation Exec Value Pack you must first add the following import statement in your rule file:

```
            import com.hp.uca.exec.mvp.mapper.*;
```

This will allow you to benefit from java classes that have been designed to help you write rules that execute command/executables/shell scripts (provided the OSS Open Mediation V6.2 Exec Value Pack is deployed).

Below is the list of classes that you can use to help you write rules (all classes are defined in the com.hp.uca.exec.mvp.mapper package):

Class name	Class description
ExecActionKey	Contains string constants that list all the possible values for keys when using the Action.addCommand(key, value) method

Table 7 - Java helper classes for OSS Open Mediation Exec Value Pack

Here's an example of how to use the ExecActionKey class:

```
…
Action action = new Action("Exec_localhost");
action.addCommand(ExecActionKey.COMMAND, "ping");
action.addCommand(ExecActionKey.ARGUMENT, "127.0.0.1");
…
theScenario.addAction(action);
…
action.executeSync();
…
```

3.8.2 Customized external actions

Most of the time, the actions performed in the THEN part of rules will be "external" actions. What we mean by "external" actions is non standard Java package services like print.

In other words, the question is how to use external services defined in "external" Java package such as Hibernate, JMS, …and generally initialized in the Spring configuration file of the Value Pack: `context.xml`.

The response is easy enough and is called "global". Any service defined in the application can be "retrieved" in any rule file using the "**global"** keyword.

Below is an excerpt from the Drools Expert documentation that explains the concept of global variables:

[…] With global you define global variables. They are used to make application objects available to the rules. Typically, they are used to provide data or services that the rules use, especially application services used in rule consequences, and to

return data from the rules, like logs or values added in rule consequences, or for the rules to interact with the application, doing callbacks. Globals are not inserted into the Working Memory, and therefore a global should never be used to establish conditions in rules except when it has a constant immutable value. The engine cannot be notified about value changes of globals and does not track their changes. Incorrect use of globals in constraints may yield surprising results - surprising in a bad way.

If multiple packages declare globals with the same identifier they must be of the same type and all of them will reference the same global value. [...]

☞ Please refer to the [R2] *HP UCA for Event Based Correlation – Reference Guide*

for more information about the Spring Framework integration with UCA for EBC.

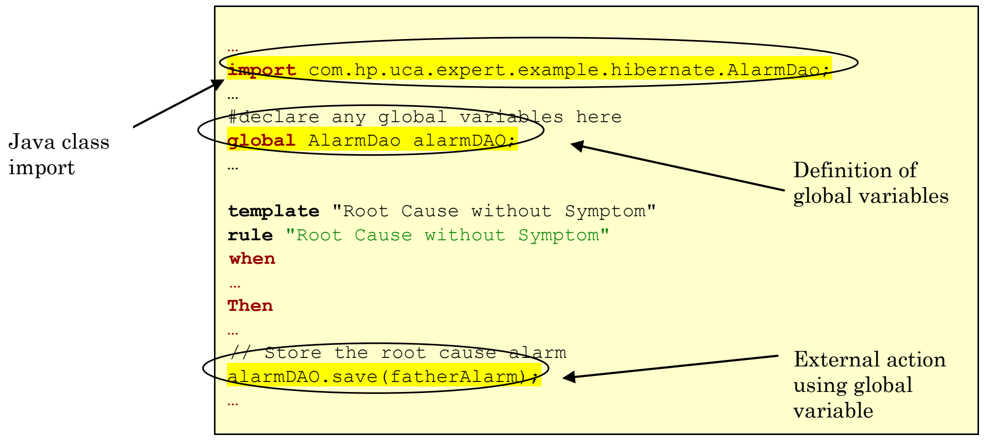
Firstly, in order to be able to use Spring beans in rules files, the Spring beans must be declared in the `context.xml` file of the Value Pack. Then global variable entries must be defined for each Spring bean in the `ValuePackConfiguration.xml` file as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<valuePackConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
       name="__PROJECT_NAME__" version="__PROJECT_VERSION__">
    <scenarios>
       <scenario name="Grouping-Scenario">

       <filterFile>src/main/resources/com/hp/uca/expert/vp/llef/groupin
g/grouping-filter.xml</filterFile>
               <fireAllRulesPolicy>EACH_ACCESS</fireAllRulesPolicy>
               <globals>
                       <global>
                               <key>alarmDAO</key>
                               <value>alarmDAO</value>
                       </global>
               </globals>
               <processingMode>STREAM</processingMode>
               <rulesFiles>
                   <rulesFile>
       <filename>file:./src/main/resources/com/hp/uca/expert/vp/llef/gr
ouping/grouping-template.drl</filename>
                       <name>grouping</name>

       <paramsFilename>file:./src/main/resources/com/hp/uca/expert/vp/l
lef/grouping/grouping-params.xml</paramsFilename>
                       <ruleFileType>XDRL</ruleFileType>
                   </rulesFile>
               </rulesFiles>
       </scenario>
    </scenarios>
</valuePackConfiguration>
```

The “globals” XML tag in the `ValuePackConfiguration.xml` file defines a list (i.e. a Java map) of beans that will be available in your rules file(s) as global variables.

The following piece of code illustrates how to define and use external Java libraries in rules files:

```
package com.hp.uca.expert.example.hibernate;

#list any import classes here.
import com.hp.uca.expert.alarm.Alarm;
import com.hp.uca.expert.x733alarm.PerceivedSeverity;
```



```
…
import com.hp.uca.expert.example.hibernate.AlarmDao;
…
#declare any global variables here
global AlarmDao alarmDAO;
…

template "Root Cause without Symptom"
rule "Root Cause without Symptom"
when
…
Then
…
// Store the root cause alarm
alarmDAO.save(fatherAlarm);
…
```

Java class import
Definition of global variables
External action using global variable

The purpose of this section is just to give you hints on how to integrate your rules files with external Java services.

3.8.2.1 How to forward alarms

A common use case is when you want to forward alarms being processed by a scenario to external systems/applications.

You might want to create an XML file containing some alarms that you want to export from the scenario so that you can import these alarms on an external system/application.

Alternatively, if the external system/application that you want to export alarms to has a JMS queue/topic that can be used to import alarms, then you might want to export alarms directly to this JMS queue/topic.

Finally, if the external system/application is accessible from OSS Open Mediation V6.2 via a specific Channel Adapter, then you might want to export the alarms directly to the OSS Open Mediation V6.2 bus.

The UCA for EBC framework defines standard classes that enable you forwarding Alarm objects (or collections thereof) located in Drools Working Memory or that have been defined in the rules of a scenario to either a file, a JMS queue/topic or OSS Open Mediation V6.2.

The following Java classes are part of the UCA for EBC framework:

- To forward alarms to a file: com.hp.uca.expert.alarm.FileAlarmForwarder
- To forward alarms to a JMS queue/topic: com.hp.uca.expert.alarm.JMSAlarmForwarder
- To forward alarms to OSS Open Mediation V6.2: com.hp.uca.expert.alarm.OpenMediationAlarmForwarder

☞ Please refer to UCA for EBC Javadoc for complete information on these classes. The Javadoc for UCA for EBC is located both at *%UCA_EBC_DEV_HOME%\apidoc* and at *${UCA_EBC_HOME}/apidoc*.

One way to forward alarms is to define an AlarmForwarder (either FileAlarmForwarder, JMSAlarmForwarder, or OpenMediationAlarmForwarder) bean in the Spring configuration file of the scenario (`context.xml`).

Note

Please note that the recommended way of defining alarm forwarders is to define them in the Spring configuration file of the scenario: `context.xml`.

A Thread is associated with each alarm forwarder (either FileAlarmForwarder, JMSAlarmForwarder, or OpenMediationAlarmForwarder). This thread is automatically started when the associated FileAlarmForwarder, JMSAlarmForwarder, or OpenMediationAlarmForwarder object is created. If the FileAlarmForwarder has been created using the recommended method (in the Spring configuration file of the scenario: `context.xml`) then the associated thread will be automatically stopped when the bean associated with the alarm forwarder is destroyed. Otherwise you need to use the requestStop() method to explicitly stop the thread associated with the alarm forwarder when you don't need it anymore.

The thread associated with an alarm forwarder provides compression to improve performance. Alarms are not forwarded right away. They are accumulated in a queue for the duration of the compression period (by default 1 second) so that they can be forwarded as a batch of alarms at the end of the compression period (by default every second). You can change the value of the compression period using the setCompressionPeriod(long) method. If you set the compression period to 0 milliseconds, no compression will be performed.

Here's an example of how to define such a bean in the `context.xml` file of a scenario:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jms="http://www.springframework.org/schema/jms"
       xmlns:p="http://www.springframework.org/schema/p"
xmlns:context="http://www.springframework.org/schema/context"
       xmlns:amq="http://activemq.apache.org/schema/core"
xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd
                           http://www.springframework.org/schema/jms

http://www.springframework.org/schema/jms/spring-jms.xsd
                           http://activemq.apache.org/schema/core

http://activemq.apache.org/schema/core/activemq-core.xsd">


       <context:annotation-config />

       <bean name="forwardedAlarmsFile" class="java.io.File">
              <constructor-arg index="0"><value>forwarded-
alarms.xml</value></constructor-arg><!-- String pathname -->
       </bean>
```

```
        <bean name="fileAlarmForwarder"
class="com.hp.uca.expert.alarm.FileAlarmForwarder" depends-
on="forwardedAlarmsFile">
                <constructor-arg index="0"><ref
bean="forwardedAlarmsFile"/></constructor-arg><!-- File file -->
                <constructor-arg
index="1"><value>false</value></constructor-arg><!-- boolean overwrite
-->
        </bean>

        <bean name="jmsAlarmForwarder"
class="com.hp.uca.expert.alarm.JMSAlarmForwarder">
                <constructor-arg
index="0"><value>vm://localhost?broker.persistent=false</value></constr
uctor-arg><!-- String brokerURL -->
                <constructor-arg
index="1"><value>jms.alarm.forwarder.test.queue</value></constructor-
arg><!-- String destinationName -->
                <constructor-arg
index="2"><value>true</value></constructor-arg><!-- boolean isQueue -->
        </bean>

        <bean name="openMediationAlarmForwarder"
class="com.hp.uca.expert.alarm.OpenMediationAlarmForwarder">
                <constructor-arg index="0"><value>UCA-
EBC_remotesystem</value></constructor-arg><!-- String actionReference -
->
                <constructor-arg index="1"><value>Alarm Flow from UCA
EBC</value></constructor-arg><!-- String alarmFlowName -->
        </bean>
</beans>
```

Figure 26 - Defining AlarmForwarder beans in the `context.xml` file

The highlighted portion of the `context.xml` file shows the definition of a FileAlarmForwarder bean that will be used in the rule files of a scenario to forward alarms to an XML file.

Once the `context.xml` file has been properly set up, you need to define global variable entries in the `ValuePackConfiguration.xml` file for each Spring bean that you want to access from the rules as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<valuePackConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
        name="__PROJECT_NAME__" version="__PROJECT_VERSION__">

   <scenarios>
       <scenario name="alarmforwarder">

<filterFile>src/main/resources/valuepack/alarmforwarder/filters.xml</fi
lterFile>
       <fireAllRulesPolicy>EACH_ACCESS</fireAllRulesPolicy>
       <globals>
                       <global>
                               <key>fileAlarmForwarder</key>
                               <value>fileAlarmForwarder</value>
                       </global>
                       <global>
                               <key>jmsAlarmForwarder</key>
                               <value>jmsAlarmForwarder</value>
                       </global>
                       <global>
                               <key>openMediationAlarmForwarder</key>
                              <value>openMediationAlarmForwarder</value>
                       </global>
        </globals>
        <processingMode>STREAM</processingMode>
        <rulesFiles>
            <rulesFile>
        <filename>file:./src/main/resources/valuepack/alarmforwarder/ala
rmforwarder.drl</filename>
```

```
                              <name>alarmforwarder rules</name>
                              <ruleFileType>DRL</ruleFileType>
                 </rulesFile>
             </rulesFiles>
            </scenario>
        …
    </scenarios>
</valuePackConfiguration>
```

Figure 27 - Defining AlarmForwarder globals in the *`ValuePackConfiguration.xml`* file

The highlighted portion of the `ValuePackConfiguration.xml` file shows the definition of a fileAlarmForwarder global variable referencing the fileAlarmForwarder Spring bean defined in the `context.xml` file that will be used in the rule files of a scenario to forward alarms to an XML file.

Once the `ValuePackConfiguration.xml` file has been properly set up, you need to make some modifications to the rule files where you want to use the fileAlarmForwarder global variable:

- Import the proper Java class:
    - com.hp.uca.expert.alarm.FileAlarmForwarder for a FileAlarmForwarder
    - com.hp.uca.expert.alarm.JMSAlarmForwarder for a JMSAlarmForwarder
    - com.hp.uca.expert.alarm.OpenMediationAlarmForwarder for an OpenMediationAlarmForwarder
- Declare the global variables (defined in the `ValuePackConfiguration.xml` file) that you want to use in the rule file

Below is an example of how to import the proper Java class, and declare the global variables that you want to use:

```
package com.hp.uca.expert.vp.alarmforwarder;

#list any import classes here.
import com.hp.uca.expert.alarm.Alarm;
import com.hp.uca.expert.alarm.AlarmDeletion;
import com.hp.uca.expert.alarm.AlarmStateChange;
import com.hp.uca.expert.alarm.AlarmAttributeValueChange;
import com.hp.uca.expert.x733alarm.PerceivedSeverity;
import java.util.ArrayList;
import com.hp.uca.expert.scenario.Scenario;
import com.hp.uca.common.trace.LogHelper;
import com.hp.uca.expert.flag.Flag;
import com.hp.uca.expert.testmaterial.AbstractJunitIntegrationTest;
import com.hp.uca.expert.alarm.FileAlarmForwarder;
import com.hp.uca.expert.alarm.JMSAlarmForwarder;
import com.hp.uca.expert.alarm.OpenMediationAlarmForwarder;

#declare any global variables here
global Scenario theScenario;
global FileAlarmForwarder fileAlarmForwarder;
global JMSAlarmForwarder jmsAlarmForwarder;
global OpenMediationAlarmForwarder openMediationAlarmForwarder;

declare Alarm
    @role( event )
    @timestamp( timeInMilliseconds )
    @expires( 30m )
end
```

Figure 28 - Declaring the use of an AlarmForwarder global variable in a rule file

Once the proper Java classes have been imported and the global variables declared, you can just use global variable to write Alarms (or collections of Alarms) to an XML file (the one specified in the `context.xml` file):

```
…
import com.hp.uca.expert.alarm.FileAlarmForwarder;
import com.hp.uca.expert.alarm.JMSAlarmForwarder;
import com.hp.uca.expert.alarm.OpenMediationAlarmForwarder;

#declare any global variables here
global Scenario theScenario;
global FileAlarmForwarder fileAlarmForwarder;
global JMSAlarmForwarder jmsAlarmForwarder;
global OpenMediationAlarmForwarder openMediationAlarmForwarder;

declare Alarm
    @role( event )
    @timestamp( timeInMilliseconds )
    @expires( 30m )
end


# Forward any alarm received
rule "Forward any alarm received"
no-loop
       when
        $alarm : Alarm()
       then
       LogHelper.enter(theScenario.getLogger(),
drools.getRule().getName());

       // Forward the alarm to a file, jms queue/topic or OSS Open
Mediation
       fileAlarmForwarder.write($alarm);
       // Forward the alarm to a jms queue or topic
       jmsAlarmForwarder.write($alarm);
       // Forward the alarm to OSS Open Mediation
       openMediationAlarmForwarder.write($alarm);

       // Retract the alarm
       theScenario.getLogger().info("Retracting: \n"+
$alarm.toFormattedString());
       theScenario.getSession().retract($alarm);

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

Figure 29 - Using an AlarmForwarder global variable to write Alarms to an XML file

The XML file generated by the FileAlarmForwarder is fully compatible with the XML schema for UCA for EBC Alarms defined at *`${UCA_EBC_HOME}/schemas/uca-expert-alarm.xsd`*. For example, the generated XML file containing the alarms can be used as input to the *`${UCA_EBC_HOME}/bin/uca-ebc-injector`* command-line tool.

The JMSAlarmForwarder on the other hand can be used to forward alarms directly to a JMS queue/topic, for example the Alarm input queue of a UCA for EBC server (which is implemented as a JMS Topic). You can use the following values to forward alarms to a UCA for EBC alarm input queue:

- **brokerURL**: JMSAlarmForwarder.*DEFAULT_UCA_EBC_BROKER_URL* (the value of this constant is “tcp://localhost:61666”)
- **destinationName**: JMSAlarmForwarder. *DEFAULT_UCA_EBC_ALARMS_TOPIC_NAME* (the value of this constant is “com.hp.uca.ebc.alarms”)

- **isQueue**: false (because the UCA for EBC alarm input queue is in fact a JMS topic, not a JMS queue)

Finally the OpenMediationAlarmForwarder can be used to forward alarms to OSS Open Mediation V6.2. In order to use an OpenMediationAlarmForwarder, you must first create an action reference in the *${UCA_EBC_INSTANCE}/conf/ActionRegistry.xml* file that will define how to connect to the UCA for EBC Channel Adapter on OSS Open Mediation V6.2, and how to reach to Channel Adapter of the system/application that you target.

Below is an example of an action reference defined in the *ActionRegistry.xml* file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ActionRegistryXML
xmlns="http://registry.action.mediation.uca.hp.com/">

        <MediationValuePack MvpName="ApplicationX" MvpVersion="1.1"

        url="http://localhost:26700/uca/mediation/action/ActionService?W
SDL"
                brokerURL="failover://tcp://localhost:10000">

                <Action actionReference="ApplicationX_remotesystem">
                        <ServiceName>applicationX-1.1</ServiceName>
                        <NmsName>remotesystem</NmsName>
                </Action>
        </MediationValuePack>
</ActionRegistryXML>
```

In the sample *ActionRegistry.xml* file above, an action reference has been defined for an “ApplicationX” application on a remote system connected to OSS Open Mediation V6.2 via an ApplicationX Channel Adapter (ApplicationX is a fictitious application).

The brokerURL attribute must match the URL of the ActiveMQ broker defined for the OSS Open Mediation V6.2 that you target. The hostname in the URL must match the hostname of the system where OSS Open Mediation V6.2 is installed. By default the port number used for the ActiveMQ broker on OSS Open Mediation V6.2 container instance 0 is 10000.

To verify what port number is used for your OSS Open Mediation V6.2 container instance, please check the value of the activemq.port property in the */var/opt/openmediation-V60/containers/instance-<instance number>/conf/servicemix.properties* file.

The following JMS properties will be set for the alarms being forwarded to OSS Open Mediation V6.2. These properties can be used by consumer Channel Adapters to filter the alarms that they’re interested in among all alarms pushed by various Channel Adapters to the OSS Open Mediation V6.2 alarms JMS topic:

JMS Property Name	Value
NOMOriginalProvider	set to the value of ${ca.name} in UCA EBC CA
NOMOriginalProviderEndpoint	“UCA EBC *version* on *hostname*”
NOMOriginalProviderPort	*not set*
NOMOriginalProviderHost	set to the value of ${nom_hostname} in UCA EBC CA

NOMOriginalProviderContainerInstanceNumber	set to the value of ${sys.nom_instance_number} in UCA EBC CA
NOMType	set to "http://hp.com/openmediation/alarms/2011/08" in UCA EBC CA
NOMActionMessageType	*not set (this is not an action message, this is an alarm message)*
NOMActionEntityHint	*not set (this is not an action message, this is an alarm message)*
NOMActionNameHint	*not set (this is not an action message, this is an alarm message)*
NOMFinalConsumer	the value of the "*serviceName"* attribute of the action reference (in the `ActionRegistry.xml` file) associated with the OpenMediationAlarmForwarder object
NOMFinalConsumerEndpoint	"*mvpName mvpVersion* on *nmsName*", where the names in *italics* are XML entities/attributes of the action reference (in the `ActionRegistry.xml` file) associated with the OpenMediationAlarmForwarder object
NOMFinalConsumerPort	"*alarmFlowName*" associated with the OpenMediationAlarmForwarder object or "UCA EBC Alarms" by default. You can set the FlowName attribute when you create the OpenMediationAlarmForwarder object
NOMFinalConsumerHost	the value of the "*nmsName*" XML entity of the action reference (in the `ActionRegistry.xml` file) associated with the OpenMediationAlarmForwarder object
NOMFinalConsumerConstainerInstanceNumber	*not set*

Table 8 - JMS properties set for alarms being forwarded to OSS Open Mediation

3.9 How to make useful logs

☞ Please refer to [R2] *HP UCA for Event Based Correlation – Reference Guide*

for more information on how to use the Scenario Loggers.

3.10 How to create JUnit Tests

Developing Value Packs involves creating correlation rules and writing code. In any case, it is highly recommended to unit test your rules and code.

To help you in that regard, the 'skeleton' project (the project created by the UCA Eclipse plug-in) provides you with a template of a JUnit test (based on JUnit 4.4) along with the complete infrastructure to compile, run and generate reports for unit tests.

The following JUnit test is a good starting point to create new unit tests:

- It is a JUnit 4.4 test that also supports Java and Spring framework annotations: using @RunWith and @Configuration annotations automatically loads the associated Spring configuration file (called `<test file name>-context.xml`)

- The template JUnit test class that we provide extends the **AbstractJunitIntegrationTest** class. This class is part of the UCA for EBC framework. It implements the Spring framework **ApplicationContextAware** interface, and thus provides access to the Spring beans (Java objects) defined in the Spring configuration file(called `<test file name>-context.xml`). You can easily retrieve any Spring bean defined in the Spring configuration file by using the **getApplicationContext().getBean(*String name*)** method from any JUnit test class that extends the AbstractJunitIntegrationTest class.
- In JUnit 4.4, any method that represents a unit test needs to have the @Test annotation before the definition of the method.
- It is mandatory to define a Testsuite so that tests can be found in the Apache Ant project of your Value Pack. Defining the following method allows for automatic retrieval of all tests defined in the unit test class:

```
// Way to run tests via ANT Junit
public static junit.framework.Test suite() {
  return new JUnit4TestAdapter(SkeletonTest.class);
}
```

Below is the code for the template JUnit test class:

```
package com.hp.uca.expert.vp.skeleton;

import junit.framework.JUnit4TestAdapter;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.hp.uca.common.misc.Constants;
import com.hp.uca.common.trace.LogHelper;
import com.hp.uca.expert.testmaterial.AbstractJunitIntegrationTest;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration
public class SkeletonTest extends AbstractJunitIntegrationTest{

  private static Log log = LogFactory.getLog(SkeletonTest.class);
  private static final String SCENARIO_BEAN_NAME = "skeleton";
  private static final String ALARM_FILE =
"src/test/resources/com/hp/uca/expert/vp/skeleton/Alarms.xml";

  /**
   * @throws java.lang.Exception
   */
  @BeforeClass
  public static void setUpBeforeClass() throws Exception {
        log.info(Constants.TEST_START.val() + SkeletonTest.class.getName());
  }
  /**
   * @throws java.lang.Exception
   */
  @AfterClass
  public static void tearDownAfterClass() throws Exception {
          log.info(Constants.TEST_END.val() + SkeletonTest.class.getName()
                        + Constants.GROUP_ALT1_SEPARATOR.val());
  }

  // Way to run tests via ANT Junit
  public static junit.framework.Test suite() {
          return new JUnit4TestAdapter(SkeletonTest.class);
  }

  @Test
  public void test() throws Exception {
          LogHelper.enter(log, "test()");
          /*
           * Initialize variables and Enable engine internal logs
           */
          initTest(SCENARIO_BEAN_NAME, BMK_PATH);

          /*
           * Send alarms
           */
          getProducer().sendAlarms(ALARM_FILE);

          /*
           * Waiting for the TEST_END FLag that should be inserted by the rule
           * itself
           */
          waitingForTheEndTestFlag(getFlagEventListener(),1 * SECOND,10*SECOND);

          /*
           * Disable Rule Log to close the file used to compare engine historical
           * events
           */
          closeRuleLogFiles(getScenario());

          /*
           * Check test result by comparing the historical engine events with a
           * benchmark
           */
          checkTestResult(getLogRuleFileName(),getLogRuleFileNameBmk());

          LogHelper.exit(log, "test()");
  }
}
```

Note

The `AbstractJunitIntegrationTest` test utility class have been developed and is provided as part of the UCA for EBC development kit. A JavaDoc documentation is provided for this class. Please refer to the Java Documentation of the `com.hp.uca.expert.testmaterial` package for full explanations.

Using the Apache Ant `build.xml` file provided in the example project (Skeleton) project (or projects created b by the UCA eclipse plugin) allows you to automatically compile tests (using the "test-compile" Ant target), run the tests and generate the test reports (using the "test-run" Ant target).

Following is the list of all Ant targets provided by the `build.xml` file:

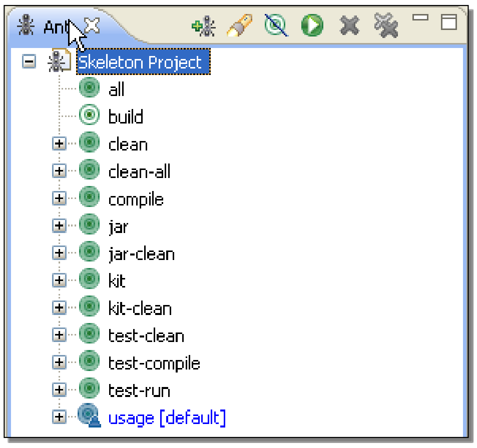
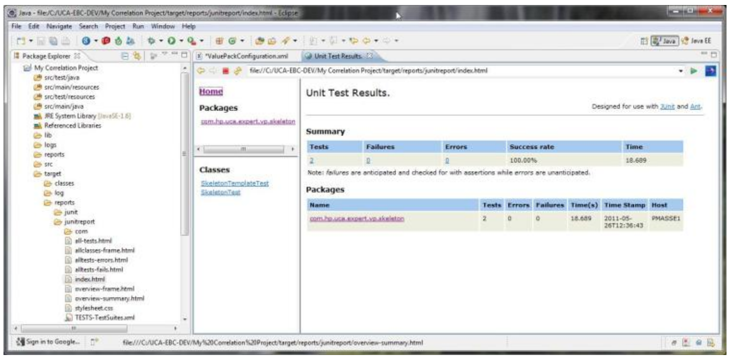


Figure 30 - Ant targets provided by the build.xml file

Note

The build.xml Ant file on runs Test Classes that have a name ended by 'Test' all other classes will not be executed when launching the 'test' target.
It is therefore highly recommended to name all test classes with a name ending with 'Test.java'.

JUnit test reports in HTML format are available in the `target/reports/junitreport` folder of your Value Pack:



Unit Test Results.

Designed for use with JUnit and Ant.

Summary

Tests	Failures	Errors	Success rate	Time
2	0	0	100.00%	18.689

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

Packages

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
com.hp.uca.expert.vp.skeleton	2	0	0	18.689	2011-05-26T12:36:43	PMASSE1

Figure 31 - JUnit tests results for your Value Pack

3.11 How to inject events

The Alarm Collector is the UCA for EBC internal component responsible for collecting events from outside UCA for EBC in order to feed them to the scenarios of the Value Packs deployed on UCA for EBC.

The Alarm Collector is implemented as a JMS Topic that is registered using JNDI so that other applications can get access to it to post events that will feed UCA for EBC Value Packs.

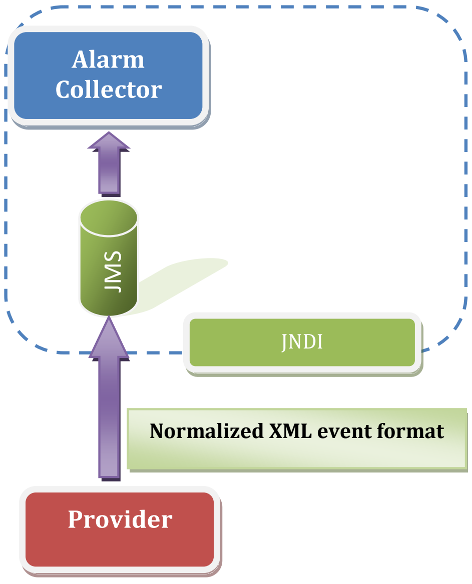


Figure 32 - UCA for EBC alarm collection

3.11.1 Normalized input

The UCA for EBC Alarm Collector defines a normalized alarm XML format based on the X.733 standard alarm format. Only alarms that comply with this format will be processed.

3.11.1.1 Sample alarms file

Here is a sample XML file that contains alarms in the X.733 alarm format:

```
<?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>BOX B1</originatingManagedEntity>
          <alarmType>COMMUNICATIONS_ALARM</alarmType>
          <probableCause>Fire</probableCause>
          <perceivedSeverity>MINOR</perceivedSeverity>
          <alarmRaisedTime>2009-09-16T12:00:00.000+02:00</alarmRaisedTime>
  </AlarmCreationInterface>
  <AlarmCreationInterface>
          <sourceIdentifier>src</sourceIdentifier>
          <identifier>2</identifier>
          <originatingManagedEntity>BOX B1</originatingManagedEntity>
          <alarmType>COMMUNICATIONS_ALARM</alarmType>
          <probableCause>Fire</probableCause>
          <perceivedSeverity>CLEAR</perceivedSeverity>
          <alarmRaisedTime>2009-09-16T12:00:00.000+02:00</alarmRaisedTime>
  </AlarmCreationInterface>
</Alarms>
```

3.11.2 Command-line injector tool

UCA for EBC provides a tool to send events described in a simple XML File containing X.733 alarms to the UCA for EBC Alarm Collector.

This tool is located in the *${UCA_EBC_HOME}/bin* folder. It is called **uca-ebc-injector**.

This tool will inject alarms contained in an XML file into the input alarm queue (implemented as a JMS Topic) of a local or remote UCA for EBC Server instance.

A sample of such an XML file containing alarms to be fed to UCA for EBC is located in the ${*UCA_EBC_HOME}/alarms* folder.

Note

☞ For more information on the uca-ebc-injector command-line tool, please refer to the [R3] *HP UCA for Event Based Correlation – Administration, Configuration and Troubleshooting Guide*

[R4] *HP UCA for Event Based Correlation – Value Pack Examples*

3.11.3 A sample Java Alarm injector

The following chapters describe how you can create your own sample Java Alarm injector application that can connect to UCA for EBC Alarm Collector JMS Topic to post Alarms to UCA for EBC.

3.11.3.1 Initializing the JNDI initial context

In order to create a sample Java Alarm injector, you must first initialize the JNDI context that will be used to retrieve the JMS Topic of the UCA for EBC Alarm Collector:

```
Context jndiContext = null;
/*
* Create a JNDI API InitialContext object
*/
try {
   jndiContext = new InitialContext();
} catch (NamingException e) {
   System.out.println("Could not create JNDI API context: " +
e.toString());
    System.exit(1);
}
```

Please note that the *jndi.properties* file must be provided in the classpath of your sample Java Alarm injector.

3.11.3.2 Configuring the jndi.properties file

Here is the content of a sample *jndi.properties* file to be used by your sample Java Alarm injector:

```
java.naming.factory.initial =
org.apache.activemq.jndi.ActiveMQInitialContextFactory
topic.uca-ebc-alarms = com.hp.uca.ebc.alarms


# use the following property to configure the default connector
java.naming.provider.url =tcp\://localhost\:61666
```

The **topic.uca-ebc-alarms** property is used to record the name the UCA for EBC Alarm Collector JMS topic: **com.hp.uca.ebc.alarms**

The **java.naming.provider.url** property can be configured to match the hostname and port number of UCA for EBC JNDI service.

3.11.3.3 Looking up the UCA for EBC Alarm Collector JMS topic

Once the JNDI context is initialized, the codes in your sample Java Alarm injector shall first lookup for the JNDI connection factory, and then retrieve the UCA for EBC Alarm Collector JMS topic by looking up its name:

```
ConnectionFactory connectionFactory = null;
Destination destination = null;
/*
* Look up connection factory and destination.
*/
try {
  connectionFactory = (ConnectionFactory) jndiContext
             .lookup("ConnectionFactory");
  destination = (Destination) jndiContext.lookup("uca-ebc-alarms");
} catch (NamingException e) {
  System.out.println("JNDI API lookup failed: " + e);
  System.exit(1) ;
}
```

3.11.3.4 Connect and send the message

With the connectionFactory retrieved, you then need to create the connection, then the session, and finally the producer:

```
Connection connection = null;
MessageProducer producer = null;

try {
  connection = connectionFactory.createConnection();
  session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
  producer = session.createProducer(destination);
  TextMessage message = session.createTextMessage();

  StringBuffer buf = new StringBuffer();
  buf.append("<?xml version=\"1.0\" encoding=\"UTF-8\"
standalone=\"yes\"?>");
  buf.append("<Alarms>");
  buf.append("<AlarmCreationInterface>");
  buf.append("<sourceIdentifier>src</sourceIdentifier>");
  buf.append("<identifier>12301</identifier>");
  buf.
append("<originatingManagedEntityClass>BOX</originatingManagedEntityCla
ss>");
  buf.append("<originatingManagedEntity>BOX
B1</originatingManagedEntity>");
  buf.append("<alarmType>COMMUNICATIONS_ALARM</alarmType>");
  buf.append("<probableCause>Fire</probableCause>");
  buf.append("<perceivedSeverity>MAJOR</perceivedSeverity>");
  buf.append("<alarmRaisedTime>2009-09-
16T12:00:00.000+02:00</alarmRaisedTime>");
  buf.append("</AlarmCreationInterface>");
  buf.append("</Alarms>");
  message.setText(buf.toString());
  System.out.println("Sending message: " + message.getText());
  producer.send(message);
} catch (JMSException e) {
  System.out.println("Exception occurred: " + e);
} finally {
if (connection != null) {
  try {
    connection.close();
  } catch (JMSException e) {
  }
}
```

By now you should have a functioning sample Java Alarm injector.

Chapter 4

Specific use cases

4.1 Using the Flag Object

☞ Please refer to [R2] *HP UCA for Event Based Correlation – Reference Guide*

for more information on how to use the Flag Object.

4.2 Alarm Custom fields

☞ Please refer to [R2] *HP UCA for Event Based Correlation – Reference Guide*

for more information on how to use the CustomFields Object.

4.3 Alarm Raised Time

The UCA for EBC provides a helper to set the alarmRaisedTime field, just use the setTimeInMillisecond() that sets all time related fields.

☞ Please refer to [R2] *HP UCA for Event Based Correlation – Reference Guide*

, Chapter 5.1.1.2 General Attributes of Alarm for more information on how to deal with time fields.

4.4 Scenario specific configuration

The UCA for EBC provides a way to manage complex configuration based on XML file when the Customer Value Pack need a complex specific configuration.

☞ Please refer to [R2] *HP UCA for Event Based Correlation – Reference Guide*

for more information on how to use the Specific Configuration.

4.5 Performing initialization at scenario startup

☞ Please refer to [R2] *HP UCA for Event Based Correlation – Reference Guide*

for more information on how to perform initialization of customer object needed by a Value Pack.

4.6 Configuring the filter tags editor

☞ Please refer to [R2] *HP UCA for Event Based Correlation – Reference Guide*

for more information on how to perform configuration to enable the GUI tags editor feature.

Appendix A

A. Ant `build.xml` targets

The value pack examples provided with UCA for EBC come with an Ant *build.xml* file that can build and package the project as described in this document.

Following is the full list of Apache Ant targets defined in the *build.xml* file that can be executed from the command line using the **ant** tool:

- ### eclipse

  Command:

  ```
  # ant eclipse
  ```

  Creates the .project and .classpath files used by eclipse when importing a project.

- ### clean

  Command:

  ```
  # ant clean
  ```

  Removes all files created during the build from the build directory.

- ### compile

  Command:

  ```
  # ant compile
  ```

  Compiles all Java files of the project.

- ### test

  Command:

  ```
  # ant test
  ```

  Runs the JUnit tests defined in the project.

- ### package

  Command:

  ```
  # ant package
  ```

  Build the final, “ready to deploy” value pack ZIP file.

- ### all

  Command:

  ```
  # ant all
  ```

  Is equivalent to executing the following targets: “clean”, “compile”, “test” and “package”.

Glossary

UCA: Unified Correlation Analyzer

EBC: Event Based Correlation

IDE: Integrated Development Environment

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

Inference engine: Process that uses a Rete algorithm for expert behavior

DRL: Drools Rule file

XML: Extensible Markup Language

XSD: Schema of an XML file, describing its structure

X.733: Standard describing the structure of an Alarm used in telecommunication environment.

EVP: UCA for EBC Value Pack