



# Talend ESB Infrastructure Services

## Configuration Guide

### 6.4.1

Adapted for v6.4.1. Supersedes previous releases.

Publication date: June 29, 2017

Copyright © 2017 Talend Inc. All rights reserved.

## Copyright

This documentation is provided under the terms of the Creative Commons Public License (CCPL). For more information about what you can and cannot do with this documentation in accordance with the CCPL, please read: <http://creativecommons.org/licenses/by-nc-sa/2.0/>

This document may include documentation produced at The Apache Software Foundation which is licensed under The Apache License 2.0.

## Notices

Talend and Talend ESB are trademarks of Talend, Inc.

Apache CXF, CXF, Apache Karaf, Karaf, Apache Cellar, Cellar, Apache Camel, Camel, Apache Maven, Maven, Apache Archiva, Archiva, Apache Syncope, Syncope, Apache ActiveMQ, ActiveMQ, Apache Log4j, Log4j, Apache Felix, Felix, Apache ServiceMix, ServiceMix, Apache Ant, Ant, Apache Derby, Derby, Apache Tomcat, Tomcat, Apache ZooKeeper, ZooKeeper, Apache Jackrabbit, Jackrabbit, Apache Santuario, Santuario, Apache DS, DS, Apache Avro, Avro, Apache Abdera, Abdera, Apache Chemistry, Chemistry, Apache CouchDB, CouchDB, Apache Kafka, Kafka, Apache Lucene, Lucene, Apache MINA, MINA, Apache Velocity, Velocity, Apache FOP, FOP, Apache HBase, HBase, Apache Hadoop, Hadoop, Apache Shiro, Shiro, Apache Axiom, Axiom, Apache Neethi, Neethi, Apache WSS4J, WSS4J are trademarks of The Apache Foundation. Eclipse Equinox is a trademark of the Eclipse Foundation, Inc. SoapUI is a trademark of SmartBear Software. Hyperic is a trademark of VMware, Inc. Nagios is a trademark of Nagios Enterprises, LLC.

All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

This product includes software developed at AOP Alliance (Java/J2EE AOP standards), ASM, AntLR, Apache ActiveMQ, Apache Ant, Apache Avro, Apache Axiom, Apache Axis, Apache Axis 2, Apache Batik, Apache CXF, Apache Camel, Apache Chemistry, Apache Common Http Client, Apache Common Http Core, Apache Commons, Apache Commons Bcel, Apache Commons JXPath, Apache Commons Lang, Apache Derby Database Engine and Embedded JDBC Driver, Apache Geronimo, Apache Hadoop, Apache Hive, Apache HttpClient, Apache HttpComponents Client, Apache JAMES, Apache Log4j, Apache Lucene Core, Apache Neethi, Apache POI, Apache Pig, Apache Qpid-Jms, Apache Tomcat, Apache Velocity, Apache WSS4J, Apache WebServices Common Utilities, Apache Xml-RPC, Apache Zookeeper, Box Java SDK (V2), CSV Tools, DataStax Java Driver for Apache Cassandra, Ehcache, Ezmorph, Ganymed SSH-2 for Java, Google APIs Client Library for Java, Google Gson, Groovy, Guava: Google Core Libraries for Java, H2 Embedded Database and JDBC Driver, HsqlDB, Ini4j, JClouds, JLine, JSON, JSR 305: Annotations for Software Defect Detection in Java, JUnit, Jackson Java JSON-processor, Java API for RESTful Services, Jaxb, Jaxen, Jettison, Jetty, Joda-Time, Json Simple, MetaStuff, Mondrian, OpenSAML, Paracel JDBC Driver, PostgreSQL JDBC Driver, Resty: A simple HTTP REST client for Java, Rocoto, SL4J: Simple Logging Facade for Java, SQLite JDBC Driver, Simple API for CSS, SshJ, StAX API, StAXON - JSON via StAX, Talend Camel Dependencies (Talend), The Castor Project, The Legion of the Bouncy Castle, W3C, Woden, Woodstox : High-performance XML processor, XML Pull Parser (XPP), Xalan-J, Xerces2, XmlBeans, XmlSchema Core, Xmlsec - Apache Santuario, Zip4J, atinject, dropbox-sdk-java: Java library for the Dropbox Core API, google-guice. Licensed under their respective license.

---

# Table of Contents

<b>Chapter 1. Introduction .....</b>	<b>1</b>
1.1. Prerequisites to using Talend ESB products .....	3
<b>Chapter 2. Service Locator .....</b>	<b>5</b>
2.1. Installing Service Locator .....	6
2.1.1. Standard installation .....	6
2.1.2. Alternative installation .....	22
2.1.3. (Auto-)Unregister non ESB Provider via an endpoint time-to-live mechanism .....	25
2.1.4. Additional Metadata .....	26
2.1.5. Service Locator endpoint selection strategy configuration .....	27
<b>Chapter 3. Service Activity Monitoring .....</b>	<b>29</b>
3.1. Technical overview .....	30
3.1.1. Architecture diagram .....	30
3.2. Installing the Service Activity Monitoring .....	31
3.2.1. DataSource installation .....	32
3.2.2. Standard installation .....	35
3.2.3. Alternative installation .....	43
3.3. Typical use cases .....	47
3.3.1. Building examples .....	48
3.3.2. Pre-requisites .....	48
3.3.3. General Test .....	48
3.3.4. Filters and Handlers Test .....	49
3.3.5. Monitoring events from database .....	50
<b>Chapter 4. Event Logging .....</b>	<b>51</b>
4.1. Overview .....	52
4.2. Installing the Event Logging in the Talend Runtime container .....	53
4.2.1. Default start/stop .....	53
4.2.2. Manual start/stop .....	54
4.3. Event Logging - Listeners .....	55
4.3.1. Log Listener .....	56
4.3.2. OSGi Event Listener .....	60
4.3.3. SAM Listener .....	62
4.3.4. Locator Listener .....	64
4.4. Event Logging - Agent .....	64
4.4.1. Agent - Receiver part .....	65
4.4.2. Agent - Processing Part .....	67
4.5. Event Logging - Sender .....	69
4.5.1. REST Sender .....	69
4.5.2. JMS Sender .....	70
4.6. Event Logging - Server .....	71
4.6.1. Collector .....	71
4.6.2. Server Pre-processing part .....	72
4.6.3. Server persistence and post-processing .....	73
4.7. Event Logging - Service .....	76
4.8. Talend Log Server .....	76
4.8.1. Configuring the EventLogging server to connect to a secured Elasticsearch .....	77
4.9. Logging page in Talend Administration Center .....	78
4.10. Robust event processing .....	78
4.11. Event Logging - API's and Data Structures .....	80
4.11.1. Event Data - Structure .....	80
4.11.2. Event Database Structure .....	82
4.11.3. Event Logging Service API .....	83
<b>Chapter 5. Service Registry .....</b>	<b>89</b>
5.1. Introduction .....	90
5.2. Installing the Service Registry .....	93
5.2.1. Activating the Service Registry .....	94
5.3. ESB Policies .....	95
5.3.1. WS-Security policies .....	96
5.3.2. Custom policies .....	97
5.3.3. Order of policy execution .....	106
5.4. Typical use cases .....	106
5.4.1. Using the Service Registry with Talend ESB .....	107
5.4.2. Using the Service Registry with REST Services .....	108
5.4.3. Referencing WS-Policy resources within Service Registry .....	118
5.4.4. Storing and retrieving Metadata within Service Registry .....	119
<b>Chapter 6. Talend Identity and Access Management .....</b>	<b>127</b>
6.1. Accessing Talend Identity and Access Management .....	128
6.2. Managing user authentication .....	129
6.2.1. Creating a new user .....	129

6.3. Managing user authorization .....	131
6.3.1. Creating a new group .....	131
6.3.2. Assigning a user to a group .....	132
<b>Chapter 7. Authorization with Talend ESB .....</b>	<b>133</b>
7.1. TESB Client and Endpoint .....	134
7.2. XACML Standard .....	135
7.2.1. Role-Based Access Control .....	137
7.2.2. XACML policies .....	137
7.3. TESB Authorization XACML PolicyDecisionPoint .....	140
7.3.1. Policy Retrieval Point .....	140
7.3.2. Policy Information Point (PIP) .....	141
7.3.3. Deployment/Configuration .....	141
7.3.4. Using a custom PDP implementation .....	142
7.4. TESB Authorization XACML Policy Registry .....	143
7.4.1. Deployment/Configuration .....	143
7.4.2. Atom REST interface .....	144
7.5. Starting and stopping the Authorization service in the Talend Runtime container .....	145
7.6. XACML Request creation .....	145
7.7. XACML Response validation .....	147
7.8. TESB service provider PEP .....	148
7.8.1. Enabling and configuring the TESB PEP .....	148
7.9. TESB client REST STS Interceptor .....	149
<b>Chapter 8. XKMS Service .....</b>	<b>151</b>
8.1. Overview .....	152
8.1.1. Encryption functional architecture .....	153
8.1.2. Signature functional architecture .....	154
8.2. Configuring the XKMS Service .....	154
8.3. Generating key pairs for Signing and Encryption with ESB .....	155
8.4. Configuring encryption for multiple service providers on the same container .....	156
<b>Chapter 9. Using STS with the Talend Runtime .....</b>	<b>159</b>
9.1. Deploying the STS into the Talend Runtime container .....	160
9.2. Deploying the STS into a Servlet Container (Tomcat) .....	160
9.3. Security Token Service Configuration .....	161
9.4. Setting up the security management system in Security Token Service .....	162
9.5. Setting up logging parameters in Security Token Service .....	163
9.6. Data Service Configuration for using STS .....	163
9.7. Creating keys for the Security Token Service .....	164
9.7.1. Using OpenSSL to create certificates .....	164
9.7.2. Deploying and Using a Security Token Service (STS) .....	165
<b>Chapter 10. Provisioning Service .....</b>	<b>167</b>
10.1. Introduction .....	168
10.1.1. Architecture diagram .....	169
10.2. Installing and running the Provisioning Service .....	169
10.3. Stopping the the Provisioning Service .....	170
10.4. Starting the Provisioning Web agent .....	170
10.5. Managing profiles via Provisioning commands .....	171
10.5.1. Provisioning Service Commands .....	171
10.5.2. Provisioning Agent Commands .....	174
10.6. Use cases .....	174
10.6.1. Application profile use case .....	175
10.6.2. System profile use case .....	182
<b>Chapter 11. ActiveMQ .....</b>	<b>187</b>
11.1. Overview .....	188
11.1.1. Download and install .....	188
11.2. Standalone ActiveMQ broker .....	188
11.2.1. Configuration .....	188
11.3. ActiveMQ OSGi bundles .....	189
11.4. ActiveMQ broker inside a Talend Runtime container .....	190
11.4.1. Broker creation .....	190
11.4.2. Broker querying .....	190
11.5. ActiveMQ Web Console .....	190
11.5.1. Configuring ActiveMQ Web Console .....	191
11.5.2. Install the Web Console to a container .....	191
11.5.3. Additional configuration for authentication .....	191
11.6. Examples .....	191
<b>Chapter 12. Artifact Repository .....</b>	<b>193</b>
12.1. Nexus Artifact Repository .....	194
12.1.1. Downloading and installing Nexus .....	194
12.1.2. Deploying in Nexus repository .....	194
12.2. Archiva Artifact Repository (deprecated) .....	195
12.2.1. Downloading and installing Archiva .....	196

12.2.2. Browsing repositories .....	196
12.2.3. Configuring Maven to use an Archiva repository .....	197
12.2.4. Deploying to a Repository .....	199
<b>Chapter 13. Auxiliary Storage .....</b>	<b>201</b>
13.1. Implementation details and configuration .....	202
<b>Appendix A. Backend configuration .....</b>	<b>203</b>
A.1. Configuring database-based storage in Apache Jackrabbit .....	204
A.2. Changing the default Apache Derby DB to Postgres .....	204
A.3. Configuring the Apache Jackrabbit storage to use Oracle .....	206





# Chapter 1. Introduction

This guide covers the installation and configuration information for Talend ESB Infrastructure Services. The services covered by this guide are the following:

For both community and subscription products,

- Service Locator, that provides automatic and transparent failover and load balancing between service Consumers and Providers.
- Service Activity Monitoring, that facilitates the capture of analysis of service activity.
- Security Token Service, that supports Security Assertion Markup Language 2.0 (SAML 2.0) to federate security credentials. For additional information on the Security Token Service, see the *Talend ESB STS User Guide*.
- ActiveMQ, that provides a number of different messaging options to transport events between distributed applications, guaranteeing that they reach their intended recipients.

For subscription products only,

- Event Logging, that collects events across distributed containers and enables to index them and search through them via a Web User Interface. It also supports custom processing, aggregation, signing, and so on.
- Service Registry, a repository for storing service WSDL and WS-Policy files.
- Talend Identity Management, that handles digital identities in enterprise environments - mandatory to use authentication and authorization with the subscription versions of Talend ESB.
- XACML, that specifies access control via XACML policies and registry, on which Authorization is based.
- Authorization, the complete Talend ESB Authorization solution based on Identity Management, XACML, STS.
- XML Key Management Specification (XKMS), an XML-based protocol that is used for the distribution and registration of public keys. Talend ESB uses it for encryption and signing of messages.
- Artifact Repository, that stores and provides the deployment of artifacts for the Talend Runtime container.

---

Some of the services described in this guide can be installed either as standalone or as OSGi bundles in a Talend Runtime container.

For more detailed information about the installation of Talend ESB and the Infrastructure Services, see the *Talend Installation Guide*.

Almost all these services can be started at once in the Talend Runtime container via the `tesb:start-all` command if they have been installed as OSGi bundles into the container, except the Event Logging feature that can be started with a special command.

For more detailed information on how to configure, start and stop the different Infrastructure Services, see the *Talend ESB Container Administration Guide*.



## 1.1. Prerequisites to using Talend ESB products



*There is a number of software and hardware prerequisites you should be aware of, prior to starting the installation of Talend ESB products.*

*For a complete list of installation requirements, compatible software and software versions, see the Talend Installation Guide.*



The term `<TalendRuntimePath>` is used for the directory where the Talend Runtime is installed. This is typically the full path of either `Runtime_ESBSE` or `Talend-ESB-V`, depending on the version of the software that is being used. Please substitute appropriately.

For instance, the Talend Runtime examples are in the `<TalendRuntimePath>/examples/talend` directory.





## Chapter 2. Service Locator

This chapter describes the steps to install and run the Service Locator. The Service Locator is a technical service that provides service consumers with a mechanism to discover service endpoints at runtime, thus isolating consumers from the knowledge about the physical location of the endpoint. Additionally, it allows service providers to automatically register and unregister their service endpoints. In this way, the providers actively advertise the availability of their service endpoints to consumers.

The Service Locator consists of two parts:

- The Service Locator server hosting an endpoint repository.
- The CXF feature used to enable usage of the locator for CXF service consumers and providers.

Like any standard CXF feature, it has separate functionalities for service and consumer:

- when the provider becomes available or unavailable, a provider-side Locator Feature extension registers and deregisters service endpoints respectively in the endpoint repository.
- when a service call to a provider is about to be made, a consumer-side Locator Feature extension transparently retrieves service endpoint addresses from the endpoint repository.

It is also possible to restrict access to the Service Locator (for example, to restrict updates permissions), please see [Restricting access to the Service Locator](#) for more details.

Since creating a distributed, fault-tolerant endpoint repository is a non-trivial task, the Service Locator server implementation is based on proven open source technology - Apache ZooKeeper. This is a highly reliable service that provides coordination between distributed processes. To learn more about Apache ZooKeeper, see <http://zookeeper.apache.org/>.

## 2.1. Installing Service Locator

The Service Locator ships with Talend ESB; its standard installation is embedded in a Talend Runtime container, so it can be started as an OSGi feature, but it is also provided as a standalone application in the `<TalendRuntimePath>/zookeeper` directory.

For detailed information on how to start the Service Locator as OSGi bundle, see the *Talend ESB Container Administration Guide*.

However, the following sections describes how to install and run the Service Locator both as OSGi bundle and as standalone.



Please note that only one Service Locator (ZooKeeper) instance can run on a machine at a time.

### 2.1.1. Standard installation

By default, the Service Locator is embedded as an OSGi feature in the Talend ESB container, so to install it you just need to configure and start it into the Talend ESB container.

#### 2.1.1.1. Configuring Service Locator in the Talend Runtime container

On the Talend Runtime container, to configure the properties of the locator feature, edit this file:

```
<TalendRuntimePath>/container/etc/org.talend.esb.locator.cfg
```

Here is a description for each of the fields:

Property name	Description
<code>locator.endpoints</code>	Specifies the endpoints of all Service Locator instances available to clients. A Service Locator client will arbitrarily pick one of these endpoints to connect to the Service Locator until a connection is established. If the property is not set, the default localhost endpoint of <code>localhost:2181</code> will be used.
<code>endpoint.http.prefix</code>	Necessary when running in a container where the stated endpoints are relative to the container. The default value is an empty string, but typically it will be preset to a value such as <code>:http://localhost:8040/services</code> in the configuration file.
<code>endpoint.https.prefix</code>	Necessary when running in a container where the endpoint is only relative to the container and secured. The default value is an empty string, but typically it will be preset to a value such as <code>:https://localhost:9001/services</code> in the configuration file.
<code>locator.strategy</code>	The endpoint selection strategy to use, as defined in the previous section. Acceptable values are <code>defaultSelectionStrategy</code> , <code>randomSelectionStrategy</code> and <code>evenDistributionSelectionStrategy</code> .
<code>locator.reloadAdressesCount</code>	This parameter is relevant only for <code>evenDistributionSelectionStrategy</code> and <code>randomSelectionStrategy</code> . These strategies cache the list of endpoints returned by the locator for a fixed number of service calls set by this parameter. After this number of calls, the list of available addresses will be refreshed. Set this parameter to a high value to reduce the number of locator refreshes if your services are proving reliable (for example, few failovers occurring).
<code>connection.timeout</code>	Specifies the time (ms) the Service Locator client waits for a connection to get established. Must be greater than zero, with a default of 5000 ms.
<code>session.timeout</code>	Specifies the timeout period in ms of the session established with the server. Sessions are kept alive by requests sent by the client. If a session becomes idle for a period approaching this timeout value, the client will send a ping request to keep the session alive. Must be greater than zero and less than 60000ms (1 minute), by default 5000 ms.

Property name	Description
authentication.name and authentication.password	Authentication properties for the Service Locator Client. Uncomment them to enable the Service Locator client to communicate with a secured locator server.

Here is an example of a `org.talend.esb.locator.cfg` file:

```
locator.endpoints=localhost:2181
endpoint.http.prefix=http://localhost:8040/services
endpoint.https.prefix=https://localhost:9001/services
locator.strategy=defaultSelectionStrategy
locator.reloadAdressesCount=10
connection.timeout=5000
session.timeout=5000
#authentication.name=tesb
#authentication.password=tesb
```

## Service Locator configuration with multiple machines

You may need to update some of these values if the containers are not all on the same machine. This section describes an example scenario, where two containers are accessing the Service Locator, which may be in a third container.

- If the containers are running on different machines, then replace "localhost" with the actual IP address.
  - You may also need to check the endpoint prefixes that are to be published within the locator.
1. Examine the properties in the file `etc/org.talend.esb.locator.cfg` in each container which uses the Service Locator.
  2. The `locator.endpoints` property is set to where the Service Locator is running - this is the normal preset value:

```
locator.endpoints=localhost:2181
```

If the services share the same Service Locator, this needs to be the same in each configuration file. Replace "localhost" with the IP address of where the locator is running, for example, if the IP of where Service Locator is running is 192.168.0.5:

```
locator.endpoints=192.168.0.5:2181
```

3. The endpoint prefixes may also need to be updated - the default configuration uses localhost (as described in the properties table):

```
endpoint.http.prefix=http://localhost:8040/services
endpoint.https.prefix=https://localhost:9001/services
```

- If the IP of a container is 192.168.0.10:

```
endpoint.http.prefix=http://localhost:8040/services    should    be    replaced    with:
endpoint.http.prefix=http://192.168.0.10:8040/services.
```

- If the IP of a second container is 192.168.0.20:

```
endpoint.http.prefix=http://localhost:8040/services    should    be    replaced    with:
endpoint.http.prefix=http://192.168.0.20:8040/services.
```

- If a second container is running on the same host as the first container:

```
endpoint.http.prefix=http://localhost:8041/services    should    be    replaced    with:
endpoint.http.prefix=http://192.168.0.10:8041/services.
```

This above provides just an example; you may need to update your own deployment differently, depending on its configuration.

## 2.1.1.2. Installing and running Service Locator in the Talend Runtime container

Once the Service Locator feature configured to work in your Talend Runtime container(s), you can install and run it.

1. Start the container:

- `container/bin/trun.bat` for Windows,
- `container/bin/trun` for Linux.

New files appear in the `container/etc` directory.

2. If you need to configure the Service Locator server, edit the configuration file named: `org.talend.esb.locator.server.cfg`.

The parameters available are the following:

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=${karaf.base}/zookeeper/data
clientPort=2181
maxClientCnxns = 0
authentication = false
```

Here is a description for each of the fields:

Field name	Description
<code>tickTime</code>	The basic time unit in milliseconds used by the Service Locator. It is used to do heartbeats, and the minimum session timeout will be twice the <code>tickTime</code> .
<code>initLimit</code>	The number of ticks that the initial synchronization phase can take.
<code>syncLimit</code>	The number of ticks that can pass between sending a request and getting an acknowledgment.
<code>dataDir</code>	The location to store the in-memory database snapshots and, unless specified otherwise, the transaction log of updates to the database.
<code>clientPort</code>	The port to listen for client connections.
<code>maxClientCnxns</code>	Number of client connection. Default is 10. 0 is unlimited.
<code>authentication</code>	To enable the authentication in the Service Locator server.

3. Now that you have configured the Service Locator, you can start it in the Talend Runtime container. To do so, execute the following command at the console prompt:

```
tesb:start-locator
```

4. To check that the service has been successfully started and is active, execute the console command:

```
list | grep Locator
```

You should see an output similar to this:

```
ID      State      Blueprint  Spring  Level  Name
[ 211] [Active ] [Created ] [      ] [ 80]  Talend ESB :: Locator :: Server (6.4.1)
```

To ensure that the feature is installed successfully, you can try examples that use the Service Locator server.

To uninstall and stop the Service Locator, execute the console command:

```
tesb:stop-locator
```

### 2.1.1.3. Managing endpoints via Locator commands

The Talend ESB Service Locator feature provides commands that can help you manage endpoints directly from the console.

To access this functionality:

1. Start the container and make sure the Service Locator feature is started.
2. On the container console, execute the following command to install the locator commands:

```
tesb:start-locator-commands
```

3. Once it is successfully installed, you can use the following commands on the console:

```
tlocator:list
```

```
tlocator:register
```

```
tlocator:remove
```

```
tlocator:unregister
```

Enterprise and Platform users can also manage the endpoints via the Service Registry and Service Locator interfaces in Talend Administration Center. For more information, see the *Talend Administration Center User Guide*.

Those commands can also be useful, for Enterprise and Platform users, to clean the Locator registry of undeployed services, especially if they are using the **ESB Locator Endpoints** dashboard in the **Logging** page of Talend Administration Center. For more information, see the *Talend Administration Center User Guide*.

#### tlocator:list

This command lists all the Endpoints registered in the Service Locator. You can use it via the following command:

```
tlocator:list [options] [filter]
```

Where:

- options can be:

Option	Description
-O, --offline-services	Prints only services with no active endpoint.
-ns, --namespace	Prints service name including namespace.
-o, --offline-endpoints	Prints only services with at least one offline endpoint.
-t, --transport	Prints transport protocol for endpoints.
-ep, --properties, --prop	Prints optional endpoint properties.
--help	Displays this help message.
-v, --verbose	Displays a verbose output. It prints all service and endpoint attributes.
-p, --protocol	Prints message protocol for endpoints.

Option	Description
-d, --date	Prints date information for endpoints: online/offline since.

- `filter` corresponds to the Servicename. It is true if any part of the service name matches this filter. This filter is case sensitive.

## tlocator:register

This command registers an endpoint to the Service Locator. You can use it via the following command:

```
tlocator:register [options] serviceName URL
```

Where:

- options can be:

Option	Description
--help	Displays this help message.
-p, --persistent	With this option, the endpoint will be registered as always online. No heartbeat will be required.

- `serviceName` corresponds to the service name for endpoint to be added. It must be fully qualified if adding a new / unknown service name. For adding an endpoint to a known service name, the local part of service name is sufficient.
- `URL` corresponds to the endpoint address to be registered to the Service Locator.

Example of command:

```
tlocator:register -p
    "{http://my.company.com/my-service-namespace}MyServiceName"
    http://my.server.com:8040/ services/MyServiceName tlocator:register
MyServiceName
    http://another.server.com:8040/services/MyServiceName
```

## tlocator:remove

This command removes the endpoint from the Service Locator. You can use it via the following command:

```
tlocator:remove [options] serviceName URL
```

Where:

- The only option available is `--help`, that displays this help message.
- `serviceName` corresponds to the service name of endpoint to be removed. If the Service name is unique in the Service Locator, it is sufficient to only type the local part of the service name. Command completion is available.
- `URL` corresponds to the endpoint address to be removed from the Service Locator. This endpoint will not be tracked / listed any longer.

Example of command:

```
tlocator:remove
    "{http://my.company.com/my-service-namespace}MyServiceName"
    http://my.server.com:8040/servi ces/MyServiceName tlocator:remove
MyServiceName
    http://another.server.com:8040/services/MyServiceName
```



## tlocator:unregister

This command notifies the Service Locator that a service endpoint is offline. You can use it via the following command:

```
tlocator:unregister [options] serviceName URL
```

Where:

- The only option available is `--help`, that displays this help message.
- `serviceName` corresponds to the service name for endpoint to be updated. If the Service name is unique in the Service Locator, it is sufficient to only type local part of service name.
- `URL` corresponds to the endpoint address to be unregistered from the Service Locator. This endpoint will be marked as offline.

### 2.1.1.4. Accessing Service Locator operations via SOAP

The Service Locator SOAP Service component provides a way to access Service Locator operations via a SOAP interface, to:

- *Register an endpoint,*
- *Unregister an endpoint,*
- *Look up all endpoints for a given service,*
- *Lookup one endpoint for a given service.*

To access the Service Locator instance operations via SOAP, you will need to extend the Service Locator by installing an additional proxy service component called the Service Locator SOAP service in the Talend Runtime container. To do so, follow these steps:

1. Type **feature:install tesb-locator-soap-service** in the Talend Runtime container to enable the Service Locator service component.
2. Type **feature:install tesb-zookeeper-server** in the Talend Runtime container to enable the Service Locator server (ZooKeeper server) component.
3. Type **list** in the Talend Runtime container. You should see an output similar to:

ID	State	Blueprint	Spring	Level	Name
[ 189]	[Active ]	[	]	[ 60]	Locator Service :: Common (6.4.1)
[ 190]	[Active ]	[	]	[ 60]	Locator Service :: SOAP Service (6.4.1)
[ 191]	[Active ]	[	]	[ 60]	ZooKeeper server control bundle (1.2)

This output shows that the Service Locator service component and Service Locator server (ZooKeeper server) are enabled in the Talend Runtime container.

Also the ZooKeeper server can be configured in the Talend Runtime container by editing the `container/etc/org.talend.esb.locator.server.cfg` configuration file:

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgment
syncLimit=5
```

```
# the directory where the snapshot is stored.
dataDir=${karaf.base}/zookeeper/data
# the port at which the clients will connect
clientPort=2181
#Number of client connection (default = 10; unlimited = 0)
maxClientCnxns = 0
# Enable authentication in Locator Server
authentication = false
```

This configuration is the same as the Service Locator configuration, described in [Alternative installation](#).

To check that the service is working, access its WSDL at: `http://localhost:8040/services/ServiceLocatorService?wsdl`.

The WSDL file for the Service Locator SOAP Service can be found at: `add-ons/locator/LocatorService.wsdl`

The corresponding schema files with definitions of the types are:

- `add-ons/locator/locator-common-types.xsd`
- `add-ons/locator/locator-soap-types.xsd`

## Register an endpoint

For a specific service, register an endpoint on the Service Locator server, so the user can access this endpoint through the service locator server.

Parameters: fully qualified service name, endpoint URL, user defined properties (optional).

Return: void

The **Register an endpoint** operation is described in `LocatorService.wsdl` as follows:

```
<operation name="registerEndpoint">
  <input message="lps:registerEndpointInput"/>
  <output message="lps:registerEndpointOutput"/>
  <fault name="InterruptedExceptionFault"
    message="lps:InterruptedExceptionFault"/>
  <fault name="ServiceLocatorFault" message="lps:ServiceLocatorFault"/>
</operation>
```

```
<message name="registerEndpointInput">
  <part name="parameters" element="lpx:registerEndpoint"/>
</message>
<message name="registerEndpointOutput">
  <part name="parameters" element="lpx:registerEndpointResponse"/>
</message>
```

The related message type definition is separately described in `locator-soap-types.xsd` and `locator-common-types.xsd` as follows:

```
<xsd:element name="registerEndpoint">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="serviceName" type="xsd:QName"/>
      <xsd:element name="endpointURL" type="xsd:anyURI"/>
      <xsd:element name="binding" type="lpx:BindingType"/>
      <xsd:element name="transport" type="lpx:TransportType"/>
      <xsd:element name="properties" type="lpx:SLPropertiesType"
        minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="registerEndpointResponse">
<xsd:complexType>
  <xsd:sequence/>
</xsd:complexType>
</xsd:element>
```

```
<xsd:simpleType name="BindingType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="SOAP11" />
    <xsd:enumeration value="SOAP12" />
    <xsd:enumeration value="JAXRS" />
    <xsd:enumeration value="OTHER" />
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="TransportType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="HTTP" />
    <xsd:enumeration value="HTTPS" />
    <xsd:enumeration value="JMS" />
    <xsd:enumeration value="OTHER" />
  </xsd:restriction>
</xsd:simpleType>
```

An example of registering an endpoint for a specific service is provided in the project `/examples/talend/tesb/locator-service/soap-service/war/`:

An example of simple locator service configuration is in `/examples/talend/tesb/locator-service/soap-service/war/src/main/resources/client.xml`:

```
<jaxws:client id="locatorService"
  address="http://localhost:8040/services/ServiceLocatorService"
  serviceClass="org.talend.services.esb.locator.v1.LocatorService"
</jaxws:client>
```

An example of how to register an endpoint using this configuration is in `/examples/talend/tesb/locator-service/soap-service/war/src/main/java/demo/service/ContextListener.java`:

```
ClassPathXmlApplicationContext context =
  new ClassPathXmlApplicationContext (" /client.xml");
LocatorService client =
  (LocatorService) context.getBean("locatorService");
String serviceHost = "localhost:8040";

try {
  client.registerEndpoint(new QName(
    "http://talend.org/esb/examples/", "GreeterService"),
    serviceHost, BindingType.SOAP_11, TransportType.HTTP, null);
} catch (InterruptedException e) {
  e.printStackTrace();
} catch (ServiceLocatorFault e) {
  e.printStackTrace();
}
```

## Unregister an endpoint

Unregister an endpoint, which has been registered on the Service Locator server, from the Service Locator server. After unregistering the endpoint, it can not be accessed by the Service Locator server.

Parameters: fully qualified service name, endpoint URL.

Return: success or non-success (endpoint did not exist)

The **Unregister an endpoint** operation is described in `LocatorService.wsdl` as follows:

```
<operation name="unregisterEndpoint">
  <input message="lps:unregisterEndpointInput" />
  <output message="lps:unregisterEndpointOutput" />
  <fault name="InterruptedExceptionFault"
    message="lps:InterruptedExceptionFault" />
  <fault name="ServiceLocatorFault" message="lps:ServiceLocatorFault" />
</operation>
```

```
<message name="unregisterEndpointRequest">
  <part element="lpx:unregisterEndpointRequest" name="input" />
</message>
<message name="unregisterEndpointInput">
  <part name="parameters" element="lpx:unregisterEndpoint" />
</message>
<message name="unregisterEndpointOutput">
  <part name="parameters" element="lpx:unregisterEndpointResponse" />
</message>
```

The related message type definition is separately described in `locator-soap-types.xsd` and `locator-common-types.xsd` as follows:

```
<xsd:element name="unregisterEndpoint">
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="serviceName" type="xsd:QName" />
    <xsd:element name="endpointURL" type="xsd:anyURI" />
  </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

```
<xsd:element name="unregisterEndpointResponse">
<xsd:complexType>
  <xsd:sequence/>
</xsd:complexType>
</xsd:element>
```

Example of unregistering an endpoint for a specific service provided in project `/examples/talend/tesb/locator-service/soap-service/war/`:

```
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("/
client.xml");
LocatorService client = (LocatorService) context.getBean("locatorService");

String serviceHost = this.context.getInitParameter("serviceHost");

...

client.unregisterEndpoint(new QName("http://talend.org/esb/examples/",
  "GreeterService"), serviceHost);
```

## Look up all endpoints for a given service

Lookup all endpoints for a specific service presently registered on the Service Locator server.

Parameters: fully qualified service name, required user defined properties (optional).

Return: list of WS-Addressing EPR's, for all endpoints that provide the service and fulfill the required properties. If none exists return a business fault.

The **Lookup all endpoints for given Service** operation is described in `LocatorService.wsdl` as follows:

```
<operation name="lookupEndpoints">
  <input message="lps:lookupEndpointsInput" />
  <output message="lps:lookupEndpointsOutput" />
</operation>
```

```

<fault name="InterruptedExceptionFault"
  message="lps:InterruptedExceptionFault"/>
<fault name="ServiceLocatorFault" message="lps:ServiceLocatorFault"/>
</operation>

```

```

<message name="lookupEndpointsInput">
  <part name="parameters" element="lpx:lookupEndpoints"/>
</message>
<message name="lookupEndpointsOutput">
  <part name="parameters" element="lpx:LookupEndpointsResponse"/>
</message>

```

The related message type definition is separately described in `locator-soap-types.xsd` and `locator-common-types.xsd` as follows:

```

<xsd:complexType name="lookupRequestType">
  <xsd:sequence>
    <xsd:element name="serviceName" type="xsd:QName"/>
    <xsd:element name="matcherData" type="lpx:MatcherDataType"
      minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="LookupEndpointsResponse">
<xsd:complexType>
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="return"
      nillable="false" type="wsa:EndpointReferenceType"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>

```

## Lookup one endpoint for a given service

Lookup only one endpoint for the given service which has been registered on the Service Locator server.

Parameters: fully qualified service name, required user defined properties (optional).

Return: one WS-Addressing EPR, for an endpoint that provides the service and fulfills the required properties. If several endpoints match, select one randomly. If none exists, return business fault.

The **Lookup one endpoint for given Service** operation is described in `LocatorService.wsdl` as follows:

```

<operation name="lookupEndpoint">
  <input message="lps:lookupEndpointInput"/>
  <output message="lps:lookupEndpointOutput"/>
  <fault name="InterruptedExceptionFault"
    message="lps:InterruptedExceptionFault"/>
  <fault name="ServiceLocatorFault" message="lps:ServiceLocatorFault"/>
</operation>

```

```

<message name="lookupEndpointInput">
  <part name="parameters" element="lpx:lookupEndpoint"/>
</message>
<message name="lookupEndpointOutput">
  <part name="parameters" element="lpx:lookupEndpointResponse"/>
</message>

```

The related message type definition is separately described in `locator-soap-types.xsd` and `locator-common-types.xsd` as follows:

```

<xsd:element name="lookupEndpoint" type="lpx:lookupRequestType"/>
<xsd:element name="lookupEndpointResponse">
<xsd:complexType>
  <xsd:sequence>

```

```
<xsd:element name="value" type="wsa:EndpointReferenceType"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Example of Lookup endpoint for the given service provided in project `/examples/talend/tesb/locator-service/soap-service/client/`:

Example of simple locator service configuration you can see in `/examples/talend/tesb/locator-service/soap-service/client/src/main/filtered-resources/META-INF/client.xml`:

```
<jaxws:client id="locatorService"
  address="http://localhost:8040/services/ServiceLocatorService"
  serviceClass="org.talend.services.esb.locator.v1.LocatorService"
</jaxws:client>
```

Example how to lookup endpoint using this configuration you can see in `/examples/talend/tesb/locator-service/soap-service/client/src/main/java/demo/client/Client.java`:

```
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("/META-INF/
client.xml");
LocatorService client = (LocatorService) context.getBean("locatorService");

W3CEndpointReference endpointReference = client.lookupEndpoint(new QName("http://
talend.org/esb/examples/", "GreeterService"), null);
System.out.println(endpointReference.toString());

javax.xml.ws.Service jaxwsServiceObject = Service.create(new QName("http://talend.org/
esb/examples/", "GreeterService"));

Greeter greeterProxy = jaxwsServiceObject.getPort(endpointReference, Greeter.class);
String reply = greeterProxy.greetMe("HI");
System.out.println("Server said: " + reply);
```

## 2.1.1.5. Accessing Service Locator operations via REST

The Service Locator REST Service component provides a way to access the Service Locator operations using REST calls, to:

- [Register an endpoint](#),
- [Unregister an endpoint](#),
- [Lookup all endpoints](#),
- [Lookup one endpoint](#).

To access the Service Locator instance operations via REST, the Service Locator will need to be extended by installing an additional proxy service component in the Talend Runtime container. To do so, follow the below steps:

1. Type **feature:install tesb-locator-rest-service** in the Talend Runtime container to enable the REST Service Locator component.
2. Type **feature:install tesb-zookeeper-server** in the Talend Runtime container to enable the Service Locator server (zookeeper server) component.
3. Type **list** in the Talend Runtime container. You should see the output:

ID	State	Blueprint	Spring	Level	Name
[ 190]	[Active ]	[	[	[ 60]	Locator Service :: Common (6.4.1)
[ 191]	[Active ]	[	[	[ 60]	Locator Service :: REST Service (6.4.1)

```
[ 192] [Active ] [      ] [      ] [ 60] ZooKeeper server control bundle (1.2)
```

The above output shows that the Service Locator REST Service component and Service Locator server (ZooKeeper server) are enabled in the Talend Runtime container.

The Service Locator server (Zookeeper server) configuration is the same as described in [Accessing Service Locator operations via SOAP](#).

To check that the service is working, access its WADL in a browser at: `http://localhost:8040/services/ServiceLocatorRestService?_wadl&_type=xml`

The WADL file for the Service Locator REST Service can be found at: `add-ons/locator/LocatorService.wadl`

The corresponding schema files with definitions of types are:

- `add-ons/locator/locator-common-types.xsd`
- `add-ons/locator/locator-rest-types.xsd`
- `add-ons/locator/ws-addr.xsd`



If you have Talend ESB, there is GUI functionality provided by the Talend Administration Center, for viewing the Service Locator information. Please see the *Talend Installation Guide* and *Talend Administration Center User Guide* for more details.

## Register an endpoint

Register an endpoint for a specific service.

Parameters: fully qualified service name, endpoint URL, user defined properties (optional).

Return: void.

The **Register an endpoint** for a specific service operation is described in `LocatorService.wadl` as follows:

```
<resource path="endpoint">
  <method name="POST" id="registerEndpoint">
    <request>
      <representation mediaType="application/xml"
        element="ns:RegisterEndpointRequest"/>
      <representation mediaType="application/json"
        element="ns:RegisterEndpointRequest" />
    </request>
  </method>
</resource>
```

Example of request url with POST method: `locator/endpoint/`

```
<?xml version="1.0" encoding="UTF-8"?>
<lpx:RegisterEndpointRequest
  xmlns:lpx="http://talend.org/schemas/esb/locator/rest/2011/11"
  xmlns:tns="http://www.w3.org/2005/08/addressing"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://talend.org/schemas/esb/locator/rest/2011/11 types.xsd">
  <serviceName>
    {http://service.proxy.locator.esb.talend.org}LocatorServiceImpl
  </serviceName>
  <endpointURL>
    http://services.talend.org/TestEndpoint
  </endpointURL>
  <binding>JAXRS</binding>
  <transport>HTTP</transport>
```

```
<EntryType>
  <key>systemTimeout</key>
  <value>200</value>
</EntryType>
</lpx:RegisterEndpointRequest>
```

## Unregister an endpoint

Unregister an endpoint for specific Service from the Service Locator server, which has been registered on the Service Locator server. After unregistering the endpoint, it can not be accessed.

Parameters: fully qualified service name, endpoint URL.

Return: void

The **Unregister an endpoint** operation is described in `LocatorService.wadl` as follows:

```
<resource path="endpoint/{serviceName}/{endpointURL}">
  <method name="DELETE" id="unregisterEndpoint">
    <request>
      <param name="serviceName" type="xsd:string" style="template"
        required="true" />
      <param name="endpointURL" type="xsd:string" style="template"
        required="true" />
    </request>
  </method>
</resource>
```

Example of request url with DELETE method: `locator/endpoint/{namespaceURI}serviceName/endpointURL`. Note that any `/` in the `namespaceURI` has to be replaced with `%2F`.

## Lookup all endpoints

Lookup all endpoints for the given service which has been registered on the Service Locator server.

Parameters: fully qualified service name, required user defined properties (optional).

Return: list of WS-Addressing EPR's, for all endpoints that provide the service and fulfill the required properties. If none exists return `WebApplicationException` and status 404.

The **Lookup all endpoints** for given Service operation is described in `LocatorService.wadl` as follows:

```
<resource path="endpoints/{serviceName}">
  <method name="GET" id="lookupEndpoints">
    <request>
      <param name="serviceName" type="xsd:string" style="template"
        required="true" />
      <param name="param" type="xsd:string" style="matrix"
        repeating="true" />
    </request>
    <response status="200">
      <representation mediaType="application/xml">
        <element="ns:EndpointReferenceList" />
      </representation>
      <representation mediaType="application/json">
        <element="ns:EndpointReferenceList" />
      </representation>
    </response>
  </method>
</resource>
```

Example of request url with GET method: `locator/endpoints/{namespaceURI}localPart?p=key1,value1;p=key2,value2;p=key3,value3`. Note that any `/` in the `namespaceURI` has to be replaced with `%2F`.



%2F. For example, `http://localhost:8040/services/ServiceLocatorRestService/locator/endpoint/{http:%2F%2Ftalend.org%2Fgreeter}GreeterService?location=germany`.

## Lookup one endpoint

Lookup one endpoint for a given service.

Parameters: fully qualified encoded service name, required user defined properties (optional).

Return: one WS-Addressing EPR, for an endpoint that provides the service and fulfills the required properties. If several endpoints match select one randomly. If none exists return `WebApplicationException` and status 404.

The **Lookup one endpoint** for given Service operation is described in `LocatorService.wadl` as follows:

```
<resource path="endpoint/{serviceName}">
  <method name="GET" id="lookupEndpoint">
    <request>
      <param name="serviceName" type="xsd:string" style="template"
        required="true" />
      <param name="param" type="xsd:string" style="matrix"
        repeating="true" />
    </request>
    <response status="200">
      <representation mediaType="application/xml"
        element="wsa:EndpointReference" />
      <representation mediaType="application/json"
        element="wsa:EndpointReference" />
    </response>
  </method>
</resource>
```

Example of request url with GET method: `locator/endpoint/{namespaceURI}localPart?p=key1,value1;p=key2,value2;p=key3,value3`. Note that any / in the namespaceURI has to be replaced with %2F.

### 2.1.1.6. Restricting access to the Service Locator

By default, access to the Service Locator server is not restricted; anyone can add, delete or lookup services.

This access restriction is added by enabling authentication functionality using the Java Authentication and Authorization Service (JAAS) login module in the container.

To do that, you have to set corresponding properties in specific container configuration files, and this section describes this in detail.



The authentication feature is only relevant for Service Locator servers running in the Talend Runtime container, not for the stand-alone version (and not for a pure Apache Zookeeper server).

## Enabling authentication for a Service Locator server



Part of this configuration involves specifying users with corresponding passwords and roles. It depends on type of your JAAS login module where and how this information is specified. For example, if the `JDBCLoginModule` is used then user, passwords and roles are stored in a database.

Please take a look at the **Security framework** section of the *Karaf Developers Guide* (<http://karaf.apache.org/>) to get information on how to configure and use these different JAAS login modules in the container.

The configuration steps needed are as follows:

1. Enable authentication in a server container, by setting the corresponding property in the ZooKeeper server configuration file `<container>/etc/org.talend.esb.locator.server.cfg`:

```
authentication = true
```



*Do not switch off authentication after Service Locator is secured and services have been registered with the Service Locator.*

2. Specify users with corresponding passwords and roles.

By default all information about users is stored in `<container>/etc/users.properties`. So, modify this file in the container where the Service Locator is running, and add roles for the user(s).

For example, add the following lines to `<container>/etc/users.properties`:

```
# tadmin is user with administrator privileges
tadmin=tadmin,admin,sl_admin
# sluser is a user for the client side that is just able to lookup
# endpoints on Service Locator
sluser=upassword,sl_read
# slservice is a user for server side that is able to register and
# lookup endpoints on Service Locator
slservice=spassword,sl_maintain
```

Note that the following roles are available for Service Locator clients:

Role	Description
sl_read	This role is for clients, that only lookup endpoints.  If the <code>sl_read</code> role is given to a user, they can get data from a node and list its children.
sl_maintain	This role is for users that register endpoints on the Service Locator server. The user can: <ul style="list-style-type: none"> <li>• get data from a node and list its children</li> <li>• create a child node</li> <li>• set data for a node</li> <li>• delete a child node</li> </ul>
sl_admin	It is the same as <code>sl_maintain</code> , but in addition, the user can set permissions.



Roles are case insensitive - you can use either uppercase or lowercase letters for roles in configuration files.



*For production use, the sample passwords used here will need to be replaced with your project's own passwords.*

## Enabling authentication for a Service Locator client

To enable authentication for a client, define user names and passwords (corresponding to the ones on the server) by adding authentication properties in the Service Locator configuration file `<container>/etc/org.talend.esb.locator.cfg`.

For example:

- in a container, where a consumer is looking up services from the Service Locator server, add:

```
authentication.name=sluser
authentication.password=upassword
```

- in a container, where a Web Service is adding or deleting services from the Service Locator server, add:

```
authentication.name=slservice
```

```
authentication.password=spassword
```

## Securing the Service Locator SOAP Service

The Service Locator SOAP Service provides additional security configuration.



*The Service Locator REST service can not currently be secured.*



The configuration files described here are created in the container when you install the Service Locator SOAP Service component.

The predefined security configurations support two scenarios: using a UserName token or a SAML token. For switching between these scenarios and configuring additional security parameters use the `etc/org.talend.esb.locator.service.cfg` configuration file:

You can specify the following properties in that file:

Property name	Description
<code>locator.authentication</code>	NO (default) - No security scenario SAML - SAML token scenario TOKEN - UserName token scenario
<code>policy.token</code>	Location of the UserName token scenario policy file.
<code>policy.saml</code>	Location of the SAML token scenario policy file.
<code>security.signature.properties</code>	Link to the properties file which contains signature parameters. Used for SAML token verification. Default value is <code>file:\${tesb.home}/etc/keystores/serviceKeystore.properties</code> .
<code>security.signature.username</code>	SAML token signature username. Used for SAML token verification.
<code>security.signature.password</code>	SAML token signature password. Used for SAML token verification.

The UserName token policy is located and can be configured here: `etc/org.talend.esb.locator.token.policy`.

The SAML token policy is located and can be configured here: `etc/org.talend.esb.locator.saml.policy`.

## Implementing authentication for the Rent-a-Car example

To enable authentication for the Rent-a-Car example, update its configuration files as follows:

1. In the first container (where you run the Locator feature and the Rent-a-Car services), update `<container>/etc/org.talend.esb.locator.cfg` with the user information:

```
authentication.name=slservice
authentication.password=spassword
```

2. Then update `<container>/etc/users.properties` with the role information:

```
sluser=upassword,sl_read
slservice=spassword,sl_maintain
```

3. In the second container (where you run the Rent-a-Car client API), update `<container>/etc/org.talend.esb.locator.cfg` and add:

```
authentication.name=sluser
authentication.password=upassword
```

## Running clients and services in the same container

Note that ideally, when running the Rent-a-Car example, the Service Locator server, every service and every consumer (app-reservation) are in different containers. But this method is still valid if the application or service runs in the same container with Service Locator server.

You just have to keep in mind, that all the consumers or services in the same container use the same locator client with the same credentials (set by the properties `authentication.name` and `authentication.password` in `org.talend.esb.locator.cfg`).

### 2.1.1.7. Logging

The Service Locator server logs messages using log4j. You will see log messages logged at the console (default) and/or in the following log file: `/container/log/tesb.log`.

## 2.1.2. Alternative installation

The Service Locator can also be installed as standalone:

1. Navigate to `<TalendRuntimePath>/zookeeper` or the root of the unpacked Apache Zookeeper package.
2. To start the Service Locator, a configuration file is needed, so create this file. Its default name is `conf/zoo.cfg`, but you can give it a different name.
3. It must contain the following configuration parameters:

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/zookeeper/data
clientPort=2181
maxClientCnxns = 0
```

Here is a description for each of the possible fields:

Field name	Description
<code>tickTime</code>	The basic time unit in milliseconds used by the Service Locator. It is used to do heartbeats, and the minimum session timeout will be twice the <code>tickTime</code> .
<code>initLimit</code>	The number of ticks that the initial synchronization phase can take.
<code>syncLimit</code>	The number of ticks that can pass between sending a request and getting an acknowledgment.
<code>dataDir</code>	The location to store the in-memory database snapshots and, unless specified otherwise, the transaction log of updates to the database.
<code>clientPort</code>	The port to listen for client connections.
<code>maxClientCnxns</code>	Number of client connection. Default is 10. 0 is unlimited.
<code>authentication</code>	To enable the authentication in the Service Locator server.

4. Change the value of `dataDir` to specify an existing, initially empty directory.

5. Now that you have created the configuration file, you can start the Service Locator server. The `<TalendRuntimePath>zookeeper/bin` directory contains scripts that allow easy access (classpath in particular) to the Service Locator server and command-line client:

- **bin/zkServer.sh start [configFilename]** (Linux)
- **bin/zkServer.cmd start [configFilename]** (Windows)

Where `configFilename` needs to be specified if it is not the default `zoo.cfg` configuration file.

These steps run the Service Locator in standalone mode. There is no replication, so if the Service Locator process fails, the service will go down, so you may want to consider using a replicated Service Locator. For more information, see [Running a replicated Service Locator](#).

## 2.1.2.1. Configuring the Service Locator

To customize properties, edit the `locator.properties` in your classpath instead.

The following properties can be specified in the Service Locator configuration file:

Property name	Description
<code>locator.endpoints</code>	Specifies the endpoints of all Service Locator instances available to clients. A Service Locator client will arbitrarily pick one of these endpoints to connect to the Service Locator until a connection is established. If the property is not set, the default localhost endpoint of <code>localhost:2181</code> will be used.
<code>endpoint.http.prefix</code>	Necessary when running in a container where the stated endpoints are relative to the container. The default value is an empty string, but typically it will be preset to a value such as <code>:http://localhost:8040/services</code> in the configuration file.
<code>endpoint.https.prefix</code>	Necessary when running in a container where the endpoint is only relative to the container and secured. The default value is an empty string, but typically it will be preset to a value such as <code>:https://localhost:9001/services</code> in the configuration file.
<code>locator.strategy</code>	The endpoint selection strategy to use, as defined in the previous section. Acceptable values are <code>defaultSelectionStrategy</code> , <code>randomSelectionStrategy</code> and <code>evenDistributionSelectionStrategy</code> .
<code>locator.reloadAdressesCount</code>	This parameter is relevant only for <code>evenDistributionSelectionStrategy</code> and <code>randomSelectionStrategy</code> . These strategies cache the list of endpoints returned by the locator for a fixed number of service calls set by this parameter. After this number of calls, the list of available addresses will be refreshed. Set this parameter to a high value to reduce the number of locator refreshes if your services are proving reliable (for example, few failovers occurring).
<code>connection.timeout</code>	Specifies the time (ms) the Service Locator client waits for a connection to get established. Must be greater than zero, with a default of 5000 ms.
<code>session.timeout</code>	Specifies the timeout period in ms of the session established with the server. Sessions are kept alive by requests sent by the client. If a session becomes idle for a period approaching this timeout value, the client will send a ping request to keep the session alive. Must be greater than zero and less than 60000ms (1 minute), by default 5000 ms.
<code>authentication.name</code> <code>authentication.password</code>	and Authentication properties for the Service Locator Client. Uncomment them to enable the Service Locator client to communicate with a secured locator server.

Here is an example of a `zoo.cfg` file:

```
locator.endpoints=localhost:2181
endpoint.http.prefix=http://localhost:8040/services
endpoint.https.prefix=https://localhost:9001/services
locator.strategy=defaultSelectionStrategy
locator.reloadAdressesCount=10
connection.timeout=5000
session.timeout=5000
#authentication.name=tesb
#authentication.password=tesb
```

## Service Locator configuration with multiple machines

You may need to update some of these values if the containers are not all on the same machine. This section describes an example scenario, where two containers are accessing the Service Locator, which may be in a third container.

- If the containers are running on different machines, then replace "localhost" with the actual IP address.
  - You may also need to check the endpoint prefixes that are to be published within the locator.
1. Examine the properties in the file `locator.properties` in each container which uses the Service Locator.
  2. The `locator.endpoints` property is set to where the Service Locator is running - this is the normal preset value:

```
locator.endpoints=localhost:2181
```

If the services share the same Service Locator, this needs to be the same in each configuration file. Replace "localhost" with the IP address of where the locator is running, for example, if the IP of where Service Locator is running is 192.168.0.5:

```
locator.endpoints=192.168.0.5:2181
```

3. The endpoint prefixes may also need to be updated - the default configuration uses localhost (as described in the properties table of [Configuring the Service Locator](#)):

```
endpoint.http.prefix=http://localhost:8040/services
endpoint.https.prefix=https://localhost:9001/services
```

- If the IP of a container is 192.168.0.10:

```
endpoint.http.prefix=http://localhost:8040/services    should    be    replaced    with:
endpoint.http.prefix=http://192.168.0.10:8040/services.
```

- If the IP of a second container is 192.168.0.20:

```
endpoint.http.prefix=http://localhost:8040/services    should    be    replaced    with:
endpoint.http.prefix=http://192.168.0.20:8040/services.
```

- If a second container is running on the same host as the first container:

```
endpoint.http.prefix=http://localhost:8041/services    should    be    replaced    with
endpoint.http.prefix=http://192.168.0.10:8041/services.
```

This above provides just an example; you may need to update your own deployment differently, depending on its configuration.

### 2.1.2.2. Running a replicated Service Locator

Running the Service Locator server in standalone mode is convenient for evaluation, development, and testing. But in production, you should run the Service Locator in replicated mode. A replicated group of servers in the same application is called a quorum, and in replicated mode, all servers in the quorum have copies of the same configuration file. The configuration is similar to the one used in standalone mode, but with a few differences:

```
tickTime=2000
dataDir=/var/locator
clientPort=2181
maxClientCnxns = 0
initLimit=10
syncLimit=5
```

```
server.1=locator_host1:2888:3888
server.2=locator_host2:2888:3888
server.3=locator_host3:2888:3888
```

- The new configuration entry, `initLimit` corresponds to the number of ticks that the initial synchronization phase can take. For `initLimit` and `syncLimit` timeouts, the unit of time is specified using `tickTime`. In this example, the timeout for `initLimit` is 5 ticks at 2000 milliseconds a tick, or 10 seconds total.
- The configuration entry `syncLimit` corresponds to the number of ticks that can pass between sending a request and getting an acknowledgment.
- The entries of the form `server.x` list the servers that make up the Service Locator service. When the server starts up, it knows which server it is by looking for the file **myid** in the data directory. That file contains the server number in ASCII format.
- Note the two port numbers after each server name: "2888" and "3888". Peers use the former port to connect to other peers. Such a connection is necessary so that peers can communicate, for example, to agree upon the order of updates. More specifically, a Service Locator server uses this port to connect followers to the leader. When a new leader arises, a follower opens a TCP connection to the leader using this port. Because the default leader election also uses TCP, we currently require another port for leader election. This is the second port in the server entry.

### 2.1.2.3. Maintaining a Service Locator

The Service Locator continually saves znode snapshot files and, optionally, transactional logs in a Data Directory to enable you to recover data. It is a good idea to back up the Service Locator data directory periodically. Although the Service Locator is highly reliable due to persistent copies being replicated on each server, recovering from backups may be necessary in cases of catastrophic failure.

The Service Locator server does not remove snapshots and log files, so they will accumulate over time. This directory will need to be cleaned up periodically based on your backup schedules and processes. To help automate a cleanup, a `zkCleanup.sh` script is provided in the `bin` directory. Modify this script as necessary for your situation. In general, you will want to run this as a cron task based on your backup schedule.

The data directory is specified by the `dataDir` parameter in the Service Locator server [configuration file](#), and the data log directory is specified by the `dataLogDir` parameter. For more information, see [Ongoing Data Directory Cleanup](#).

### 2.1.2.4. Logging

The Service Locator server logs messages using log4j. You will see log messages logged at the console (default) and/or a log file depending on the log4j configuration.

## 2.1.3. (Auto-)Unregister non ESB Provider via an endpoint time-to-live mechanism

It is possible to set a time-to-live parameter for all endpoints, via the REST and SOAP Service Locator services, so make sure they are both started to be able to use the time-to-live feature.

Time to live means how long this endpoint should be considered active since the moment at which the time to live of the endpoint is set up. When this time is over, then the endpoint is considered inactive and may be automatically unregistered.

This feature can be used by non-ESB Providers (.NET Provider, non ESB Java Provider, for example), in case they can not unregister themselves correctly, due to a non graceful shutdown, a platform that might not allow this, a broken network, and so on. If this is the case, then the Provider should call this REST/SOAP operation to just update the endpoint's time-to-live in regular time intervals (which the non-ESB Provider has to implement on its own). This update is done via an `updateTimetolive` method, to make sure the non-ESB Provider endpoint is "online" until the specified time to live is over, everytime it is registered. This will lead to the following behaviours:

- When the `updateTimetolive` method is used on an "online" endpoint, it remains "online" but becomes "expirable", which means it will automatically become "offline" when the specified time to live is over.
- Invoking `updateTimetolive` on an endpoint which is "online" and already has a time-to-live, will reset the time-to-live to the new value.
- Invoking `updateTimetolive` on "offline" endpoint will bring the endpoint "online" and set the specified time-to-live for it.
- Re-registering an endpoint (which means invoking the registering of an endpoint which is already registered) which has time-to-live, will keep it "online" but will erase the time-to-live flag (which means it will make the endpoint non-expirable). But, calling the `updateTimetolive` method after that will make the endpoint expirable again.

Then, Service Locator internally checks that endpoints whose time-to-live is over are being unregistered automatically. This check is only enabled when the REST or SOAP Services are started for the Service Locator.

The `AutoUnregister` feature can be configured in the Talend Runtime container configuration file `org.talend.esb.locator.cfg`:

```
locator.endpoints.timetolive.check=true
locator.endpoints.timetolive.interval=300
```

To disable the feature, the property `locator.endpoints.timetolive.check` should be set to *false*. In this case, there will have no check for expired endpoints, even when the REST or SOAP services are started.

To define the interval between two checks, set the value of the property `locator.endpoints.timetolive.interval` in seconds.

Example of SOAP request that sets time-to-live for the `CRMSERVICE` endpoint to be five minutes from now:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://talend.org/schemas/esb/locator/2011/11">
  <soapenv:Header/>
  <soapenv:Body>
    <ns:updateTimetolive> <ns:serviceName xmlns:ns4="http://services.talend.org/
CRMSERVICE">ns4:CRMSERVICEProvider</ns:serviceName>
    <ns:endpointURL>http://localhost:8040/services/CRMSERVICEProvider</
ns:endpointURL>
    <ns:timetolive>300</ns:timetolive>
    </ns:updateTimetolive>
  </soapenv:Body>
</soapenv:Envelope>
```

Example of PUT request to the Service Locator REST service:

```
/services/ServiceLocatorRestService/locator/endpoint/{serviceName}/{endpointURL}/meta?
timetolive={timetolive}
```

## 2.1.4. Additional Metadata

Sometimes a finer grained control of endpoints for a specific service a client gets when retrieving the endpoints is needed. For this purpose you can define additional metadata for an endpoint, such as the country for which the



endpoint is only valid or the bandwidth it provides. The client on the other side may define the metadata it requires from the endpoint from which a service call is to be made.

### Example 2.1. Service Locator enabled endpoint with additional metadata

```
<jaxws:endpoint xmlns:tns="http://talend.org/esb/examples/" id="greeter"
  implementor="demo.service.GreeterImpl"
  serviceName="tns:GreeterService" address="/GreeterService">
  <jaxws:features>
    <bean class="org.talend.esb.servicelocator.cxf.LocatorFeature">
      <property name="availableEndpointProperties">
        <map>
          <entry key="country" value="Luxembourg, Belgium"/>
          <entry key="bandwidth" value="Class A"/>
        </map>
      </property>
    </bean>
  </jaxws:features>
</jaxws:endpoint>
```

In the example above, the endpoint provides a metadata entry for `country` with the values `Luxembourg` and `Belgium` and an entry for `bandwidth` with value `Class A`.

### Example 2.2. Service Locator enabled client with additional metadata requirements

```
<jaxws:client id="GreeterClient" serviceClass="demo.common.Greeter"
  address="locator://">
  <jaxws:features>
    <bean class="org.talend.esb.servicelocator.cxf.LocatorFeature">
      <property name="requiredEndpointProperties">
        <map>
          <entry key="country" value="Belgium"/>
        </map>
      </property>
    </bean>
  </jaxws:features>
</jaxws:client>
```

In the example above, the client requires the endpoint to have a metadata entry for `country` that at least includes `Belgium` as value.

## 2.1.5. Service Locator endpoint selection strategy configuration

Currently three endpoint selection strategies are supported: `defaultSelectionStrategy`, `randomSelectionStrategy` and `evenDistributionSelectionStrategy`.

- `defaultSelectionStrategy` uses the same endpoint as long as there is no failover with no distribution between endpoints.
- `evenDistributionSelectionStrategy` uses a client-side round robin strategy. For example, if there are three instances (endpoints), round robin uses sequential distribution: "1 2 3 1 2 3 1 2 3". If multiple clients use this strategy, it could happen that all clients choose subsequently the same endpoints since the locator instances for each client operate independently.

In case of failover (for example if the second instance goes down), when the Service Locator client again executes a request for endpoints, it will just get the remaining endpoints (here, the first and third). One endpoint will be picked arbitrarily with the sequential distribution on remaining nodes resuming after that.

- `randomSelectionStrategy` selects randomly from the available endpoints for each call. This strategy reduces the chances of clients choosing the same endpoints.

In summary, in case of failover, a random alternative endpoint is selected to start with, and then the selected strategy resumes as normal.

The selection strategy at a container level is configured in the properties file as described in [Configuring Service Locator in the Talend Runtime container](#) and in [Configuring the Service Locator](#) by setting the `locator.strategy` property. If not configured, the `defaultSelectionStrategy` will be used.

The endpoint selection strategy can also be configured for each consumer by adding an additional property in the consumer configuration. For the consumer selection strategy setting, add the `selectionStrategy` property in the `beans.xml` file as shown below:

```
<jaxws:features>
  <bean class="org.talend.esb.servicelocator.cxf.LocatorFeature">
    <property name="selectionStrategy"
              value="randomSelectionStrategy" />
  </bean>
</jaxws:features>
```



## Chapter 3. Service Activity Monitoring

The Service Activity Monitoring component allows for logging and monitoring service calls made with the Apache CXF framework. Typical use cases are collecting usage statistics and fault monitoring.

The Service Activity Monitoring consists of two parts:

- Agents (sam-agent) which gather and send monitoring data.
- A server (sam-server) which processes and stores the data.

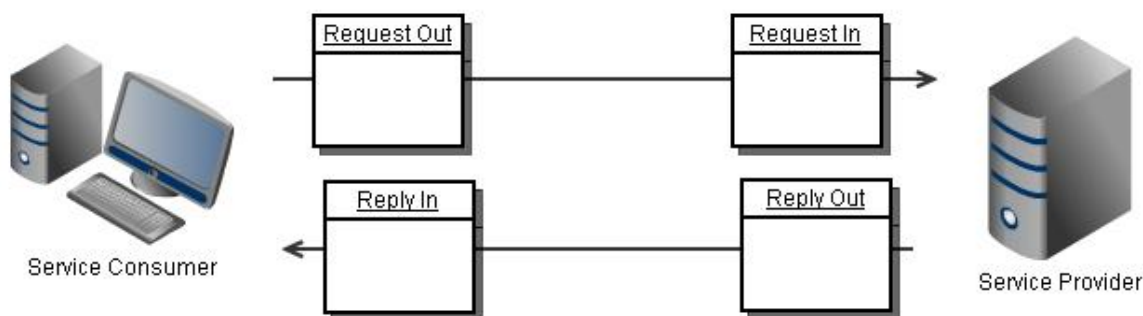
The sequence of how these are used is as follows:

1. The Agent creates events out of requests and replies from both the service consumer and provider side.
2. The events are first collected locally and then sent to the Service Activity Monitoring Server periodically (so as not to disturb normal message flow).
3. When the server receives events from the Agent, it optionally uses filters and/or handlers on those events and stores them in a database.

The Service Activity Monitoring Agent and Server are made available as follows:

- The Service Activity Monitoring Server is available in the Talend Runtime (via the command **tesb:start-sam**).
- Alternatively, the Service Activity Monitoring Server can be deployed as a WAR in a servlet container with database access information configured.
- The Agent is automatically enabled for Data Services deployed on Talend Runtime with the **Use Service Activity Monitor** option selected in the Talend Studio.
- The Agent is also available as a JAR that needs to be on the classpath of the service consumer and provider.

## 3.1. Technical overview



One service call can generate four events: for example, a consumer is sending a request (REQ\_OUT), the service receives the request (REQ\_IN), the service sends a response (RESP\_OUT) and the consumer receives the response (RESP\_IN).

An Agent can be configured to collect all four events of this single service call, from both the consumer and provider side. For further event processing, all of these events will get the same `flow id`. For more detailed information, see [Architecture diagram](#).

Consumer side	Provider side
REQ_OUT	REQ_IN
RESP_IN	RESP_OUT
FAULT_IN	FAULT_OUT

Besides normal Event types, additional Lifecycle Events are also generated by SAM agent.

In the Talend Runtime container, when the agent bundle is started or stopped, the `SERVER_START/SERVER_STOP` events will be generated. For Service or Data Service bundles, when they have been started/stopped, the `SERVICE_START/SERVICE_STOP` (for Provider) or `CLIENT_CREATE/CLIENT_DESTROY` (for Consumer) events will be generated.

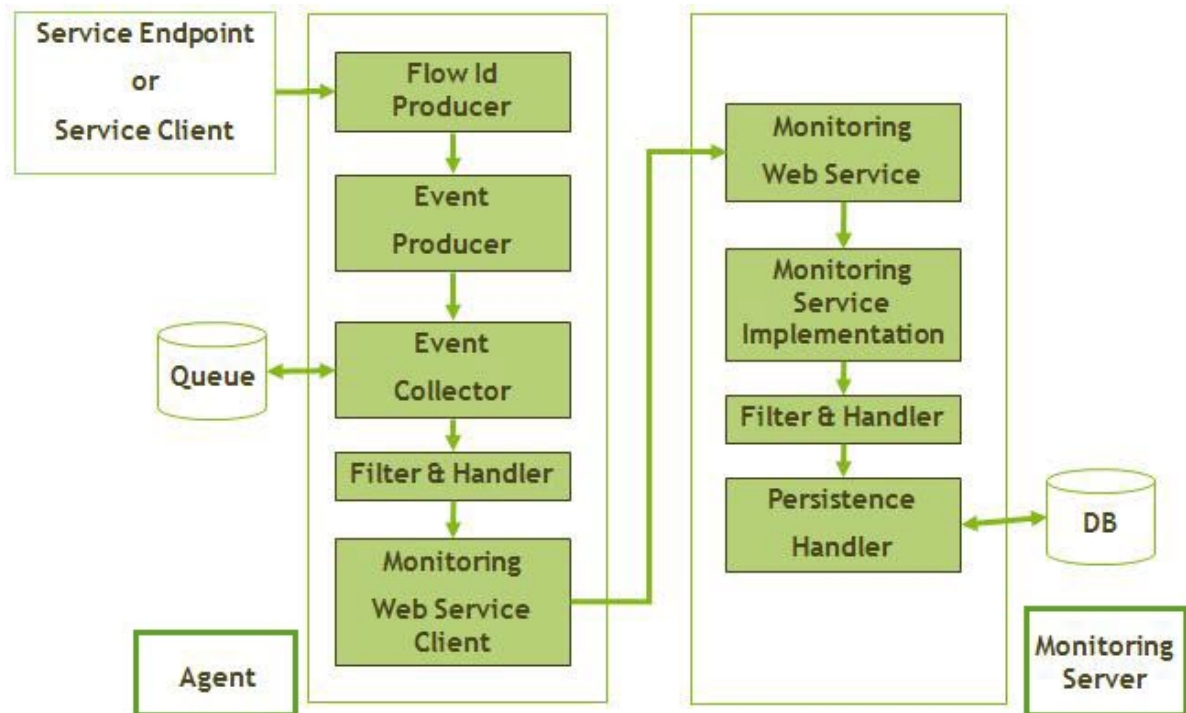


The value of the `collector.lifecycleEvent` property must be set to true in order to generate and store the lifecycle events.

Lifecycle	Event type
Talend Runtime container	SERVER_START/SERVER_STOP
Service Provider/Consumer	SERVICE_START/SERVICE_STOP; CLIENT_CREATE/CLIENT_DESTROY
Data Service	SERVICE_START/SERVICE_STOP; CLIENT_CREATE/CLIENT_DESTROY

### 3.1.1. Architecture diagram

On the left of the below diagram the Agent is described, on the right the Service Activity Monitoring Server. The Agent is used to collect all message data from both the service and client, and sends this data to the Service Activity Monitoring Server. This Server will receive events and store them into the database. A Web service is used as the interface between the Agent and the Server.



The FlowId Producer is a component used to generate the FlowId (a UUID) for the Message Header and pass it to subsequent messages. For each message exchange, the flow id is created if there is no flow id present. So, for the first client, the flow id is created for each service call. When you have an intermediary, this receives a service call, but also calls other services; then the flow id is carried from the incoming call to all calls that follow this call. Then, on the server side, the flow id is taken from the request and also set on the response.

Filters or handlers can be set up on both the Agent side and the Service Activity Monitoring Server side, and can subsequently be used to filter events and manipulate the event's content. There are some built-in filters and handlers (for example: `StringContentFilter`, `PasswordHandler`) and you can develop your own filters and handlers by extending the `EventFilter` or `EventHandler` Service Provider Interface (SPI).

For the structure of information on events, please see [EVENTS Structure](#).

## 3.2. Installing the Service Activity Monitoring

The Service Activity Monitoring installation includes Agent side installation and Server side installation. But before this, you need to install the datasource that will store the Service Activity Monitoring data.

Service Activity Monitoring can be installed in several different ways: into the Talend Runtime container, the standard installation, or into a servlet container like Apache Tomcat or Jetty, for example.

Examples (`sam-example-client`, `sam-example-service`, `sam-example-service2` and `sam-example- osgi`) are available to demonstrate how to install a Service Activity Monitoring Agent into a Servlet container or an OSGi Container.



Multiple instances of the Service Activity Monitoring Server can be running at the same time.

To use the same Service Activity Monitoring Server from multiple containers, update the `service.url` property in the `<TalendRuntimePath>/container/etc/org.talend.esb.sam.agent.cfg` file in each container. (For more information, see [Configuring the Service Activity Monitoring Agent](#) and [Configuring the Service Activity Monitoring Agent in a Servlet container](#)).

## 3.2.1. DataSource installation

The installation of DataSource is a prerequisite to the installation of Service Activity Monitoring, as it is used to store Events data. The supported DataSources are the following: Apache Derby, MySQL, Oracle, SQL Server, IBM DB2, PostgreSQL and H2 Database Engine. For more information about the installation of JDBC drivers and the installation of the datasources into the Talend Runtime container, see the *Talend ESB Container Administration Guide*.

There are several out-of-box DataSource features which can be installed into a Talend Runtime container or when using J2EE/Tomcat: MySQL, H2, Oracle, DB2, SQLServer and PostgreSQL. This section also contains the instructions to install the JNDI DataSource: [Configuring the Service Activity Monitoring Server in a Servlet container](#) and [Configuring the Service Activity Monitoring Server](#) in the Talend Runtime container.

### 3.2.1.1. Installing the DataSource into J2EE/Tomcat

Information on how to configure a DataSource in the J2EE/Tomcat container can be found in the corresponding J2EE/Tomcat documentation. For example, to configure a H2 DataSource in Tomcat:

1. Download the H2 driver jar (h2-1.3.165.jar) and put it into CATALINA\_HOME/lib directory.
2. Add a Resource entry for the H2 DataSource to the CATALINA\_HOME/conf/context.xml:

```
<Resource name="jdbc/datasource" auth="Container"
    type="javax.sql.DataSource" username="sa" password=""
    driverClassName="org.h2.Driver"
    url="jdbc:h2:tcp://localhost/~:/test"
    maxActive="8" maxIdle="30" maxWait="10000"/>
```

The JNDI DataSource name jdbc/datasource is available to be used in the Service Activity Monitoring Server.

Here are Resource entries for other databases:

- Derby:

```
<Resource name="jdbc/datasource" auth="Container"
    type="javax.sql.DataSource" username="test" password="test"
    driverClassName="org.apache.derby.jdbc.ClientDriver"
    url="jdbc:derby://localhost:1527/db;create=true"
    maxActive="8" maxIdle="30" maxWait="10000"/>
```

- MySQL:

```
<Resource name="jdbc/datasource" auth="Container"
    type="javax.sql.DataSource" username="test" password="test"
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/test"
    maxActive="8" maxIdle="30" maxWait="10000"/>
```

- DB2:

```
<Resource name="jdbc/datasource" auth="Container"
    type="javax.sql.DataSource" username="db2admin" password="qwaszx"
    driverClassName="com.ibm.db2.jcc.DB2Driver"
    url="jdbc:db2://localhost:50000/TEST"
    maxActive="8" maxIdle="30" maxWait="10000"/>
```

- SQLServer:

```
<Resource name="jdbc/datasource" auth="Container"
```

```

type="javax.sql.DataSource" username="test" password="test"
driverClassName="com.microsoft.sqlserver.jdbc.SQLServerDriver"
url="jdbc:sqlserver://localhost:1029;instanceName=sqlexpress;datab
aseName=Test"
maxActive="8" maxIdle="30" maxWait="10000"/>

```

- Oracle:

```

<Resource name="jdbc/datasource" auth="Container"
type="javax.sql.DataSource" username="xxx" password="xxx"
driverClassName="oracle.jdbc.pool.OracleDataSource"
url="jdbc:oracle:thin:@localhost:1521:XE"
maxActive="8" maxIdle="30" maxWait="10000"/>

```

- PostgreSQL:

```

<Resource name="jdbc/datasource" auth="Container"
type="javax.sql.DataSource" username="postgres" password="qwazsx"
driverClassName="org.postgresql.Driver"
url="jdbc:postgresql://localhost:5432/"
maxActive="8" maxIdle="30" maxWait="10000"/>

```

### 3.2.1.2. Database installation and initialization

This section describes database initialization.

1. Make sure your chosen database is installed properly and is accessible.
2. Log in with a user account with CREATE permissions and run the "init SQL" scripts for the corresponding database (see table below). There are two initial scripts for each database. The script with "\_ind" suffix is used to create indexes in database.

The script files for the corresponding databases are described in the following table. The SQL scripts can be found in the <TalendRuntimePath>/add-ons/sam/db directory.

SQL script filename	Database
create.sql	Apache Derby
create_ind.sql	
create_mysql.sql	MySQL
create_mysql_ind.sql	
create_oracle.sql	Oracle
create_oracle_ind.sql	
create_sqlserver.sql	SQL Server
create_sqlserver_ind.sql	
create_h2.sql	H2 Database Engine
create_h2_ind.sql	
create_db2.sql	IBM DB2
create_db2_ind.sql	
create_postgres.sql	PostgreSQL
create_postgres_ind.sql	

Once the scripts executed, the EVENTS and EVENTS\_CUSTOMINFO tables are created in your database.

## Automatically starting Derby

For the Derby database, it can be started automatically by adding `-Dorg.talend.esb.sam.server.embedded=true` to the environment variable `CATALINA_OPTS` in the Tomcat script.

In case of OSGi container, you can start Derby database by installing the feature: **tesb-derby-starter**.

## SQL server and TCP/IP

By default SQL server does not allow connections via TCP/IP - please consult the relevant documentation on how to enable it.

### 3.2.1.3. EVENTS Structure

This is the information stored in the Service Activity Monitoring Server database on a particular event:

Field	Type	Description
ID	bigint(20)	The persistence id of the Event.
MESSAGE_CONTENT	longtext	The SOAP or REST message content which comes from Service Provider/Service Consumer. <i>Note: It will be null for all Lifecycle Events.</i>
EI_TIMESTAMP	datetime	The time at which the Event is created.
EI_EVENT_TYPE	varchar(255)	EventType is an enumeration. Values: REQ_IN; REQ_OUT; RESP_IN; RESP_OUT; FAULT_IN; FAULT_OUT; SERVER_START; SERVER_STOP; SERVICE_START; SERVICE_STOP; CLIENT_CREATE; CLIENT_DESTROY
ORIG_CUSTOM_ID	varchar(255)	Reserved field. It is not being used, currently.
ORIG_PROCESS_ID	varchar(255)	The process id is the OS process id.
ORIG_HOSTNAME	varchar(128)	The name of the Host on which the SAM agent is running.
ORIG_IP	varchar(64)	The IP address of the Host on which the SAM agent is running.
MI_PORT_TYPE	varchar(255)	The Service port type which enabled the SAM agent. <i>Note: It will be null for SERVER_START/SERVER_STOP Events.</i>
MI_OPERATION_NAME	varchar(255)	The Service operation name which enabled the SAM agent. It can be standard QName operations for SOAP events and GET/POST methods for REST events. <i>Note: It will be null for all Lifecycle Events.</i>
MI_MESSAGE_ID	varchar(255)	<p>The MessageID which is generated/transferred using the CXF Addressing feature, for SOAP events. According to the common definition of the MessageId in the WS-Addressing specifications: REQ_OUT and REQ_In are the same message, so they should have the same MessageId; and, RESP_OUT and RESP_IN are the same message, so they should have the same MessageId too.</p> <p>Note:</p> <ol style="list-style-type: none"> <li>1. The MessageID will be null for all Lifecycle Events.</li> <li>2. If <code>log.enforceMessageIDTransfer=false</code> and does not enable the <code>WSAddressingFeature</code> or <code>Policy</code> with <code>Addressing</code> explicitly, it will also be null.</li> <li>3. It will be null for REQ_IN/RESP_OUT if the WS-Addressing feature is enabled only on the provider side and not enabled on the consumer side.</li> </ol> <p>For REST events, this field is null.</p>



Field	Type	Description
MI_FLOW_ID	varchar(64)	The unique id (UUID) for the message flow. All events with the same id belong together. <i>Note: It will be null for all Lifecycle Events.</i>
MI_TRANSPORT_TYPE	varchar(255)	The transport type of the event: <ul style="list-style-type: none"> <li>• <code>http://schemas.xmlsoap.org/soap/http</code> for SOAP events,</li> <li>• <code>http://cxf.apache.org/transports/http</code> for REST events.</li> </ul> <i>Note: It will be null for all Lifecycle Events.</i>
ORIG_PRINCIPAL	varchar(255)	The principal info in the message header. <i>Note: It will be null for all Lifecycle Events.</i>
CONTENT_CUT	tinyint(1)	Flag, if the event content has been cut from the Agent. <i>Note: It will be null for all Lifecycle Events.</i>

### 3.2.1.4. EVENTS\_CUSTOMINFO Structure

Field	Type	Description
ID	bigint(20)	Stores the unique persistence id of EVENTS_CUSTOMINFO
EVENT_ID	bigint(20)	Stores the relative EVENT's ID value
CUST_KEY	varchar(255)	The custom property key, for example: <ul style="list-style-type: none"> <li>• <i>address</i> for SOAP events,</li> <li>• <i>address, Accept Type, Content Type, Response Code</i> for REST events.</li> </ul>
CUST_VALUE	varchar(255)	The custom property value corresponding to the custom key, for example: <ul style="list-style-type: none"> <li>• for <i>address</i>, the URL of the service,</li> <li>• for <i>Accept Type</i>, acceptable data types,</li> <li>• for <i>Content Type</i>, actual type of the returned response,</li> <li>• for <i>Response Code</i>, the status code of the response.</li> </ul>

## 3.2.2. Standard installation

As the Service Activity Monitoring is divided into two parts: the Agent and the Server, in this section, you will see how to install both of them in to a Talend Runtime container.

### 3.2.2.1. Installing the Service Activity Monitoring Agent in the Talend Runtime container

To install the Agent in Talend Runtime container:

1. By default, the Service Activity Monitoring Agent features are already added to the Talend Runtime container, but if not present, they can be added using:

```
feature:repo-add mvn:org.talend.esb/features/6.4.1/xml
```

- To install and start the Service Activity Monitoring Agent features, execute the following command:

```
feature:install tesb-sam-agent
```

### 3.2.2.2. Installing the Service Activity Monitoring Server into the Talend Runtime container

Before installing the Service Activity Monitoring Server, the DataSource feature for your preferred Database needs to be installed in the Talend Runtime container. To do so:

- Copy the Database Driver to the <TalendRuntimePath>\lib\ext directory.
- Add the Database package details to the <TalendRuntimePath>\etc\custom.properties. For example, in case of MS SQL Server, the Database package details need to be added as shown below:

```
org.osgi.framework.system.packages.extra = \
com.microsoft.sqlserver.jdbc; \
com.sun.jmx.mbeanserver; \
com.sun.management; \
...
```

- Restart the Talend Runtime container.

For convenience the following shell commands are provided in Talend Runtime containers:

To install and start the Service Activity Monitoring Server:

```
tesb:start-sam
```

To uninstall and stop the Service Activity Monitoring Server (and embedded Derby, if used):

```
tesb:stop-sam
```

For use with another OSGi container (like Apache Karaf, for example), install the Service Activity Monitoring Server with these commands:

```
feature:repo-add mvn:org.talend.esb/features/6.4.1/xml
```

```
feature:install tesb-sam-server
```

Now, the Service Activity Monitoring Server will be installed and started. You can check its status with this URL in a browser: <http://localhost:8040/services/MonitoringServiceSOAP?wsdl>

### 3.2.2.3. Configuring the Service Activity Monitoring Agent

The main configuration files for Agents are <Talend.runtime.dir>/container/etc/org.talend.esb.sam.agent.cfg and the filter and handler configuration files. The <Talend.runtime.dir>/container/etc/org.talend.esb.sam.agent.cfg is automatically created when executing the Service Activity Monitoring features in the Talend Runtime container. Filters and handlers are based on Spring bean configuration and can be added to the application's context (for example, beans.xml).

Properties description:

Property	Default	Description
collector.scheduler.interval		Interval (in milliseconds) of Agent built-in scheduler. The Agent will make one or several calls to the Service Activity Monitoring Server sending Events from local queue at this specified value. The number of calls actually made to the Service Activity Monitoring Server when

Property	Default	Description
		scheduler interval has arrived is decided by the number of events in the local queue and the number of collector.maxEventsPerCall. This interval must be greater than 0.
collector.maxEventsPerCall		The value of this parameter is used to restrict the max number of Events per call to the Service Activity Monitoring Server. Lower values help to avoid sending overly large SOAP body messages to the Service Activity Monitoring server.
collector.lifecycleEvent	false	Whether the Agent should collect and send the lifecycle events to the Service Activity Monitoring Server. If true, the Service Activity Monitoring Server must be started before the Talend Runtime container, otherwise connection exceptions will be thrown.
log.messageContent	true	Whether the Agent should store the Producer and Consumer SOAP message content into Events and send them to the Service Activity Monitoring Server.
log.maxContentLength	-1	Sets the maximum SOAP content length per Event. -1 is unlimited.
log.enforceMessageIDTransfer	false	If true, Service Activity Monitoring will add WS-Addressing functionality implicitly and enforce MessageID transfer between Events. If false, the MessageID will be null in the Events if the user does not enable the WSAddressingFeature or Policy with Addressing explicitly.
service.url		The URL of Service Activity Monitoring Server that the Agent is to communicate with.
service.retry.number	5	Number of retries when a call to the Service Activity Monitoring Server fails.
service.retry.delay	1000	Delay in milliseconds before the next retry to call the Service Activity Monitoring Server.

For example:

```
collector.scheduler.interval=500
collector.maxEventsPerCall=10
collector.lifecycleEvent=false

log.messageContent=true
log.maxContentLength=-1
log.enforceMessageIDTransfer=true

service.url=http://localhost:8080/sam-server-war/services/MonitoringServiceSOAP
service.retry.number=3
service.retry.delay=5000
```

To filter or manipulate events, these Filter/Handler spring beans should be added into your Service provider or Service consumer bundle or jar, then these beans will be autowired by Service Activity Monitoring agent.

Some example bean definitions can be found below:

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans http://www.springframework.org/
    schema/beans/spring-beans.xsd">

  <bean id="stringContentFilter"
    class="org.talend.esb.sam.common.filter.impl.StringContentFilter">
    <property name="wordsToFilter">
      <list>
        <value>abc</value>
      </list>
    </property>
  </bean>

  <bean class="org.talend.esb.sam.common.filter.impl.JXPathFilter">
    <constructor-arg value="content='test' and eventType='FAULT_IN' and customInfo/
    key1='value1'"/>
  </bean>
```

```

</bean>

<bean id="passwordHandler"
      class="org.talend.esb.sam.common.handler.impl.PasswordHandler">
  <property name="tagNames">
    <list>
      <value>Password</value>
    </list>
  </property>
</bean>

<bean id="fixedPropertiesHandler"
      class="org.talend.esb.sam.common.handler.impl.CustomInfoHandler">
  <property name="customInfo">
    <map>
      <entry key="Application name" value="Dummy App" />
      <entry key="Stage" value="Dev" />
    </map>
  </property>
</bean>
</beans>

```

For more information about how to use Filter/Handler in WAR applications or OSGi bundles, go to the following folder of the Talend ESB: `examples/talend/tesb/sam`.

### 3.2.2.4. Configuring the Service Activity Monitoring Server

The main configuration files for the Service Activity Monitoring Server are `<Talend.runtime.dir>/container/etc/org.talend.esb.sam.server.cfg` and filter, handler configuration files.

Properties description:

Property	Default	Description
monitoringServiceUrl		The address URL published by the Service Activity Monitoring Server
db.datasource	ds-derby	DataSource name used by the Service Activity Monitoring Server to store/query data. For J2EE or Tomcat, it should be like <code>java:comp/env/&lt;Resource name&gt;</code> , for OSGi container, use similar to <code>ds-&lt;Database&gt;</code> .
db.dialect	derbyDialect	The database used to store/query Event data (with different ID Incrementer, Query, and so on.)

For Derby:

```

monitoringServiceUrl=/MonitoringServiceSOAP
db.datasource=ds-derby
db.dialect=derbyDialect

```

For H2 Database Engine:

```

monitoringServiceUrl=/MonitoringServiceSOAP
db.datasource=ds-h2
db.dialect=h2Dialect

```

For Mysql:

```

monitoringServiceUrl=/MonitoringServiceSOAP
db.datasource=ds-mysql
db.dialect=mysqlDialect

```

For Oracle:

```

monitoringServiceUrl=/MonitoringServiceSOAP

```

```
db.datasource=ds-oracle
db.dialect=oracleDialect
```

For IBM DB2:

```
monitoringServiceUrl=/MonitoringServiceSOAP
db.datasource=ds-db2
db.dialect=DB2Dialect
```

For SQL Server:

```
monitoringServiceUrl=/MonitoringServiceSOAP
db.datasource=ds-sqlserver
db.dialect=sqlServerDialect
```

For filter and handler configuration, please refer to [Configuring the Service Activity Monitoring Agent](#).

### 3.2.2.5. Retrieving Service Activity Monitoring services via REST

Talend Service Activity Monitoring REST service is providing access to the Service Activity Monitoring service when installed as an OSGi feature in the Talend Runtime container, in a REST manner.

This document emphasizes the design considerations of the service in a REST architecture, which acts as the part of Talend Service Activity Monitoring component in the overall Talend ESB architecture.

This service is automatically started when starting the general Service Activity Monitoring service via **tesb:start-sam** (or via the **tesb:start-all** command), but to only start the retrieval service, use this command:

```
tesb:start-sam-retrieval-service
```

## Resources and URI templates

This section describes the Service Activity Monitoring REST service resources and URI design.

### Check Alive

<b>URI</b>	/
<b>Possible representations</b>	text
<b>Description</b>	Resource for check is SAM retrieval API REST service online

### Flows Aggregated

<b>URI</b>	/list
<b>Possible representations</b>	JSON
<b>Description</b>	List of aggregated flows in Service Activity Monitoring component

### Flow Details

<b>URI</b>	/flow/{flowID}
<b>Possible representations</b>	JSON
<b>Description</b>	A single flow which includes events with same flowIDs

### Event Details

<b>URI</b>	/event/{eventID}
<b>Possible representations</b>	JSON
<b>Description</b>	A single event representation

## Data representation

This section describes the data representation of requests and responses for the Service Activity Monitoring REST service.

### Check Alive

```
Talend Service Activity Monitoring Server :: REST API - http://127.0.0.1:8040/
services/sam/list
```

### Flows Aggregated

```
{
  "count": 26,
  "aggregated": [
    {
      "flowID": "urn:uuid:21760804-4961-40d3-9adf-1d47dfa52e07",
      "timestamp": 1365497059651,
      "elapsed": 106,
      "transport": "http://schemas.xmlsoap.org/soap/http",
      "port": "{http://www.talend.org/service/}DemoService",
      "operation": "{http://www.talend.org/service/}DemoServiceOperation",
      "types": [
        "REQ_OUT",
        "RESP_OUT",
        "REQ_IN",
        "RESP_IN"
      ],
      "consumerIP": "192.168.144.120",
      "consumerHost": "alexoid",
      "providerIP": "192.168.144.120",
      "providerHost": "alexoid",
      "details": "http://127.0.0.1:8040/services/sam/flow/    \\
        urn:uuid:21760804-4961-40d3-9adf-1d47dfa52e07"
    },
    ....
    {
      "flowID": "urn:uuid:262c180f-467c-4844-aed9-0b023f09a06d",
      "timestamp": 1365431671820,
      "elapsed": 159,
      "transport": "http://cxf.apache.org/transports/http",
      "port": "{http://127.0.0.1:8090/services/customers}WebClient",
      "operation": "GET[/1]",
      "types": [
        "REQ_OUT",
        "RESP_OUT",
        "REQ_IN",
        "RESP_IN"
      ],
      "consumerIP": "192.168.144.120",
      "consumerHost": "alexoid",
      "providerIP": "192.168.144.120",
      "providerHost": "alexoid",
      "details": "http://127.0.0.1:8040/services/sam/flow/    \\
        urn:uuid:262c180f-467c-4844-aed9-0b023f09a06d"
    }
  ]
}
```

### Flow Details

```
{
  "events": [
    {
      "id": 488,
```

```

    "flowID": "urn:uuid:21760804-4961-40d3-9adf-1d47dfa52e07",
    "messageID": "urn:uuid:7c666024-15c0-45ee-9184-721095f49921",
    "timestamp": 1365497059545,
    "type": "REQ_OUT",
    "transport": "http://schemas.xmlsoap.org/soap/http",
    "port": "{http://www.talend.org/service/}DemoService",
    "operation": "{http://www.talend.org/service/}DemoServiceOperation",
    "ip": "192.168.144.120",
    "host": "alexoid",
    "process": 5244,
    "contentCut": false,
    "customInfo": [
      {
        "key": "address",
        "value": "http://localhost:8040/services/DemoService"
      },
      {
        "key": "custom property name",
        "value": "consumer"
      }
    ],
    "details": "http://127.0.0.1:8040/services/sam/event/488"
  },
  ...

  {
    "id": 493,
    "flowID": "urn:uuid:21760804-4961-40d3-9adf-1d47dfa52e07",
    "messageID": "urn:uuid:4c2aa9d5-e6c3-4ec6-80a2-ee91ac2c32b6",
    "timestamp": 1365497059608,
    "type": "RESP_OUT",
    "transport": "http://schemas.xmlsoap.org/soap/http",
    "port": "{http://www.talend.org/service/}DemoServicePortType",
    "operation": "{http://www.talend.org/service/}DemoServiceOperation",
    "ip": "192.168.144.120",
    "host": "alexoid",
    "process": 5244,
    "contentCut": false,
    "customInfo": {
      "key": "address",
      "value": "http://localhost:8040/services/DemoService"
    },
    "details": "http://127.0.0.1:8040/services/sam/event/493"
  },
  ...

]
}

```

## Event Details

```

{
  "id": 77,
  "flowID": "urn:uuid:f0538075-e9ae-491e-b886-05fbdf558380",
  "timestamp": 1365430132346,
  "type": "RESP_IN",
  "transport": "http://cxf.apache.org/transport/http",
  "port": "{http://127.0.0.1:8090/services/customers}WebClient",
  "operation": "GET[/1]",
  "ip": "192.168.144.120",
  "host": "alexoid",
  "process": 8388,
  "contentCut": false,
  "content": "{ \"customer\": { \"id\": 1, \"firstName\": \"Richard\",    \\
    \"city\": \"Columbus\", \"lastName\": \"Monroe\" } }"
}

```

## Exception handling and request results

HTTP defines a suite of standard status codes ([http://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](http://en.wikipedia.org/wiki/List_of_HTTP_status_codes)) that specify the result of the processed request. Status codes are organized into ranges and each range has a different meaning. For example, status codes in the 200 range mean "successful", while status codes in the 400 range mean the client issued a bad request.

## WADL

```
<application xmlns="http://wadl.dev.java.net/2009/02"    \\  
    xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <grammars></grammars>  
  <resources base="http://localhost:8040/services/sam">  
    <resource path="/">  
      <method name="GET">  
        <response>  
          <representation mediaType="text/plain"/>  
        </response>  
      </method>  
    <resource path="event/{id}">  
      <param name="id" style="template" type="xs:string"/>  
      <method name="GET">  
        <request></request>  
        <response>  
          <representation mediaType="application/json"/>  
        </response>  
      </method>  
    </resource>  
    <resource path="flow/{id}">  
      <param name="id" style="template" type="xs:string"/>  
      <method name="GET">  
        <request></request>  
        <response>  
          <representation mediaType="application/json"/>  
        </response>  
      </method>  
    </resource>  
    <resource path="list">  
      <method name="GET">  
        <request>  
          <param name="offset" style="query"    \\  
            default="0" type="xs:int"/>  
          <param name="limit" style="query"    \\  
            default="10" type="xs:int"/>  
        </request>  
        <response>  
          <representation mediaType="application/json"/>  
        </response>  
      </method>  
    </resource>  
  </resources>  
</application>
```

## Service interface

```
@Path("/")  
public interface SAMRestService {  
  
    @GET
```



```

@Path("")
@Produces({ "text/plain" })
Response checkAlive();

@GET
@Path("list")
@Produces({ "application/json" })
Response getFlows(@QueryParam("offset") @DefaultValue("0") Integer offs
et,
                @QueryParam("limit") @DefaultValue("10") Integer limit);

@GET
@Path("flow/{id}")
@Produces({ "application/json" })
Response getFlow(@PathParam("id") String id);

@GET
@Path("event/{id}")
@Produces({ "application/json" })
Response getEvent(@PathParam("id") String id);
}

```

## Talend Service Activity Monitoring REST service security scenarios

Two approaches for service security are used in Talend ESB Service Activity Monitoring REST service: Basic and SAML.

- Basic scenario is a security scenario which is based on Username and Password credentials which is adding as HTTP header to the request:

```
Authorization: Basic a2FyYWY6a2FyYWY=
```

The verification on the service endpoint side uses a JAAS filter to check and verify the provided credentials.

- SAML scenario is a security scenario which is based on a SAML token requested from the Security Token Service service which is adding to request, as an HTTP header:

```
Authorization: SAML a2FyYWY6a2FyYWY6a2FyYWY6a2FyYWY6a2FyYWY=
```

The verification on the service endpoint side uses the validation of SAML token.

## 3.2.3. Alternative installation

The Service Activity Monitoring features can also be installed into a Servlet container.

### 3.2.3.1. Installing the Service Activity Monitoring Agent into a Servlet container

Installing an Agent into a Servlet container (for example, Apache Tomcat or Jetty):

1. The Agent needs to be deployed with the customer's application. The best way to install the agent is to add it to the classpath using a Maven dependency:

```
<dependency>
```

```
<groupId>org.talend.esb</groupId>
<artifactId>sam-agent</artifactId>
<version>{talend esb version}</version>
</dependency>
```

2. With Spring, the Agent has to be added to the Spring context:

```
<import resource="classpath:META-INF/tesb/agent-context.xml" />
```

3. Then, add the Agent as a `jaxws:features` to the endpoint/client for Spring-related services, for example:

```
<jaxws:endpoint
  id="customerService" address="/CustomerServicePort"
  implementor="com.example.customerservice.server.CustomerServiceImpl">
  <jaxws:features>
    <ref bean="eventFeature" />
  </jaxws:features>
</jaxws:endpoint>
```

The Agent supports JMS and HTTP/HTTPS transport types in the same way.

### 3.2.3.2. Installing the Service Activity Monitoring Server into a Servlet container

The Service Activity Monitoring Server can be deployed into any Servlet container as a WAR. The Service Activity Monitoring Server requires a database to store event data, so make sure your RDBMS has been installed and started.

Also, the JNDI DataSource should be configured in the J2EE/Tomcat container, for more information, see [Configuring the Service Activity Monitoring Server in a Servlet container](#).

#### With Tomcat

1. Deploy the Service Activity Monitoring server WAR file:

- For Windows

```
copy <TalendRuntimePath>\add-ons\sam\sam-server-war.war $TOMCAT_HOME\webapps
```

- For Linux

```
cp <TalendRuntimePath>/add-ons/sam/sam-server-war.war $TOMCAT_HOME/webapps
```

2. Start Apache Tomcat:

- For Windows

```
$TOMCAT_HOME\bin\startup.bat
```

- For Linux

```
./$TOMCAT_HOME/bin/startup.sh
```

The Service Activity Monitoring Server can also be running on the Embedded Servlet container (Jetty) with the following command **mvn jetty:run-war**. For this, the following *sam-server-jetty* example is provided to quickly install and start the Monitoring Server on the Jetty Container:

#### With Jetty

1. Browse to the example:

```
cd <TalendRuntimePath>/examples/talend/tesb/sam/sam-server-jetty
```

- Execute the following command:

```
mvn jetty:run-war
```

Installing the Service Activity Monitoring Server with the **mvn jetty:run-war** command uses the embedded Derby database by default.

### 3.2.3.3. Configuring the Service Activity Monitoring Agent in a Servlet container

The main configuration files for Agents are `agent.properties` and the filter and handler configuration files. The `agent.properties` can be created by the user and placed in the classpath. Filters and handlers are based on Spring bean configuration and can be added to the application's context (for example, `beans.xml`).

Properties description:

Property	Default	Description
<code>collector.scheduler.interval</code>		Interval (in milliseconds) of Agent built-in scheduler. The Agent will make one or several calls to the Service Activity Monitoring Server sending Events from local queue at this specified value. The number of calls actually made to the Service Activity Monitoring Server when scheduler interval has arrived is decided by the number of events in the local queue and the number of <code>collector.maxEventsPerCall</code> . This interval must be greater than 0.
<code>collector.maxEventsPerCall</code>		The value of this parameter is used to restrict the max number of Events per call to the Service Activity Monitoring Server. Lower values help to avoid sending overly large SOAP body messages to the Service Activity Monitoring server.
<code>collector.lifecycleEvent</code>	false	Whether the Agent should collect and send the lifecycle events to the Service Activity Monitoring Server. If true, the Service Activity Monitoring Server must be started before the Talend Runtime container, otherwise connection exceptions will be thrown.
<code>log.messageContent</code>	true	Whether the Agent should store the Producer and Consumer SOAP message content into Events and send them to the Service Activity Monitoring Server.
<code>log.maxContentLength</code>	-1	Sets the maximum SOAP content length per Event. -1 is unlimited.
<code>log.enforceMessageIDTransfer</code>	false	If true, Service Activity Monitoring will add WS-Addressing functionality implicitly and enforce MessageID transfer between Events. If false, the MessageID will be null in the Events if the user doesn't enable the WSAddressingFeature or Policy with Addressing explicitly.
<code>service.url</code>		The URL of Service Activity Monitoring Server that the Agent is to communicate with.
<code>service.retry.number</code>	5	Number of retries when a call to the Service Activity Monitoring Server fails.
<code>service.retry.delay</code>	1000	Delay in milliseconds before the next retry to call the Service Activity Monitoring Server

For example:

```
collector.scheduler.interval=500
collector.maxEventsPerCall=10
collector.lifecycleEvent=false

log.messageContent=true
log.maxContentLength=-1
log.enforceMessageIDTransfer=true

service.url=http://localhost:8080/sam-server-war/services/MonitoringServiceSOAP
```

```
service.retry.number=3
service.retry.delay=5000
```

To filter or manipulate events, these Filter/Handler spring beans should be added into your Service provider or Service consumer bundle or jar, then these beans will be autowired by Service Activity Monitoring agent.

Some example bean definitions can be found below:

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans http://www.springframework.org/
schema/beans/spring-beans.xsd">

  <bean id="stringContentFilter"
    class="org.talend.esb.sam.common.filter.impl.StringContentFilter">
    <property name="wordsToFilter">
      <list>
        <value>abc</value>
      </list>
    </property>
  </bean>

  <bean class="org.talend.esb.sam.common.filter.impl.JXPathFilter">
    <constructor-arg value="content='test' and eventType='FAULT_IN' and customInfo/
key1='value1'"/>
  </bean>

  <bean id="passwordHandler"
    class="org.talend.esb.sam.common.handler.impl.PasswordHandler">
    <property name="tagNames">
      <list>
        <value>Password</value>
      </list>
    </property>
  </bean>

  <bean id="fixedPropertiesHandler"
    class="org.talend.esb.sam.common.handler.impl.CustomInfoHandler">
    <property name="customInfo">
      <map>
        <entry key="Application name" value="Dummy App" />
        <entry key="Stage" value="Dev" />
      </map>
    </property>
  </bean>
</beans>
```

For more information about how to use Filter/Handler in WAR applications or OSGi bundles, go to the following folder of the Talend ESB: `examples/talend/tesb/sam`.

### 3.2.3.4. Configuring the Service Activity Monitoring Server in a Servlet container

The main configuration files for the Service Activity Monitoring Server are `logserver.properties` and filter, handler configuration files.

Properties description:

Property	Default	Description
monitoringServiceUrl		The address URL published by the Service Activity Monitoring Server

Property	Default	Description
db.datasource	ds-derby	DataSource name used by the Service Activity Monitoring Server to store/query data. For J2EE or Tomcat, it should be like <code>java:comp/env/&lt;Resource name&gt;</code> , for OSGi container, use similar to <code>ds-&lt;Database&gt;</code> .
db.dialect	derbyDialect	The database used to store/query Event data (with different ID Incrementer, Query, and so on.)

For Derby:

```
monitoringServiceUrl=/MonitoringServiceSOAP
db.datasource=java:comp/env/jdbc/datasource
db.dialect=derbyDialect
```

For H2 Database Engine:

```
monitoringServiceUrl=/MonitoringServiceSOAP
db.datasource=java:comp/env/jdbc/datasource
db.dialect=h2Dialect
```

For Mysql:

```
monitoringServiceUrl=/MonitoringServiceSOAP
db.datasource=java:comp/env/jdbc/datasource
db.dialect=mysqlDialect
```

For Oracle:

```
monitoringServiceUrl=/MonitoringServiceSOAP
db.datasource=java:comp/env/jdbc/datasource
db.dialect=oracleDialect
```

For IBM DB2:

```
monitoringServiceUrl=/MonitoringServiceSOAP
db.datasource=java:comp/env/jdbc/datasource
db.dialect=DB2Dialect
```

For SQL Server:

```
monitoringServiceUrl=/MonitoringServiceSOAP
db.datasource=java:comp/env/jdbc/datasource
db.dialect=sqlServerDialect
```

For filter and handler configuration please refer to [Configuring the Service Activity Monitoring Agent in a Servlet container](#).

## 3.3. Typical use cases

Examples are provided in the Talend ESB, under the folder: `<TalendRuntimePath>/examples/talend/tesb/sam`.

For example with Service Activity Monitoring running as standalone, see: `<TalendRuntimePath>/examples/talend/tesb/sam/sam-example-service` and `<TalendRuntimePath>/examples/talend/tesb/sam/sam-example-service2`.

For example with Service Activity Monitoring running as an OSGi feature in Talend Runtime container, see: `<TalendRuntimePath>/examples/talend/tesb/sam/sam-example-osgi`.

### 3.3.1. Building examples

The sam-example-service.war and sam-example-service2.war provided as a whole customer application with sam-agent installed. They can be deployed into any Servlet container. For example, they can be deployed into Tomcat: \$TOMCAT\_HOME/webapps/.

### 3.3.2. Pre-requisites

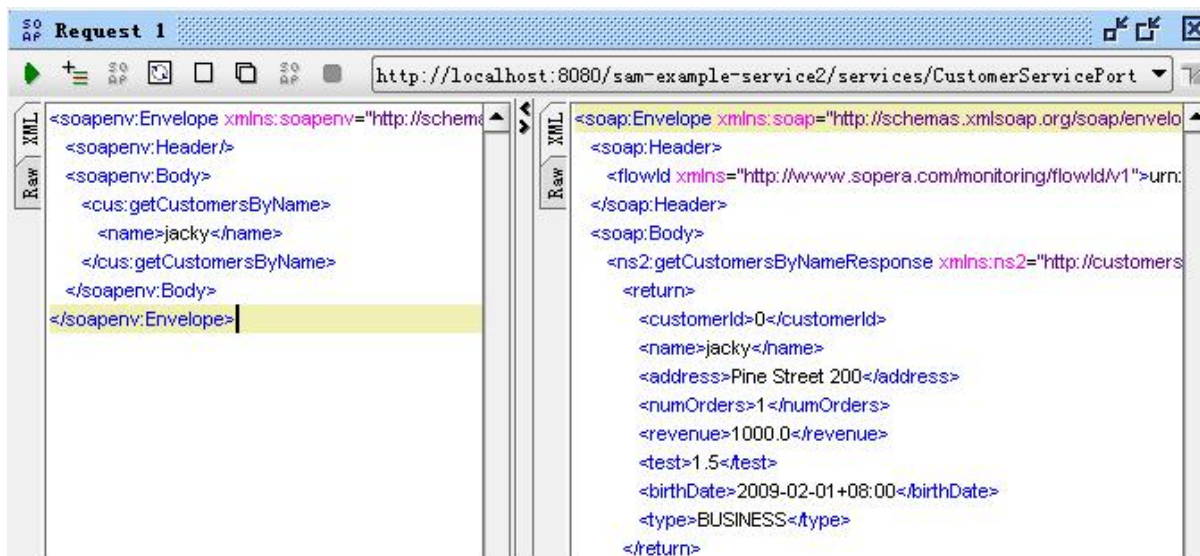
This section shows you how to run the examples (sam-example-service, sam-example-service2) with the SAM Agent supported. First, please check the following:

- Ensure the database is running and accessible.
- The Service Activity Monitoring Server is installed and running.
- The examples/talend/tesb/sam/sam-example-service and examples/talend/tesb/sam/sam-example-service2 are built, and you have deployed them into the container.
- The configuration files (agent.properties and logserver.properties at least) have been configured correctly.

### 3.3.3. General Test

Start SoapUI and send the SOAP message below to sam-example-service2 endpoint, for example: `http://localhost:8080/sam-example-service2/services/CustomerServicePort`

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:cus="http://customerservice.example.com/">
  <soapenv:Header/>
  <soapenv:Body>
    <cus:getCustomersByName>
      <name>jacky</name>
    </cus:getCustomersByName>
  </soapenv:Body>
</soapenv:Envelope>
```



### 3.3.4. Filters and Handlers Test

This test consists of three steps:

1. Add a PasswordHandler to your Application Service/Client

PasswordHandler is a pre-defined handler used to replace the real password characters with null (") for security considerations. You can set the tag name which has the password and needs to be replaced. For example:

```
<bean id="passwordFilter"
      class="org.talend.esb.sam.common.handler.impl.PasswordHandler">
  <property name="tagNames">
    <list>
      <value>Password</value>
    </list>
  </property>
</bean>
```

Then, send a Message which has the <Password> tag:

```
<soapenv:Header>
  <wsse:Security
    xmlns:wsse="http://docs.oasisopen.org/wss/2004/01/   \
    oasis-200401-wss-wssecurity-secext-1.0.xsd"
    soapenv:mustUnderstand="0">

    <wsse:UsernameToken>
      <wsse:Username>user1</wsse:Username>
      <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/   \
      oasis-200401-wss-username-token-profile-1.0#PasswordDigest">
        IR55epSSTb7sg3Z3+HKNb9MqAWg=</wsse:Password>
      </wsse:UsernameToken>

    </wsse:Security>
  </soapenv:Header>
```

The value of <Password> Element should be replaced with ".

```
<soapenv:Header>
  <wsse:Security
    xmlns:wsse="http://docs.oasisopen.org/wss/2004/01/   \
    oasis-200401-wss-wssecurity-secext-1.0.xsd"
    soapenv:mustUnderstand="0">

    <wsse:UsernameToken>
      <wsse:Username>user1</wsse:Username>
      <replaced xmlns="" />
    </wsse:UsernameToken>

    </wsse:Security>
  </soapenv:Header>
```

2. Next, add a CustomInfoHandler to your application service or client. CustomInfoHandler is a pre-defined handler used to store user-defined key/values in the database. For example:

```
<bean id="fixedProperties"
      class="org.talend.esb.sam.common.handler.impl.CustomInfoHandler">
  <property name="customInfo">
    <map>
      <entry key="Application name" value="Dummy App"/>
      <entry key="Stage" value="Dev"/>
    </map>
  </property>
</bean>
```

Then send a message, and the custom key/value properties will be stored in the database.

<input type="checkbox"/>	80102	80101	Application name	Dummy App
<input type="checkbox"/>	80103	80101	Stage	Dev

- Finally, add filter configuration on the Service Activity Monitoring Server:

Modify the Service Activity Monitoring Server's server.xml. For example:

```
.....
<bean id="monitoringService"
      class="org.talend.esb.sam.server.service.MonitoringServiceImpl">
  <property name="eventFilter">
    <list>
      <ref local="stringContentFilter" />
    </list>
  </property>
  <property name="eventManipulator">
    <list>
      <ref local="contentLengthHandler" />
    </list>
  </property>
  <property name="persistenceHandler" ref="eventRepository" />
</bean>
.....
```

- The information should now get stored in the database.

### 3.3.5. Monitoring events from database

If the events have been stored into the database successfully, you can query them from the database. For example:

<input type="checkbox"/>	80068	<soapenv:Envelope x...	317B	(NULL)	OK	2011-03-14 13:59:18	REQ_IN	(NULL)	72400DaiXilai	DaiXilai	192.168.0.54
<input type="checkbox"/>	80069	<soap:Envelope xmlns...	362B	(NULL)	OK	2011-03-14 13:59:22	REQ_OUT	(NULL)	72400DaiXilai	DaiXilai	192.168.0.54
<input type="checkbox"/>	80070	<soap:Envelope xmlns...	798B	(NULL)	OK	2011-03-14 13:59:22	RESP_IN	(NULL)	72400DaiXilai	DaiXilai	192.168.0.54
<input type="checkbox"/>	80071	<soap:Envelope xmlns...	798B	(NULL)	OK	2011-03-14 13:59:22	RESP_OUT	(NULL)	72400DaiXilai	DaiXilai	192.168.0.54
<input type="checkbox"/>	80072	<soap:Envelope xmlns...	362B	(NULL)	OK	2011-03-14 13:59:22	REQ_IN	(NULL)	72400DaiXilai	DaiXilai	192.168.0.54
<input type="checkbox"/>	80073	<soap:Envelope xmlns...	798B	(NULL)	OK	2011-03-14 13:59:22	RESP_OUT	(NULL)	72400DaiXilai	DaiXilai	192.168.0.54
<input type="checkbox"/>	80074	<soapenv:Envelope x...	293B	(NULL)	OK	2011-03-15 14:38:11	REQ_IN	(NULL)	50560DaiXilai	DaiXilai	192.168.0.54
<input type="checkbox"/>	80077	<soap:Envelope xmlns...	802B	(NULL)	OK	2011-03-15 14:38:11	RESP_OUT	(NULL)	50560DaiXilai	DaiXilai	192.168.0.54



If you purchased one of Talend's products with ESB functionalities, there is a graphical interface provided by the Talend Administration Center, for viewing the Service Activity Monitoring information. Please see the *Talend Installation Guide* and the *Talend Administration Center User Guide* for more details.

Note: If you wish to view the Service Activity Monitoring user interface in the Talend Administration Center, then both need to be deployed in the same Tomcat Servlet container.





## Chapter 4. Event Logging

The Event Logging feature within the Talend ESB allows users to collect events across distributed containers and also provides the ability to index them and search through them via a Web User Interface. In addition to the pure collection of events, the Event Logging feature supports custom processing (for example: custom filtering, customer data enrichment and reduction), aggregation, signing and also server-side custom pre- and post-processing of event to send them as a post processing step to an Intrusion Detection system or to any other kind of potential higher level log processing and management system, for example.

Event Logging is only available in the subscription version of Talend ESB; thus, it is not included in Talend Open Studio for ESB. It can be used in combination with the Service Activity Monitoring feature (SAM Agent, SAM Retrieval Service and the User Interface of the Service Activity Monitoring in Talend Administration Center). However note that the use of the Event Logging or the Service Activity Monitoring Server is optional and can be activated when needed. With Talend Open Studio for ESB, only the Service Activity Monitoring Server is available.

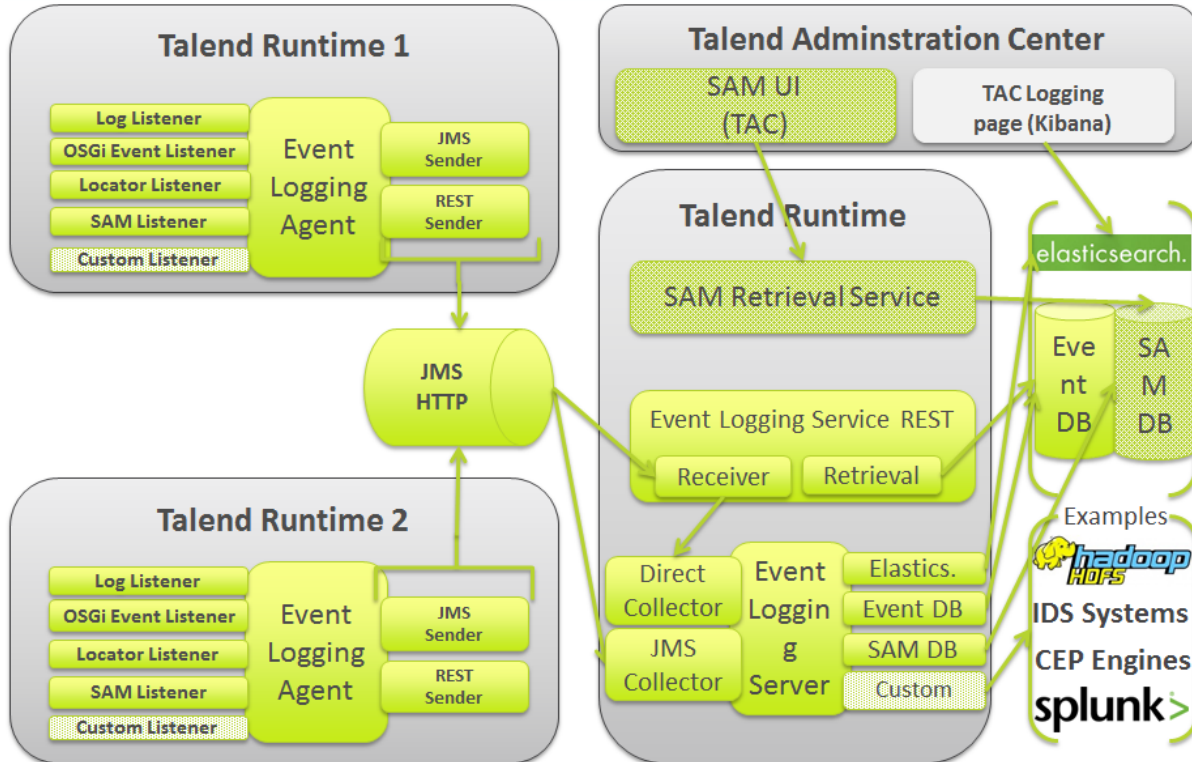
In addition, Talend provides a log indexing and searching functionality on Event Logs, based on Elasticsearch and Kibana: the Talend Log Server and the **Logging** page in Talend Administration Center.

## 4.1. Overview

The primary sub-parts of the Event Logging are:

- **Event Logging - Listener**, which gets log information and passes it to the Agent for further local processing.
- **Event Logging - Agent**, which receives events from listeners, buffers, processes and sends them to a final destination using one of the configured Event Logging Appenders.
- **Event Logging - Sender**, which sends events to the Event Logging backend. A JMS Appender and a REST Appender for the EventLogging RESTService are provided by default to write into a local log file or to a local elastic search instance, for example. But customers can add their own appenders.
- **Event Logging - Service**, which is a RESTful Service to collect events sent from the Event Logging REST Appender and to retrieve events from the backend. Events are stored in the Event Database only since version 5.4 of Talend ESB.
- **Event Logging - Collector**, which consists of routes which listen on endpoints to receive events for further server-side processing by the Event Logging Server. By default, a Standard Collector which exposes a fixed Direct-VM endpoint, and a JMS Collector which takes Event out of a JMS Queue are provided.
- **Event Logging - Server**, which retrieves events from the collector and performs processing and persistence of events, with support for Event Database (RDBMS), Service Activity Monitoring Database (Service Activity Monitoring Events only), Elasticsearch with some custom pre- and post-processing extension points.
- **Talend Log Server**, based on Elasticsearch.
- **Logging** page in Talend Administration Center, based on Elasticsearch and Kibana.

The following high level architecture shows the different components and their relations:



Technically, the Event Logging Listener, Agent, Sender, Collector and Server are implemented using Apache Camel. The Event Logging Service is a CXF (JAX-RS) based service developed in Java (also using Apache Camel for the Receiver part). The Elasticsearch-based Talend Log Server and Kibana-based **Logging** page in Talend Administration Center is available as a separate deployment outside the Talend Runtime container and is optional

for the use of the Event Logging feature. The Event Logging Database is supported on all databases supported by the Service Activity Monitoring Database (PostgreSQL included).

The following sections describe the installation and starting of the Event Logging feature, the individual components of the overall Event Logging feature, and the Data Structures and public API of this feature. For more information about the installation and starting of the Talend Log Server and Talend Administration Center, see the *Talend Installation Guide*.

## 4.2. Installing the Event Logging in the Talend Runtime container

The Event Logging feature is preinstalled in the Talend Runtime container.

To use the Event Logging feature with Elasticsearch, you should follow these steps:

1. Install and start Talend Log Server.

For more information about its installation and starting, see the *Talend Installation Guide*.

2. Start all the Talend ESB Infrastructure Services with the **tesb:start-all** command. For more information, see the *Talend ESB Container Administration Guide*.

3. Start the Event Logging. There are two ways to start/stop it:

- start/stop all default bundles necessary at once. See [Default start/stop](#).

With this option, you will use only one command to launch/stop all the components necessary to use the Event Logging feature, with the default profile.

- manually start/stop each bundle. See [Manual start/stop](#).

With this option, you will launch/stop each Event Logging component individually.

Now, all events will be collected and logged into the Talend Log Server, which stores events in index folders that change every day. To determine the URL to those events, you have to use the `elasticsearch.indexname` parameter value: `talendesb`, set by default in the `org.talend.eventlogging.server.cfg` configuration file, and followed by the current date `-<yyyy>.<mm>.<dd>` (where `yyyy` corresponds to the current year, `mm` the current month and `dd` to the current day).

For example:

```
http://localhost:9200/talendesb-2014.08.06/_search?pretty=true
```

For more information about the use of the Event Logging, see the *Talend Administration Center User Guide*.

### 4.2.1. Default start/stop

After starting the Talend Runtime container, to start the Event Logging (EL) with the default profile, enter the following command at the console prompt:

```
tesb:start-el-default
```

This will start the following components:

- Internal Derby Database,

- Event Logging Server,
- Event Logging Direct Receiver,
- Event Logging REST Service,
- Event Logging REST Sender,
- Event Logging Agent,
- Event Logging Log Listener,
- Event Logging Locator Listener,
- Event Logging SAM Listener.

Only the OSGi Listener is not started with the default profile, so if you want to use it, you have to manually start it with the following command:

```
tesb:start-el-osgilistener
```

Furthermore, if you want to use the Robust event processing feature to make sure none of the events are lost or skipped, as it is optional, it is not started by default with the `tesb:start-el-default` start command. So, you will have to start it manually as well by entering `tesb:start-el-dlq` at the console prompt. For more information about this feature, see [Robust event processing](#).

You can shutdown the Event Logging with default profile by entering:

```
tesb:stop-el-default
```

However, if you started additional manual components as described above, you will have to execute their corresponding `tesb:stop-*` command as well.

## 4.2.2. Manual start/stop

If you are using individual commands, the components must be started in a specific order:

1. The Event Logging Server must be started before the JMS Receiver or Direct Collector.
2. The Direct Collector must be started before the REST Service.
3. The REST Sender and/or JMS Sender must be started before the Event Logging Agent (depends on Event Logging Agent configuration).
4. The Event Logging Agent must be started before the Log|OSGi|SAM|Locator Listeners.

There are commands for starting/stopping individual Event Logging components. So, to start each Event Logging component individually in the required order:

1. Start the Talend Runtime container.
2. Make sure the DataSource to be used by the Event Logging Server is started.
3. Enter the following commands at the console prompt:
  1. `tesb:start-el-server` - starts the Event Logging Server.
  2. `tesb:start-el-jmsreceiver` - starts the JMS Receiver.
  3. `tesb:start-el-restservice` - starts the REST Service and the Direct Receiver.
  4. `tesb:start-el-restsender` - starts the REST Sender.

5. `tesb:start-el-jmssender` - starts the JMS Sender.
6. `tesb:start-el-agent` - starts the Event Logging Agent.
7. `tesb:start-el-loglistener` - starts the Log Listener.
8. `tesb:start-el-osgilistener` - starts the OSGi Listener.
9. `tesb:start-el-samlister` - starts the SAM Listener.
10. `tesb:start-el-locatorlistener` for the Locator Listener
11. `tesb:start-el-dlq` - starts the Robust event processing feature.

For each `tesb:start-el-*` command, there is a corresponding `tesb:stop-el-*` command. So, to stop the components, execute the `tesb:stop-el-*` commands in the reverse order.



*It is strongly recommended not to stop the Event Logging Agent when any of the listeners are still running.*

## 4.3. Event Logging - Listeners

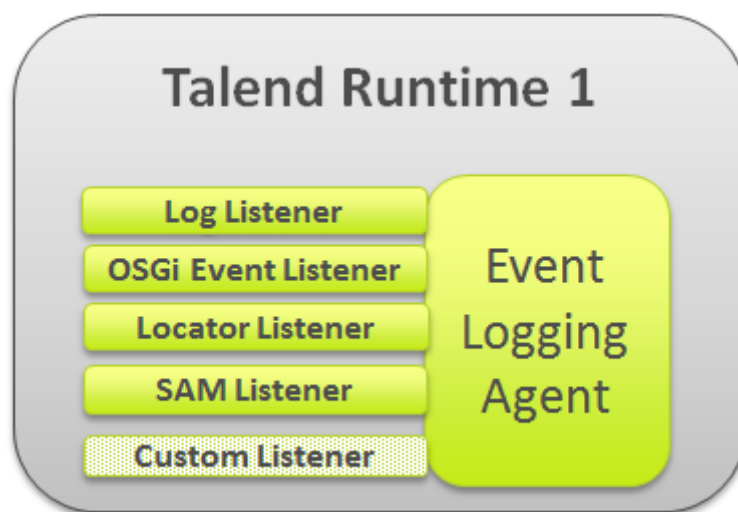
This section describes the different listeners available in Talend Event Logging feature of Talend ESB.

Talend ESB supports four default listeners:

- [Log Listener](#), to get all Pax Logging log events.
- [OSGi Event Listener](#), to get all OSGi Events on Talend Runtime.
- [SAM Listener](#), to get Service Activity Monitoring Event (via SOAP) from the Service Activity Monitoring Agent.
- [Locator Listener](#), to get the Locator events.

Additionally, the customer can also create his own Custom Listeners.

From an architectural point, the listeners are tightly coupled with the Event Logging Agent and require the Event Logging Agent to work.



Nevertheless, each listener can be started and stopped individually, as there is a single OSGi Bundle per listener as described in [Manual start/stop](#).

- **tesb:start-el-loglistener / tesb:stop-el-loglistener** for the Log Listener.
- **tesb:start-el-osgilistener / tesb:stop-el-osgilistener** for the OSGi Listener.
- **tesb:start-el-samlistener / tesb:stop-el-samlistener** for the SAM Listener.
- **tesb:start-el-locatorlistener / tesb:stop-el-locatorlistener** for the Locator Listener.

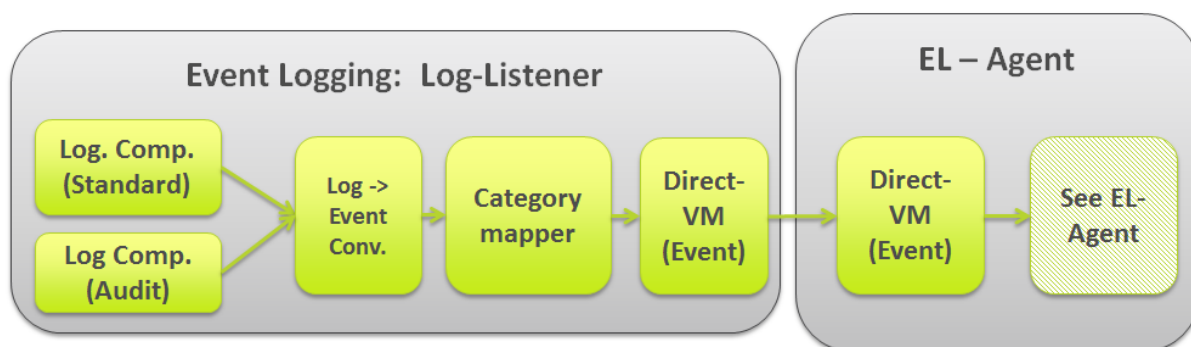
In addition to the above listeners, provided as part of the standard product container and feature, Custom Listeners which the user can implement using an Apache Camel Route (by using our RouteBuilder tooling, for example), are supported as long as:

- the same Apache Camel version, as the one used in the container where the listener will be deployed, is used,
- the *direct-vm*: component is used as producer with a fixed destination (route ID) *eventloggingagent*,
- they are deployed on the same JVM or Talend Runtime container as the Talend Event Logging Agent,
- the Camel Exchange Header contains the log information in the related event properties structure (event-related camel exchange header properties) and the log message itself within the exchange body (plain text). The EventCategory is optional for the listener but preferred to be assign and defined already in the listener.

### 4.3.1. Log Listener

The Log Listener allows the user to get all the log information, which typically will also be available in `tesb.log`, into the Event Logging. This means that the Log Listener is the primary listener for the Event Logging Agent.

The pax logging, also used by Talend Runtime, is used as the entry point for the Event Logging Log Listener. This way, the user can also use the standard pax logging configuration to configure which logs should be sent to the Log Listener and which logs should not, like for any other log appender.



To enable logging via the Log Listener, the following minimal entry is needed in the `org.ops4j.pax.logging.cfg` configuration file:

```
log4j.rootLogger=INFO, out, osgi:VmLogAppender, osgi:eventloglistener
```

The `osgi:eventloglistener` entry will now also provide the log information to the Log Listener.

Additionally and alternatively, the Log Listener will expose a second pax logging appender named `eventloglisteneraudit` where all log events sent through this appender will be set to `audit=true` and thus, would have the highest priority of all audit configurations. In general, if an Event has the audit flag set to true, no other Talend components or mappings will change it back to false. But if the audit flag is false, it might be set to true later on, by additional mapping options. The `eventloglisteneraudit` can be used in the pax logging configuration and would automatically make all events, sent via this appender, 'Audit' events.

The Log Listener is implemented using Apache Camel, via the *paxlogging*: component, and provides a configurable Category Mapper processor with a Direct-VM communication to the Talend Event Logging Agent.

The `paxlogging` component will have a fixed configuration, to expose `eventloglistener` as appender name: `paxlogging:eventloglistener`. Therefore, the `log4j.rootLogger=INFO, out, osgi:VmLogAppender` can be extended by `osgi:eventloglistener` to enable logging via the Event Logging Log Listener.

As pax logging is used, all the log messages, which are sent via different logging frameworks supported by Talend ESB, will be captured via this listener:

- Log4J-based logging: `log4jLogger.info("log4j log message");`
- SLF4J-based logging: `slf4jLogger.info("slf4j log message");`
- JDK-based logging: `jdkLogger.info("JDK log message");`
- JCL-based logging: `jclLogger.info("JCL log message");`
- Juli-based logging: `juliLogger.info("juli log message");`

And in case the logging framework supports MDC attributes, the MDC attributes are also taken into the event as additional metadata. This would also allow custom and business code logs to contain a business correlation ID or other important metadata in a structured form, for example, using `MDC.put("CorrelationID", "abc")`.

The "Log to Event" conversion step will transform the log message format into the Event format, where the Event Structure is as completely filled as possible, and the additional metadata (including the MDC attributes) is transformed in the Event Customer Information (Key, Value) list.

The Category Mapper allows the user to define an Event Logging category in a configurable way to the log message by package name from where the message comes from, and if messages from this package should be treated as audit or non-audit messages.

The Log Listener uses the `org.talend.eventlogging.listener.log.cfg` configuration file with the following parameters:

```
# Default category for events
category.default=system

# Define key in MDC attributes which will be used to get EventCategory
category.attribute=eventCategory
# Define key in MDC that specify audit
audit.attribute=watchThis

# Category mapping configuration, for example, if a log message comes from
# package org.apache.cxf, it will be mapped to the service category
category.mapping.org.apache.cxf = service
category.mapping.org.apache.cxf.rt.security = security(audit=true)
category.mapping.org.apache.wss4j = security(audit=true)
category.mapping.org.talend.esb.sts=security(audit=true)
```

The configuration is primarily for the Category Mapper and will be used as follows:

1. The first priority to define which `EventCategory` is assigned to the current log event is defined by an MDC attribute. The name of this attribute can be configured under `category.attribute`. By default, it is `eventCategory`. If this attribute exists, the Category will be set to the value of this attribute.
2. If the attribute does not exist, the Category Mapper will look for the `category.mapping` configuration. It will build a tree in memory, will go from the leaves to the root of this tree to find the closest node to the given package name, as the `log4j` log appender package based filter configuration would do, and will apply the category and audit flag definition as defined in the configuration file, but only if the audit attribute is not already set to true and no overwrite of true is allowed for the audit flag. Once the audit flag is true, it will stay true.
3. If the `audit.attribute` parameter of the log event has a custom value and the audit setting is currently set to false, the audit setting will automatically be activated and it will take the custom value referred to in the `audit.attribute` parameter.

Example:

- In the example above, if a log message comes from package `org.apache.cxf.binding.soap`, it will be mapped to the `service` category, with audit implicitly set to false (`audit=false`), as the `org.apache.cxf` is the matching node configuration.
- If a log message comes from `org.apache.cxf.rt.security.saml`, it will be mapped to the `security` category and marked as an audit message by the additional audit definition after the category: (`audit=true`).

By default, if audit is not explicitly defined after the category, event will be set to `audit=false`.

- If the audit setting of the event was not yet set to true, and its attribute contains a custom value, for example: `watchThis`, the audit will automatically be set to true.

After this step, the log message is sent to the Event Logging Agent in a synchronous way (Direct-VM) to allow the agent to do a short pre-processing before the event is stored into a local buffer within the agent for final processing and sending to the backend.

### 4.3.1.1. Propagating custom information in Log events

The MDC feature of existing logging frameworks like `log4j`, `slf4j`, and so on, can be used to propagate custom or user information as metadata in the Log event. This allows custom or business code logs to contain a business correlation ID for example, or other important metadata in a structured form. The following Java code snippet demonstrates how a custom information can be added as MDC attribute in the log message:

```
import org.apache.log4j.Logger;
import org.apache.log4j.MDC;

public class SimpleMDC {
    static public void main(String[] args) throws Exception {

        // You can put values in the MDC at any time. Before anything else
        // we put the subject name
        MDC.put("subjectName", "Joe");

        [ SNIP ]

        Logger logger = Logger.getLogger(SimpleMDC.class);
        // We now put the last name
        MDC.put("loggedInAs", "Admin");
        logger.info("Check enclosed.");
        MDC.put("myCorrelatonID", "154516516521");
        logger.info("Using business correlationid");

    }

    [ SNIP ]

    MDC.clear();
}
```

To get the MDC values printed in the logs, the `log4j.properties` configuration must be adapted. For example:

```
# Root logger
log4j.rootLogger=INFO, stdout
# CONSOLE appender not used by default
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} | %-5.5p | %-16
.16t| %X{subjectName} %X{loggedInAs} %X{myCorrelatonID}|%m%n
```

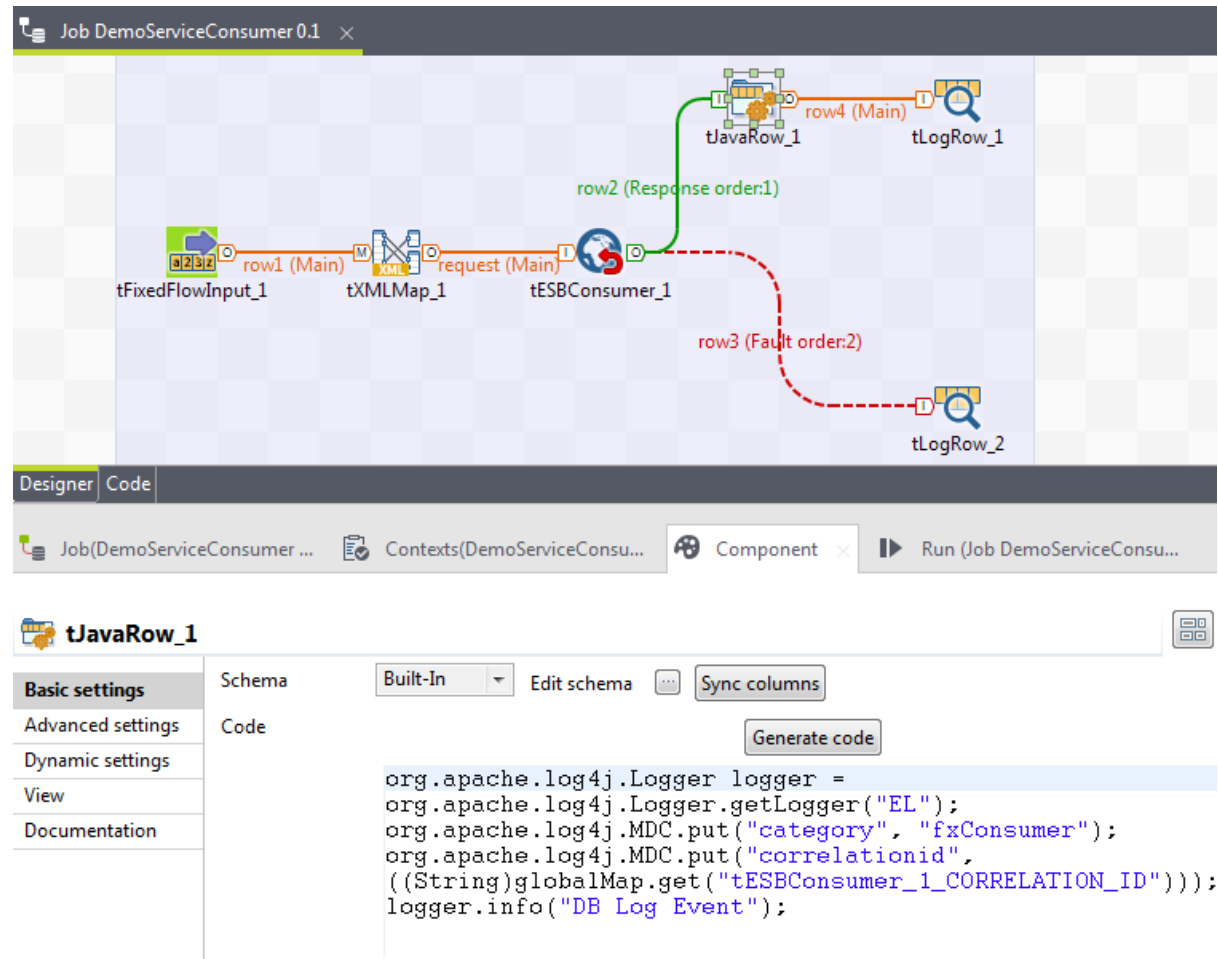
If you have the above `log4j` configuration, the log output will look as follows:

```
09:33:57,203 | INFO | SimpleMDC | Joe Admin |Check enclosed.
09:33:57,204 | INFO | SimpleMDC | Joe Admin 154516516521|Using business correlationid
```



The Log -> Event conversion step will transform the log message format into the Event format, where essentially the Event Structure is filled in as completely as possible and the additional metadata, including the MDC attributes, are transformed in the Event Customer Information (Key, Value) list.

If you are in the Talend Studio, you can use the tJavaRow component to add your custom MDC attributes as shown by the image below:



To activate the MDC properties while using Mediation routes, the cConfig component can be used as follows:

1. Create a new mediation route.
2. Add a cConfig component.
3. To enable the MDC logging in Camel, type in the following code in the Component view of the **cConfig**:

```
context.setUseMDCLogging(true);
```

4. Add a cLog component to the route.

For more information, go to: <http://camel.apache.org/mdc-logging.html>

#### 4.3.1.2. Adding attributes to Log Events automatically

The Logging Framework supports the use of MDC properties to enrich log events with attributes like Subject and CorrelationID, as shown in a previous section. To automatically assign these attributes to their log event, you can optimize the Event Logging feature to extract the CorrelationID and Subject information from your SOAP messages with a CXF interceptor (if available), and directly set them as MCD properties. This way, all

log messages within the service implementation would directly benefit from the enrichment of each log statement with Subject and CorrelationID, making it a lot easier for system administrators to correlate log messages.

To do so, the MDC mapper should be installed and configured in the Talend Runtime container:

1. Install the MDC mapper as follows:

```
feature:install tesb-el-mdc-mapper
```

All CXF Webservices within the same container will then automatically profit from this feature. No additional configuration for any Web service is required.

2. Make sure the MDC mapper comes with the following default configuration in the `org.talend.eventlogging.mdc.cfg` file:

```
# MDC key value for CoorelationID
mdc.correlationId = CorrelationID

# MDC key value for authenticated user name
mdc.principle = Subject
```

Currently, only two values can be extracted and mapped to MDC values:

- A username taken from the SecurityContext (thus being independent from authentication style)
- CorrelationId taken from the message context (if available)

Once the MDC mapper installed and configured in the Talend Runtime container:

- The value defined for `mdc.correlationId` will be used as the MDC property key, in this case: CorrelationID.
- The value defined for `mdc.principle` will be used to map the authenticated username to a MDC property with the key: Subject.

If both values are available within the message context, this feature will set two MDC properties for each log message (within the same context). For example:

```
{
  "eventUUID": "9452dd3c-dbd3-47c0-a98e-398f89c9e78a",
  . . .
  "customInfo": {
    "CorrelationID": "someBusinessCorrelationId",
    "Subject": "alice"
  }
}
```

In most cases, CorrelationID and Subject MDC properties will be defined within the agent configuration file to match the corresponding event log fields, and in that case, these values will not be stored as customInfo attributes, but as CorrelationID and Subject attributes like this:

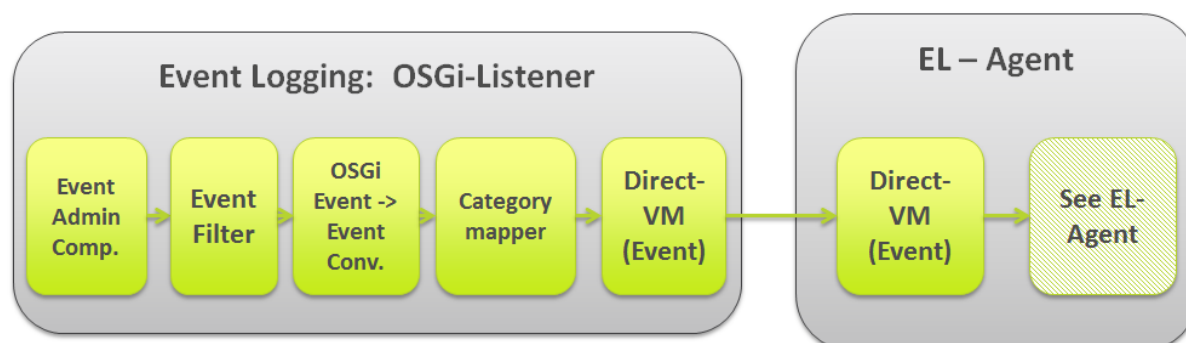
```
{
  "eventUUID": "9452dd3c-dbd3-47c0-a98e-398f89c9e78a",
  "correlationId": "someBusinessCorrelationId",
  . . .
  "subject": "alice",
  "customInfo": {
  }
}
```

## 4.3.2. OSGi Event Listener

The OSGi Event Listener allows the user to get the OSGi Events from the Talend Runtime container which are published via the EventAdmin service.

To enable Event Logging via the OSGi Listener, only the related listener bundle needs to be started. No other container related configuration is required by our standard Talend Runtime container.

The OSGi Event Listener is implemented using Apache Camel, via the *eventadmin*: component, and provides a configurable Topic Filter and Category Mapper with a Direct-VM communication to the Talend Event Logging Agent.



The *eventadmin*: component has a fixed configuration to listen (subscribe) to all topics: `eventadmin:*`. Talend ESB provides an easy-to-use Topic Filter within the OSGi Event Listener. In the configuration, the user can provide a list of topics to be included and/or excluded, including the use of "\*" for all.

```

filter.include.*
filter.include.org/osgi/service
filter.exclude.org/osgi
  
```

The Topic Filter allows the user to quickly include and/or exclude all and/or just a few selected ones. As shown in the example, each topic, or partial topic, requires a row. And if an included topic matches the current topic, it will be included, as long as it is not matching an excluded topic.

Inclusion has priority before exclusion is evaluated.

Example:

In the example above, all events are included, except the one which starts with `org/osgi/`. But within the `org/osgi/` topics, the ones in the `org/osgi/service/` sub-topic will be included.

The OSGi Event -> Event conversion step will transform the OSGi Event into the Event (Logging) format, where the Event Structure is as completely filled as possible, and the additional metadata (including the OSGi Event Properties) is transformed in the Event Customer Information (Key, Value) list.

The Category Mapper allows the user to define an Event Logging category in a configurable way for the OSGi Event, based on the OSGi Event topic, and if this event should be treated as an audit event or not.

The configuration related to the Category Mapper is as follows:

```

# Default category for events
category.default=osgi
category.attribute=eventCategory
audit.attribute=watchThis
# Category mapping configuration
category.mapping.org/osgi/framework/ServiceEvent=service
category.mapping.org/osgi/service=service(audit=true)
  
```

The priority is defined by the log Category Mapper.

After this step, the OSGi Event message is sent to the Event Logging Agent in a synchronous way (Direct-VM) to allow the agent to do a short pre-processing before the event is stored into a local buffer within the agent for final processing and sending to the backend.

The OSGi Event Listener uses the `org.talend.eventlogging.listener.osgi.cfg` configuration file with the following parameters:

```
# Default category for events
category.default=osgi
# Name of key used to get events
category.attribute=eventCategory
audit.attribute=audit

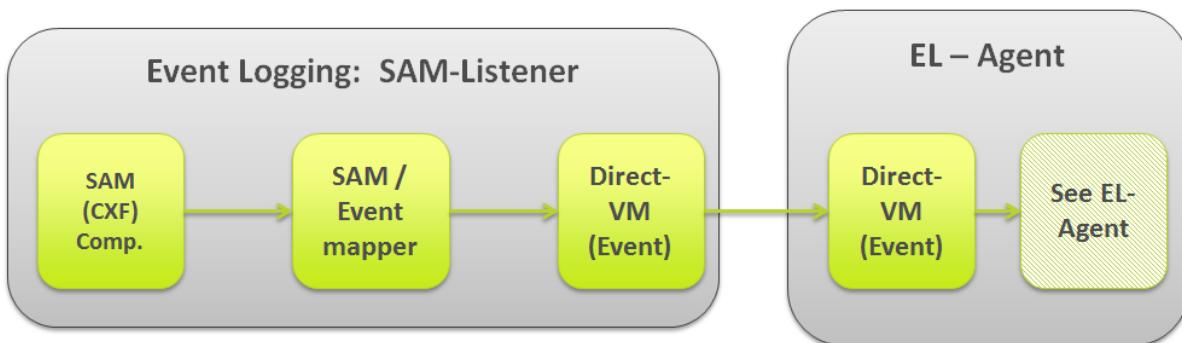
# Category mapping configuration
category.mapping.org/osgi/framework/ServiceEvent=service
#category.mapping.org/osgi/service=service(audit=true)

# Filter configuration, by default, all events are included except
# the one defined in the filter.include. properties
filter.exclude.*
filter.include.org/osgi/framework/ServiceEvent
filter.include.org/osgi/framework/BundleEvent
filter.include.org/osgi/framework/FrameworkEvent
```

Similar to the Log Listener, it will allow to set the audit flag based on the `audit.attribute`, which is here an OSGi Property, and/or on the Category Mapping. In the current version of Talend ESB, it is the only property which can be set as part of the mapping, but in future versions, the `attribute=value` notation would allow to extend this to other attributes: `attribute1=value1; attribute2=value2; custominfo={{key1, value1}{key2, value2}}`.

### 4.3.3. SAM Listener

The SAM Listener allows the user to get the Service Activity Monitoring Events from the Talend Service Activity Monitoring Agent into the Event Logging Agent, with the ability to use all the features of the Event Logging for Service Activity Monitoring Events, and to combine easily Service Activity Monitoring Events with other Event Logging events into a single search or audit list.



To enable Event Logging via the SAM Listener, only the related listener bundle needs to be started and, as usual for the Service Activity Monitoring, the related design time configuration to use it must be applied (or as with Talend ESB version 5.4 and higher), and the related Service Activity Monitoring Custom policy must be used at runtime via the Service Registry to enable Service Activity Monitoring for the related ESB Consumer/Provider.

The SAM Listener is implemented using Apache Camel, via the `cxfr` component, and exposes a SOAP Service which is exactly the same as the Service Activity Monitoring Server one (SOAP Service - `MonitoringServiceSOAP`, with the `putEvents` operation). See the WSDL below for a real example.

To use the SAM Listener, you first need to stop the Service Activity Monitoring Server, if the service is already started on the current container. To do so, use the `tesb:stop-sam` command.

The WSDL of the Service Activity Monitoring Server, used to retrieve Service Activity Monitoring Events, is available at <http://localhost:8040/services/MonitoringServiceSOAP?wsdl>.

```
<wsdl:definitions name="MonitoringWebServiceService" targetNamespace="http://service.server.sam.esb.talend.org/">
```

```

    <wsdl:import location="http://localhost:8040/services/MonitoringService
SOAP?wsdl=MonitoringService.wsdl" namespace="http://www.talend.org/esb/sam/
MonitoringService/v1">
    </wsdl:import>
    <wsdl:binding name="MonitoringWebServiceServiceSoapBinding" type="ns1:M
onitoringService">
        <soap:binding style="document" transport="http://schemas.xmlsoap.or
g/soap/http"/>
        <wsdl:operation name="putEvents">
            <soap:operation soapAction="http://www.talend.org/esb/sam/Monit
oringService/v1/putEvents" style="document"/>
            <wsdl:input name="putEvents">
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="putEventsResponse">
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="PutEventsFault">
                <soap:fault name="PutEventsFault" use="literal"/>
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="MonitoringWebServiceService">
        <wsdl:port binding="tns:MonitoringWebServiceServiceSoapBinding"
name="MonitoringWebServicePort">
            <soap:address location="http://localhost:8040/services/Monitori
ngServiceSOAP"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>

```

Basically, the SAM Listener exposes the Service Activity Monitoring Server API, so the SAM-Agent just needs to use the right URL to send the SAM-Events to the SAM Listener instead of the Service Activity Monitoring Server. In both cases, it can be a localhost or remote host. This means that a single server side Event Logging Agent with one SAM Listener can be used as a remote replacement to the standard Service Activity Monitoring Server. This is the only listener so far which can be used both locally and remotely.

The Event Logging SAM Listener only supports the following two configuration settings which is used in the "SAM to Event" converter in the `org.talend.eventlogging.listener.sam.cfg` configuration file:

```

# By default all events captured by the SAM Listener will be assigned
# to the sam category
category=sam

# if SAM events from this container would be treated as audit events
audit=false

# if map the subject of the SAML token to the subject attribute of the even
t
map.subject.samltoken=false
# if map the user name of the UsernameToken to the subject attribute of the
# event
map.subject.usertoken=false

```

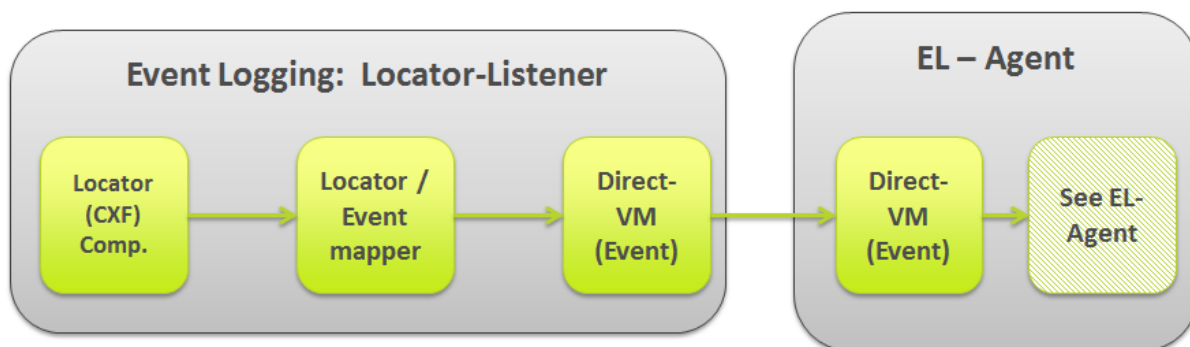
- `category=sam` means that all events captured by the SAM Listener will be assigned to the *sam* Event Category and that all those events will be set to `audit=false` but can be easily changed.
- `audit=true` means that if SAM events from this container would be treated as audit events, the above configuration will be used as default: `category=sam; audit=false`.
- `map.subject`. The SAM Event Listener particularly gives access to information about the user or about the subject of the Service call and enables to map this user or subject as an Event Subject, or not, with the following options. For performance reasons, the search will be performed from the top of the log message (exchange body) by string search to the related element name, and the values will be retrieved by pure string search to avoid an explicit XML document creation.

- `map.subject.samltoken=true` will map the subject of the SAML token to the subject attribute of the event (true /false).
- `map.subject.username.token=true` will map the user name of the UsernameToken to the subject attribute of the event (true /false).

If you want to use the dashboard specific to **Service Activity Monitoring** in addition to the **ESB SAM** one available the **Logging** page in Talend Administration Center, use the `tesb:start-sam-retrieval-service` command.

## 4.3.4. Locator Listener

The Locator Listener allows the user to get the Service Locator Events from the Talend Service Locator Agent into the Event Logging Agent, with the ability to use all the features of the Event Logging for Service Locator Events, and to combine easily Service Locator Events with other Event Logging events into a single search or audit list.



Once the Locator Listener installed and started, you can configure the interval between two searched of locator endpoints:

1. Open the `etc/org.talend.esb.monitoring.locator.cfg` configuration file.
2. Update the following parameter:

```
scanIntervall=15 #the value is in seconds
```

## 4.4. Event Logging - Agent

The Event Logging Agent is responsible for collecting events from the Event Logging listeners: to buffer these events for further local processing, to process the events (including signing, data enrichment, custom processing), and sending out events to the central Event Logging Server. The Agent should only do the required minimal processing before an event is stored into the local buffer, to keep the listener waiting as little as possible. And the Agent should work offline as long as the buffer technology supports to collect events locally, and allows events and containers to continue to be collected while the needed backend services are temporarily unavailable. The temporary amount of unavailable time would approximately be up to eight hours, but it is only an estimate, not a fix limit.

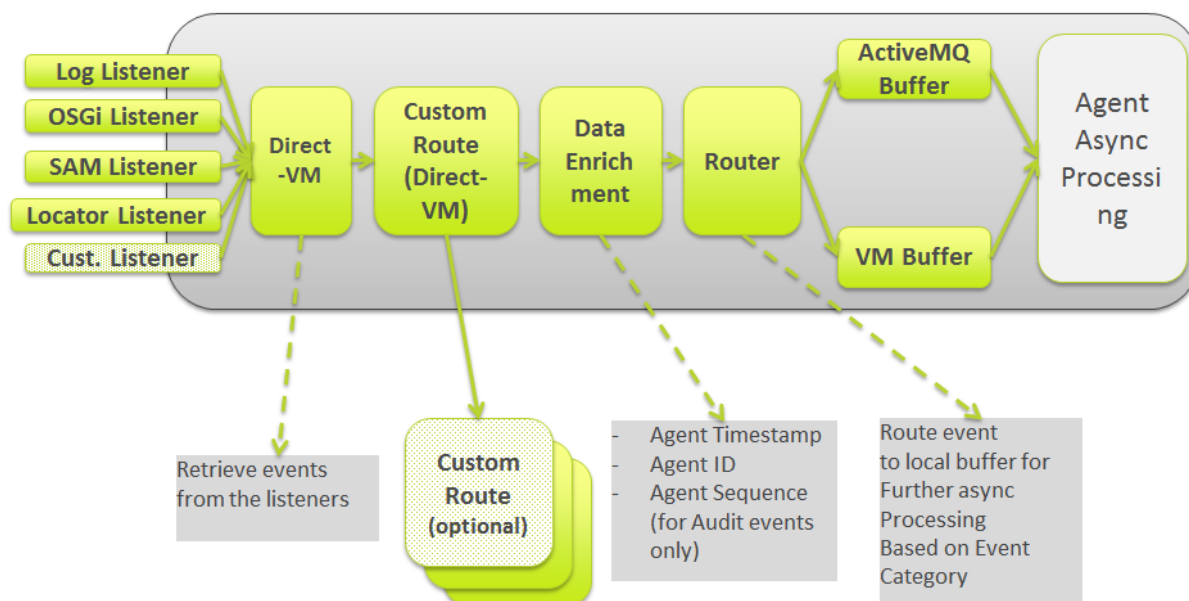
In the following sections, the Agent is split into two logical parts:

- **Receiver part** - to get events from the listeners until they reach the local buffer.
- **Processing part** - to process each event from the local buffer.

The two parts will be described in the following sections in more details.

## 4.4.1. Agent - Receiver part

The Agent Receiver part is responsible for getting the messages from the local listeners and doing a minimal processing before the event gets stored in the local buffer.



1. The Event Logging listener sends each event via in-memory synchronous communication (*direct-vm*: component) to the Agent using a fixed route ID *eventloggingagent*.
2. The user can optionally configure a custom route at this early stage to filter, shorten, enrich metadata of the log event, as soon as the listener receives it.

The custom route must be deployed on the same JVM or on the same Container and must expose a unique route ID via the *direct-vm*: component. The custom route has full access to the Event data. The configuration part of the `org.talend.eventlogging.agent.cfg` file in the agent configuration is as follows:

```
agent.receiver.custom.routeid.default=myCustomReceiverRoute
agent.receiver.custom.routeid.audit=myCustomReceiverRoute
agent.receiver.custom.routeid.security=myCustomReceiverSecurityRoute
```

The `agent.receiver.custom.routeid.` is the fixed part followed by the event category. `default` and `audit` are reserved categories and can not be used as normal category names. Default will be used if no specific mapping for the current event category is found and the default custom route will be called. If the Event is already marked as an audit event and if an audit custom route is defined, this one will be called. Even though we do not limit what the user can do at this stage with the custom route, it is strongly advised that these early processing routes are designed for maximum performance as we are still in a synchronous processing with the listener at this stage. By default, no custom route for none of the categories is defined. The above example will be commented out, for reference, with an `#` character in our default agent configuration file, and no custom route will be called by default.

3. The data enrichment component in the agent will add the minimal required system data to the event before it is stored for further processing in the local buffer.

At this stage, if one of the following attributes of the Event Data Structure is not already filled (for example by the listener, or the customer route), the Data Enrichment will fill the values as follows:

Attribute	Value	Remark
eventuid	string (UUID)	If empty, it will be generated within the agent.
agentid	string (Unique agentID )	It is retrieved from the agent configuration file. Example: <code>property agentID=agent1.</code>

Attribute	Value	Remark
agent_timestamp	timestamp	Local machine date and time (converted to UTC time). See the event structure.  Example: 2013-07-23T08:45:30.453Z
hostname	string (hostname)	Hostname of the current machine.  If the hostname cannot be resolved via the Java API, the IP address will be used as fallback.
processid	string (processID)	Current JVM process ID.
audit	boolean (default=false, if not set at this stage)	
auditsequenceno	long - unique sequence number for audit event	A unique sequence ID will be assigned only if the event is marked as audit. The sequence ID is a long number that starts with 1 and is incremented for each new audit event from this agent. To avoid any repetition, <i>agentid</i> and <i>auditsequenceno</i> should be shown on the tables in the Backend Event Database with no gaps. The <i>auditsequenceno</i> will be persisted in a local file and saved each time a new number is used.
correlationid	string	If empty, an attribute will be found via the <i>event.correlationid.map.attribute</i> configuration parameter in the <i>eventlogging_custominfo</i> attribute collection.
subject	string	If empty, the <i>event.subject.map.attribute</i> configuration will look in the in the <i>eventlogging_custominfo</i> attribute collection to find an attribute.

The configuration parts for this step are defined in the `org.talend.eventlogging.agent.cfg` as follows:

```
# Unique agentID (string)
agentid=agent1

# The location of file system used to persist audit sequence
agentsequencedir=./data/audit-sequence

# Take correlation from customInfo using this key
event.correlationid.map.attribute=CorrelationID

# Used to map an attribute to the Subject property in the Event. User can
# provide this value in his logs as attributes (e.g. for log events as
# MDC attribute)
event.subject.map.attribute=Subject
```

4. The Router will send the Event to the appropriate configured local buffer. Two buffers are supported in Talend ESB:

- *jms*, a JMS-based buffer (for the time being, only ActiveMQ is supported) with or without persistent queue, intended to be a local ActiveMQ Broker for the local buffer.
- *memory*, use of the in-memory queue, as provided by the VM (component of Apache Camel).

The router configuration is defined in the agent as follows:

```
# Receiver buffer config
# For 'agent.buffer.jms.queue' (and only for it) its possible to
# reuse other property values defined in this file by putting
# property name between ${ and }. For example:
#   agentid=some-agent-id
#   agent.buffer.jms.queue=event.logging.${agentid}.cache
# will be interpreted by EL as 'event.logging.some-agent-id.cache'
agent.buffer.jms.url = vm://eventloggingbroker?create=true&broker.useJ
mx=false&broker.persistent=true
agent.buffer.jms.queue =event.logging.${agentid}.cache
agent.buffer.jms.username=tadmin
agent.buffer.jms.password=tadmin

# By default, all events will be sent via a memory buffer but the user
# can easily change this and use jms instead for audit event and for
```

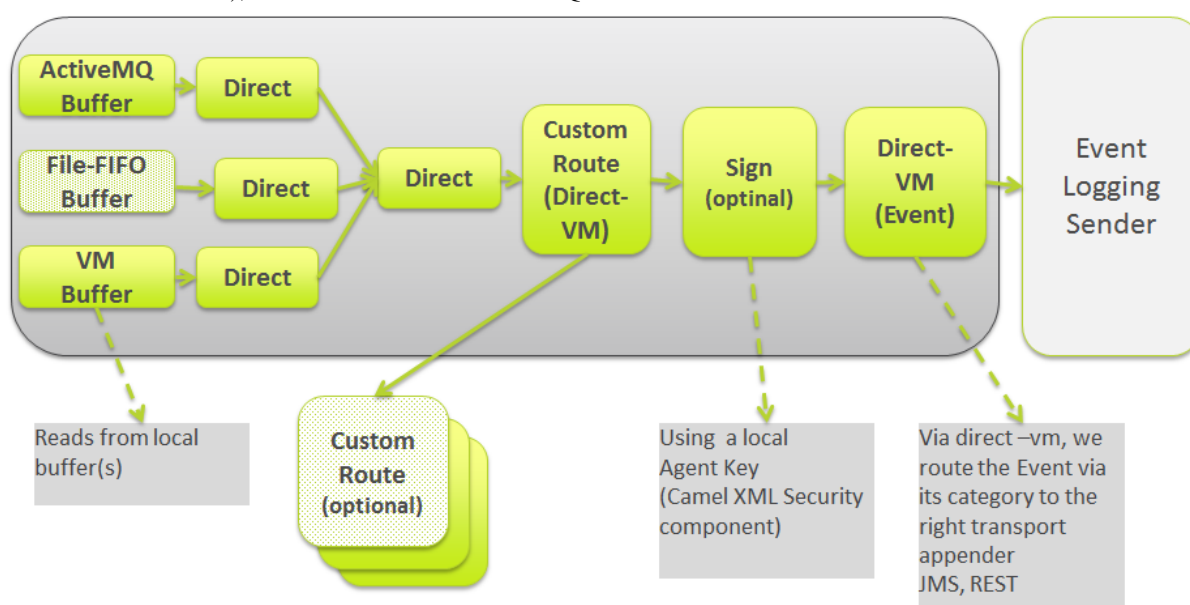


```
# other event categories.
agent.receiver.buffer.default=memory
#agent.receiver.buffer.audit=jms
#agent.receiver.buffer.security=jms
```

By default, all events will be sent via a *memory* buffer but the user can easily change this and use *jms* instead for audit event and for other event categories. For example, in the example above, if the user uncomments `agent.receiver.buffer.security=jms`, all events of the *security* category, that correspond to the value after `agent.receiver.buffer.` (static part), will also be stored via the *jms* buffer.

## 4.4.2. Agent - Processing Part

The Agent - Processing Part is responsible for getting the messages from the local buffer and processing them up to the point where they are ready to be sent to the final destination, to the Event Logging Collector Service (via HTTP/HTTPS REST), or to the Server JMS Broker Queue.



Consumer of the local buffer will read existing buffered event from the buffer and start processing it via a synchronous processing and send it via *direct:* communication (transactional, in the JMS case) to the core processing route which will start a custom processing route, if one is defined for the given event category in the `org.talend.eventlogging.agent.cfg` file.

Example:

```
agent.processing.custom.routeid.default=myCustomProcRoute
agent.processing.custom.routeid.audit=myCustomProcRoute
agent.processing.custom.routeid.security=myCustomProcSecurityRoute
```

In the example above, the `myCustomProcSecurityRoute` will be called for the *security* category and the `myCustomProcRoute` will be called for the *audit* one and all others (*default*). By default, no custom processing route will be called.

In the next step, the Event Log message (as stored as plain text in the exchange body) will be signed and a *signedLogMessage* header property will be created which contains the logMessage in a XML Digital Signature (enveloped). The Camel XML Security component (based on Apache Santuario) is used for the XML DSIG signature creation.

If signing is required, it can be defined by category within the `org.talend.eventlogging.agent.cfg` file.

Example:

```
agent.processing.signing.default=false
agent.processing.signing.audit=true
agent.processing.signing.security=true
```

In the above example, *audit* events and events in the *security* category (but which are not marked as audit) will be signed, but all others (with the *default* definition) will not be signed.

The default is:

```
agent.processing.signing.default=false
agent.processing.signing.audit=true
```

The keystore and the certificate configuration used for signing the events are defined by the following within the `org.talend.eventlogging.agent.cfg`:

```
agent.signing.keystore.properties=./etc/keystores/trunKeystore.properties
```

A default local keystore (`trun.jks`) is provided for the private key which will be used to sign the log message, however it is strongly recommended to use a custom keystore and certificate for production.

After the signing step, the Camel Exchange (Event Structure) is treated as ready and the event is sent with the *direct* component synchronously (transactional in the JMS case) to the sender part of the Agent.

The last part of the processing is a *direct-vm*: component which gets configured per event category as follows:

```
agent.sender.destination.default=eventlogsenderrest
agent.sender.destination.audit=eventlogsenderjms
```

With those two parameters, the event will be sent to the Event Logging Sender according to its related category. Two sender destinations are available:

- `eventlogsenderrest`
- `eventlogsenderjms`

Even though these values are just the route IDs where the *direct-vm*: will send the event to, with this configuration, the customer can easily create custom senders as routes with a Direct-VM Endpoint and by configuring the related route ID of the exposed direct-vm endpoint.

```
agent.sender.destination.system=eventlogsendermysender
```

The above parameter would send the event to the route with the `eventlogsendermysender` route ID. And this route can put the log event wherever it likes.

### 4.4.2.1. Event log enrichment

Event log enrichments can be added to the log events, for example, an attribute can be added to the Subject property in the Event, static log event attributes can be added to all log messages processed within the agent and log event attributes can be removed from all log messages processed within the agent. To do so, you need to edit the `etc/org.talend.eventlogging.agent.cfg` configuration file.

- To map an attribute to the Subject property in the Event, edit the following parameter:

```
event.map.subject = Subject
```

- To retrieve the correlation ID from `custom_Info`, edit the following parameter:

```
event.map.correlationid = CorrelationID
```

- To add static log event attributes to all log messages processed within the agent, edit this parameter: `event.add.[attribute-name] = [value]`

Example:

```
event.add.logSource.country=Germany
event.add.logSource.city=Bonn
event.add.customInfo.projectID=POC Talend ESB
```

Currently only `event.add.logSource.[sub-attribute]` and `event.add.customInfo.[sub-attribute]` are supported. Be careful, once the values defined here, they will override any previously existing values within the log event.

- To remove log event attributes from all log messages processed within the agent, edit this parameter:  
`event.remove.[attribute-name]`

Example:

```
#event.remove.logSource.class.name
#event.remove.customInfo.activemq.broker
```

Currently only `event.remove.logSource.[sub-attribute]` and `event.remove.customInfo.[sub-attribute]` are supported. This feature is especially helpful if you need to remove (customInfo) MDC properties from third party components.

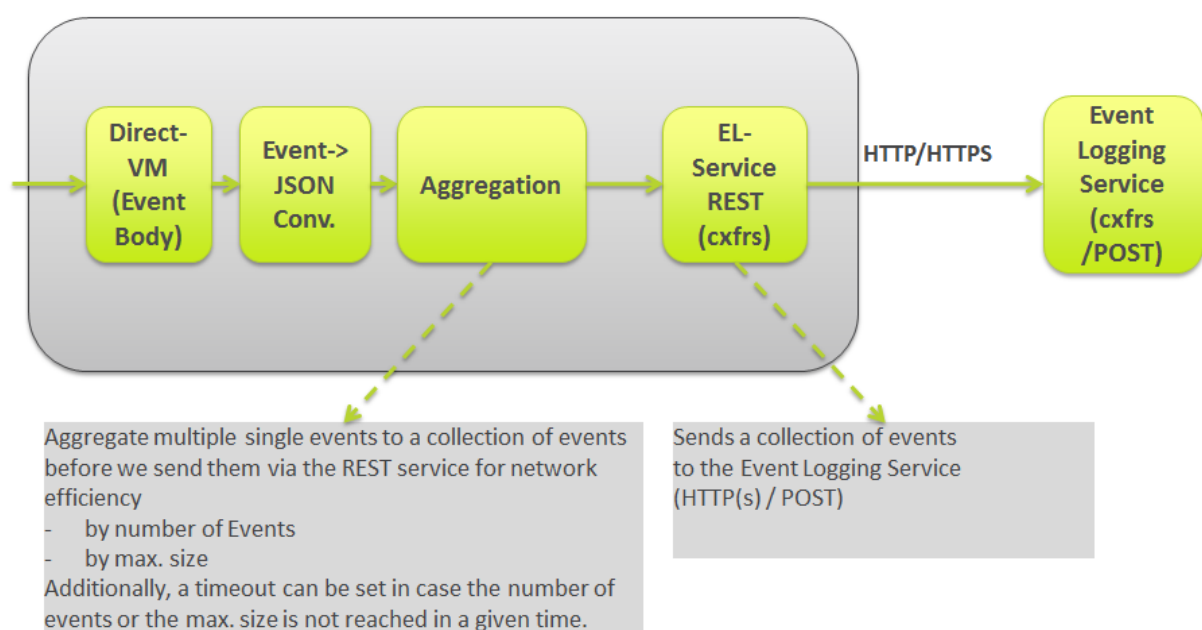
## 4.5. Event Logging - Sender

By default, two Senders are supported for the Agent:

- a JMS-based sender
- a REST-based sender

The Event Logging Sender are responsible for retrieving the messages from the processing part of the Agent, and sending them to the final destination, which can either be a JMS Broker Queue (for the time being, only ActiveMQ is supported as broker for Event Logging Events in Talend ESB) or the Event Logging Collector Service (HTTP/HTTPS - RESTful service).

### 4.5.1. REST Sender



The sender receives the event from the processing part via the *direct-vm:* component and does a technical conversion from the exchange header and body to a JSON format which is stored in the exchange body and all header fields of the event will be removed.

After the conversion from the Exchange Event Structure to the JSON structure, an aggregator will be used to optimize the network transfer with the following aggregation strategy defined in the `org.talend.eventlogging.sender.rest.cfg` configuration file:

```
sender.aggregation.eventcount=10
sender.aggregation.eventsize=1024 # in KB
sender.aggregation.sendtimeout=20000 # in milliseconds
```

Which will be interpreted as follows:

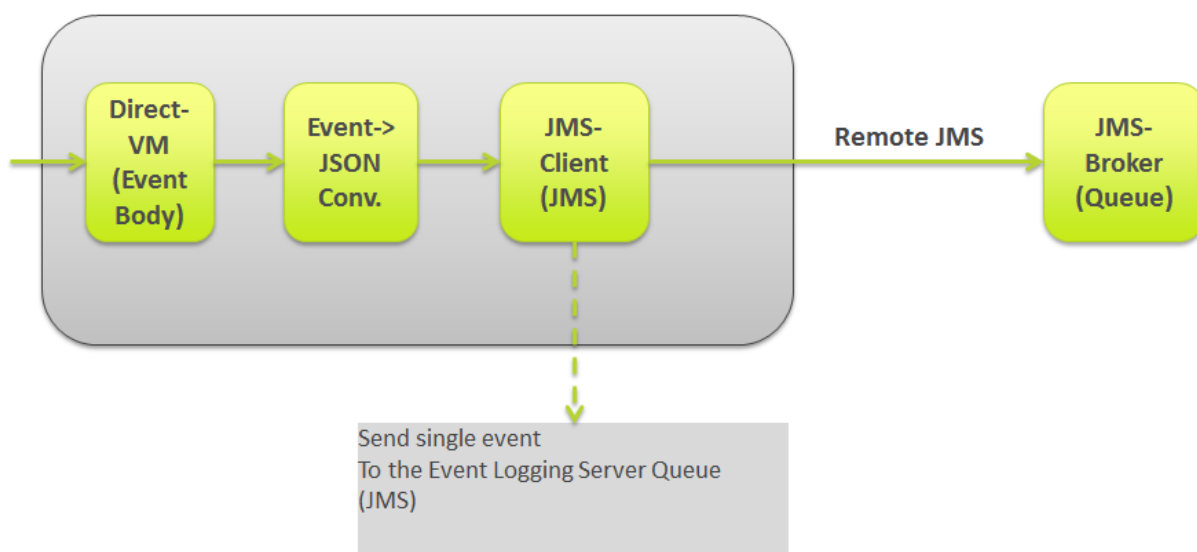
- If the amount of individual aggregated events reaches 10, the collection of events will be sent as one REST - (POST) to the backend remote Event Logging Service.
- If the count is not reached but the total size of the event collection within the aggregator reaches the maximum size of 1024 KB, the collection will also be sent, even if the count is not reached.
- And finally, if the count is not reached and the size is still below the maximum threshold, the event will be sent if the timeout (in milliseconds) is reached, in the above case after 20000 milliseconds. In the above configuration, if the event count is set to 1, it would mean that no real aggregation is done, even though all events will be handed by the aggregator.

This way, events will be sent in a network-optimized way while still be sent in a timely manner. As the aggregator collects events in memory, this transport destination is not as reliable as the JMS option and audit events should preferably always be sent via JMS even though the default will be *service* for all events in order to limit the initial setup effort.

The *service* destination can be configured in the `org.talend.eventlogging.sender.rest.cfg` file as follows:

```
sender.destination.service.url = https://localhost:8040/eventlogging/events
sender.destination.service.authentication=NO # NO, BASIC
sender.destination.service.username=tadmin
sender.destination.service.password=tadmin
```

## 4.5.2. JMS Sender



The sender receives the event from the processing part via the *direct-vm*: component and does a technical conversion from the exchange header and body to a JSON format which is stored in the exchange body and all header fields of the event will be removed from the exchange.

After the conversion from the Exchange Event Structure to the JSON structure, the event (exchange body as JMS message) will be sent via the *jms* component (so far, only ActiveMQ Broker is supported in Talend ESB).

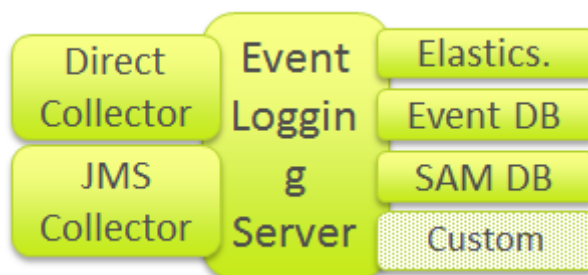
The *jms* destination can be configured in the `org.talend.eventlogging.sender.jms.cfg` file as follows:

```
sender.destination.jms.url=tcp://localhost:61616
sender.destination.jms.queue=event.logging.server
sender.destination.jms.username=tadmin
sender.destination.jms.password=tadmin
```

## 4.6. Event Logging - Server

Within the overall Event Logging architecture, the Event Logging Server will collect and receive all the events from the agents and send them to the defined destination. Primarily, the service will be able to save the events into the new EventLogging Database (RDBMS), into the Service Activity Monitoring Database (for Service Activity Monitoring Events only), to a custom persistence destination and to Elasticsearch for indexing.

In general, the server will not process much, to be able to handle a large amount of events as fast as possible.



In the following sections, the details of the Collection Server and how it can be configured are described. Technically, the Collection Server is implemented using Apache Camel and is preinstalled in Talend Runtime (subscription version only).

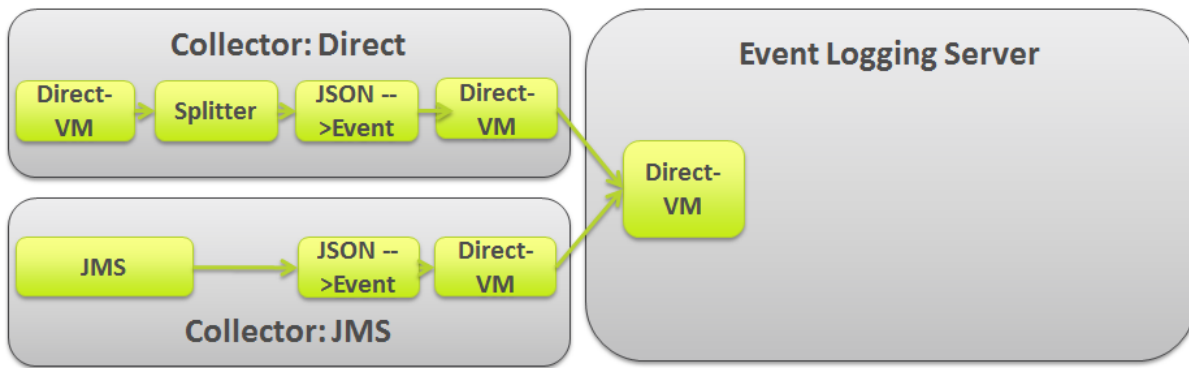
### 4.6.1. Collector

In this part of the Server, events are received (via Direct-VM or JMS).

Each collector has its own bundle and additional collectors can be added by the user, if needed.

The product supports:

- Direct Collector, which exposes a direct-vm endpoint and which is used by the Event Logging Service for new events (POST).
- JMS Collector, which reads and gets events from a JMS Broker.



The Direct Collector does not need any configuration, exposes a direct-vm endpoint: *eventlogcollector*, and uses the direct-vm fix endpoint *eventlogserver* to put the event into the server processing part.

The JMS Collector needs to be configured in the `org.talend.eventlogging.collector.jms.cfg` configuration file:

```
collector.jms.url = tcp://localhost:61616
collector.jms.queue=event.logging.server
collector.jms.username=tadmin
collector.jms.password=tadmin
collectorjms.dlq.buffer.jms.queue = collectorjms.dlq
```

Here, you can configure the queue and JMS parameters to connect to the remote broker, and the queue used by the JMS Collector to store events in case it is not able to deliver them to the Event Logging server. By default, the `event.logging.server` queue is used but it can be changed.

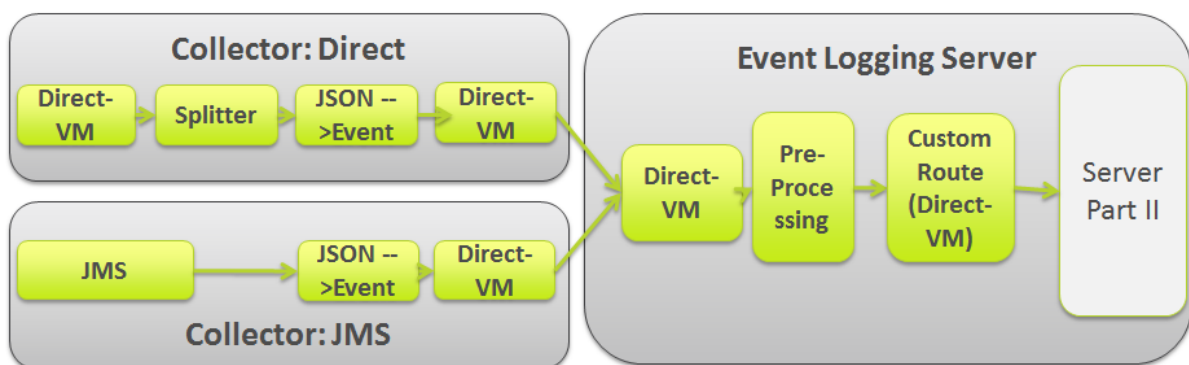
In Talend ESB, only ActiveMQ is supported as broker.

The Direct Collector is able to work with collections of events (1 and more events at the same time) and it has the logic to split the events in single events for further processing. Technically, this is the counterpart to the *RESTSender* on the Agent side which has the feature to Aggregate events to a collection of events.

Events received via the JMS Collector are always received as one event per read. Therefore, no splitting is required in this case.

Both APIs expect that the Event is provided by the Event Logging Agent even though events can technically also be sent directly to the Collector Service if they fulfil the format and completeness required by the event structure.

## 4.6.2. Server Pre-processing part



After one of the collector routes receives the event(s) and sends it/them via Direct-VM communication to the server route, the pre-processing will be done as a first step.

The system Pre-processing step will do the following:

- Check if the category of the event is *ping*, and if it is the case, it will discard the event without any further processing. Ping events are sent by the agent to check the Broker and Queue availability during the period where the network or the broker communication is not working as expected. But as defined in the Agent, the *ping* category is the third reserved category. These events are no real events and processing will stop immediately at this stage for ping events.
- Convert the JSON body back to the Event Exchange Structure, as defined in the Event Structure Camel Exchange mapping below.
- Check if an Event Category exists and if not, applies a default one to the event.
- Set the server timestamp to the Event.

The following configuration in the `org.talend.eventlogging.server.cfg` file applies to this part:

```
preprocessing.eventCategory=system
```

This parameter set the event to the *system* category if no Category exists until this stage, which can only be the case if the event was not sent via the Event Logging Agent.

After this step, the user can configure a user-defined pre-processing using a customer route, which can be configured as follows for each individual category in the `org.talend.eventlogging.server.cfg` file:

```
preprocessing.custom.routeid.default=myCustomPreProcRoute
preprocessing.custom.routeid.audit=myCustomrAuditPreProcRoute
preprocessing.custom.routeid.sam=myCustomSamPreProcRoute
```

As in the Event Logging Agent, the `preprocessing.custom.routeid.` will be followed by the category name, where *default* and *audit* are reserved. Default configuration will be applied to all categories which are not explicitly mapped. Audit will be applied to all events which have the audit flag set, regardless of the event category.

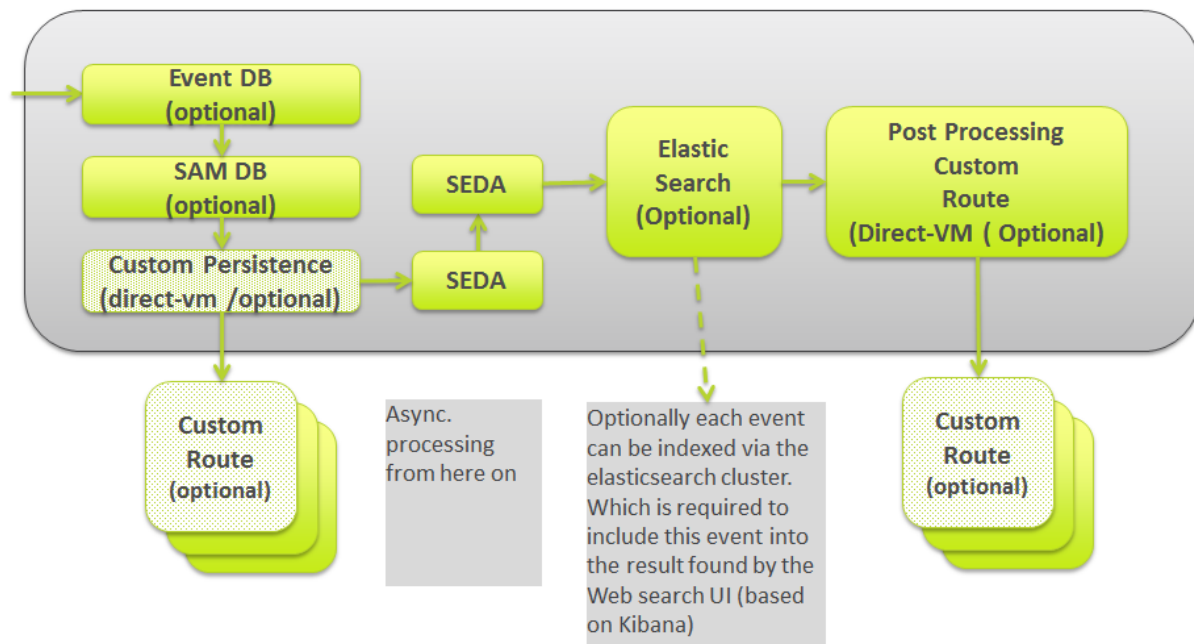
In the above example, *audit*, and *sam* events not marked as *audit*, will be pre-processed by the related custom routes while all other events will be pre-processed by the default route `myCustomPreProcRoute`.

By default, no custom processing is required and no processing is configured.

The custom processing will be started using the configured route ID via a Direct-VM communication between routes. To do this, the customer route needs to be deployed on the same Talend Runtime or JVM as the one on which the Collector Server is running.

### 4.6.3. Server persistence and post-processing

After the pre-processing part, the event goes into the persistence, search and post-processing steps, which are shown in the following diagram:



An event can be handled by one or more persistence backends:

1. Event Logging Database,
2. Service Activity Monitoring Database (only applicable for events created by the Service Activity Monitoring Agent),
3. the custom persistent step.

None of these steps is mandatory (even the Event Logging Database one) and each step can be activated or deactivated by category.

Example:

```

persistence.event.db.active.default=false
persistence.event.db.active.audit=true
persistence.event.db.active.sam=false

persistence.sam.db.active.default=false
persistence.sam.db.active.sam=true

persistence.custom.active.default=false
persistence.custom.active.system=false
persistence.custom.routeid.system=myCustomServerPersistenceRoute
  
```

In the above example configuration:

- Events with the `audit=true` parameter will be stored in the Event Database.
- Events with the `sam` category will be stored in the Service Activity Monitoring Database.
- Events with the `system` category will be handled by the custom route named `myCustomServerPersistenceRoute` and called via Direct-VM, even though what the custom route does with the event is up to the custom route.

The Event Database has the database structure as defined in the sub-[Event Database Structure](#) of the [Event Logging - API's and Data Structures](#), and is accessible via JDBC using a predefined datasource:

```

event.logging.db.datasource=el-ds-mysql
event.logging.db.dialect=mysqlDialect
  
```



The Service Activity Monitoring Database is exactly the same database as the one used by the Service Activity Monitoring Server. The Service Activity Monitoring Database step will remap the Event Structure to the Service Activity Monitoring Event structure and will try to map custom information fields to the fixed columns of the Service Activity Monitoring Event structure (flowid, servicename, and so on), as this data is provided by the Service Activity Monitoring Agent and just - in between - converted to the Event Logging Structure. All information is available and the Service Activity Monitoring Database will be filled consistently in the same way as the Service Activity Monitoring Server is. This way, records written by the Event Logging Service Activity Monitoring Database step can also be retrieved and viewed via the Service Activity Monitoring Retrieval Service (part of the Service Activity Monitoring feature) and via the Service Activity Monitoring User Interface in the Talend Administration Center.

The Service Activity Monitoring Database has the same database structure as the one defined by the Service Activity Monitoring feature, for more information, see the Service Activity Monitoring chapter in the *Talend ESB Infrastructure Services Configuration Guide*, and is accessible via JDBC using a predefined datasource:

```
sam.db.datasource=ds-mysql
sam.db.dialect=mysqlDialect
```

The Event Database persistence uses the same data source as the one used by the Service Activity Monitoring Server. This way, the configuration and setup of the Database driver is exactly the same as the one described in the Service Activity Monitoring chapter of the *Talend ESB Infrastructure Services Configuration Guide*.

It is important to note that an event can be handled by each of the persistent steps. This means a single event can be saved in the Event Database, Service Activity Monitoring Database and handled by the custom route. Even though this implementation will certainly reduce the performance of the overall handling of a single event, it is technically possible with the feature.

After this, the synchronous processing is done and an asynchronous seda communication will now be used.

The last step is to send events to:

- the search indexing (Talend Log Server based on Elasticsearch).
- a final custom post-processing, for example, to reformat the event and to send it to an intrusion detection system (IDS) stored on Hadoop HDFS, to process this event further with Big Data technologies, or to send it to a Complex Event Processing engine (CEP) or to a larger scale log analysing system like Splunk, or to any other destination it can be sent to.

The search indexing step is optional and can be configured per category in the `org.talend.eventlogging.server.cfg` file.

Example:

```
search.active.default=false
search.active.sam=true
```

In the above example, only the *sam* Event category events will be indexed by the Talend Log Server, and no other event.

Example:

```
search.active.default=true
search.active.sam=false
```

In the example above, all events will be indexed, except events with the *sam* category. Unless, they have the audit flag set to true. In that case, the default is used, so they will be indexed. To avoid this, `search.active.audit=false` must be configured to also exclude audit events from being indexed.

The Search Indexing step would convert the Event Exchange structure to a JSON format. The same conversion as the one performed by the Talend Event Logging Agent to send it to the Event Logging Collector Server.

The Event Logging Search Service is based on Elasticsearch and can be configured as follows in the `org.talend.eventlogging.server.cfg` file:

```
elasticsearch.available=true
elasticsearch.host=localhost
elasticsearch.port=9200
elasticsearch.indexname=talendesb
elasticsearch.indextype=ESB
```

With the `elasticsearch.available` parameter, the entire feature can be activated or deactivated. If it is deactivated, the category-based configuration shown above will not be used at all.

The `elasticsearch.host` specifies how the Talend Log Server or Cluster can be reached, locally on the same machine or remotely. The other parameters are specific to the Talend Log Server.

The last step in the Collector Service is the Custom Post-processing step, which allows the user to send the message to any kind of destination.

For example, the user might want to reformat the event and send it to an intrusion detection system (IDS), store it on Hadoop HDFS to process this event further with Big Data technologies, send it to a Complex Event Processing Engine (CEP) or to a larger scale log analysing system like Splunk, or any other destination.

Example:

```
postprocessing.custom.routeid.default=myCustomServerPostRoute
postprocessing.custom.routeid.audit=myCustomServerAuditPostRoute
postprocessing.custom.routeid.security=myCustomServerSecurityPostRoute
```

As in many other configurations, the related route will be called via Direct-VM and must be deployed on the same Talend Runtime or JVM as the Talend Event Logging Collector Server.

## 4.7. Event Logging - Service

The Event Logging Service can be used to query and read events from the Event Database, and only from the Event Database, and to post events via REST to the Event Logging Server.

The service will be implemented in Java and will provide a RESTful API as described in [Event Logging Service API](#) of [Event Logging - API's and Data Structures](#).

The Retrieval Service can be configured via the `org.talend.eventlogging.service.cfg` file:

```
# Authentication method BASIC,NO
eventlogging.authentication = NO
eventlogging.retrieval.api.enabled = false
```

The database settings will be used from the Event Logging Server configuration. Therefore, the Event Logging Service and the Event Logging Server currently need to be co-located. The `eventlogging.authentication` can be used to define the authentication method for the RESTful service.

- NO = no authentication
- BASIC = HTTP Basic Authentication (plain text)

Additionally, via the standard container configuration, the Retrieval service can be made accessible via HTTP and/or HTTPS. The `eventlogging.retrieval.api.enabled` parameter can be used to enable or disable the access to the REST event retrieval APIs.

## 4.8. Talend Log Server

The Talend Log Server is based on Elasticsearch. Elasticsearch is an open source search and analytics engine that makes data easy to explore.

The Talend Log Server will be installed outside the Talend Runtime container and is optional for the Event Logging feature. But when the Talend Log Server is installed, the log data - even large amounts of log data - can be searched much more quickly and easily. And in the combination with the Event Logging Search Web Application, it is easier for users to find related data by different criteria including full text search.

For more information about the Talend Log Server, see the *Talend Administration Center User Guide*. And for more information about Elasticsearch, go to its Web site: <http://www.elasticsearch.org/>.

## 4.8.1. Configuring the EventLogging server to connect to a secured Elasticsearch

This procedure explains how to configure the EventLogging server to access a secured Elasticsearch.

### Prerequisite:

- You must have [set up SSL/TLS for Elasticsearch with Shield](#).

- Start the Talend Runtime container to access the secured Elasticsearch.

```
source scripts/configEventLogging_REST.sh agent1
tesb:start-locator
```

- Edit the `etc/org.talend.eventlogging.server.cfg` file in the Talend Runtime container, as follows:

```
elasticsearch.secured=true
elasticsearch.ssl.truststore=any
elasticsearch.ssl.truststore.password=none
elasticsearch.ssl.pass.hostname=any
elasticsearch.username=es_admin
elasticsearch.password=es_admin
```

You can set different values for the `elasticsearch.username` and `elasticsearch.password` properties, depending on the Elasticsearch configuration.

- (Optional) Create a TrustStore in the JKS format by importing the Elasticsearch server public key:

```
keytool -export -rfc -keystore keystore.jks -storepass password -alias mykey -file
esServer.cer
keytool -import -trustcacerts -keystore truststore.jks -storepass password -alias
mytrust -file esServer.cer -noprompt
```

In the `etc/org.talend.eventlogging.server.cfg` file, add the TrustStore configuration properties:

```
elasticsearch.ssl.truststore=./etc/keystores/truststore.jks
elasticsearch.ssl.truststore.password=password
```

With this configuration, the EventLogging server only trusts the certificates provided in the `truststore.jks` file.

Refresh the Elasticsearch server bundle to activate the changes:

```
karaf@trun>refresh event-logging-server
```

- From your browser, open [https://localhost:9200/talendesb-\*\*<yyyy>.<mm>.<dd>/ESB/\\_search?pretty=true\*\*](https://localhost:9200/talendesb-<b><yyyy>.<mm>.<dd>/ESB/_search?pretty=true</b>) to test the configuration.

You should see the *hits* and *total* numbers increasing continuously. If not, refer to the troubleshooting note below.



### Troubleshooting

```
[2016-10-19 16:13:46,437][DEBUG][rest.suppressed      ] path: /talendesb-2016.10.19/
ESB, params: {index=talendesb-20
16.10.19, type=ESB}
ElasticsearchSecurityException[unable to authenticate user [es_admin11] for REST request
[/talendesb-2016.10.19/ESB]]
    at
    org.elasticsearch.shield.support.Exceptions.authenticationError(Exceptions.java:39)
```

The input you provided in the `elasticsearch.username` or `elasticsearch.password` property is not correct.

```
[2016-10-17 11:19:13,069][WARN ][shield.transport.netty  ] [Knickknack] Caught
exception while handling client http tra
ffic, closing connection [id: 0x22446442, /127.0.0.1:52527 => /127.0.0.1:9200]
javax.net.ssl.SSLException: Received fatal alert: certificate_unknown
    at sun.security.ssl.Alerts.getSSLException(Alerts.java:208)
    at sun.security.ssl.SSLEngineImpl.fatal(SSLEngineImpl.java:1666)
```

The `elasticsearch.ssl.truststore` parameter has been configured but the JKS TrustStore does not reference the imported Elasticsearch certificate.

```
Caused by: java.security.UnrecoverableKeyException: Password verification failed
    at sun.security.provider.JavaKeyStore.engineLoad(JavaKeyStore.java:778)[1.8.0_101]
```

Exception in the Talend Runtime log meaning the keystore password you defined in the `elasticsearch.ssl.truststore.password` property is not correct.

```
javax.net.ssl.SSLPeerUnverifiedException: Host name 'localhost' does not match the
certificate subject provided by the peer (CN=localhost, OU=ESB, O=Talend, L=Beijing,
ST=Beijing, C=CN)
    at
    org.apache.http.conn.ssl.SSLConnectionSocketFactory.verifyHostname(SSLConnectionSocketFactory.java:465)
[274:org.apache.httpcomponents.httpclient:4.5.2]
```

Exception in the Talend Runtime log meaning the Elasticsearch hostname you provided has failed verification. Input the correct Elasticsearch hostname or type in the value "any" in the `elasticsearch.ssl.hostname` property.

## 4.9. Logging page in Talend Administration Center

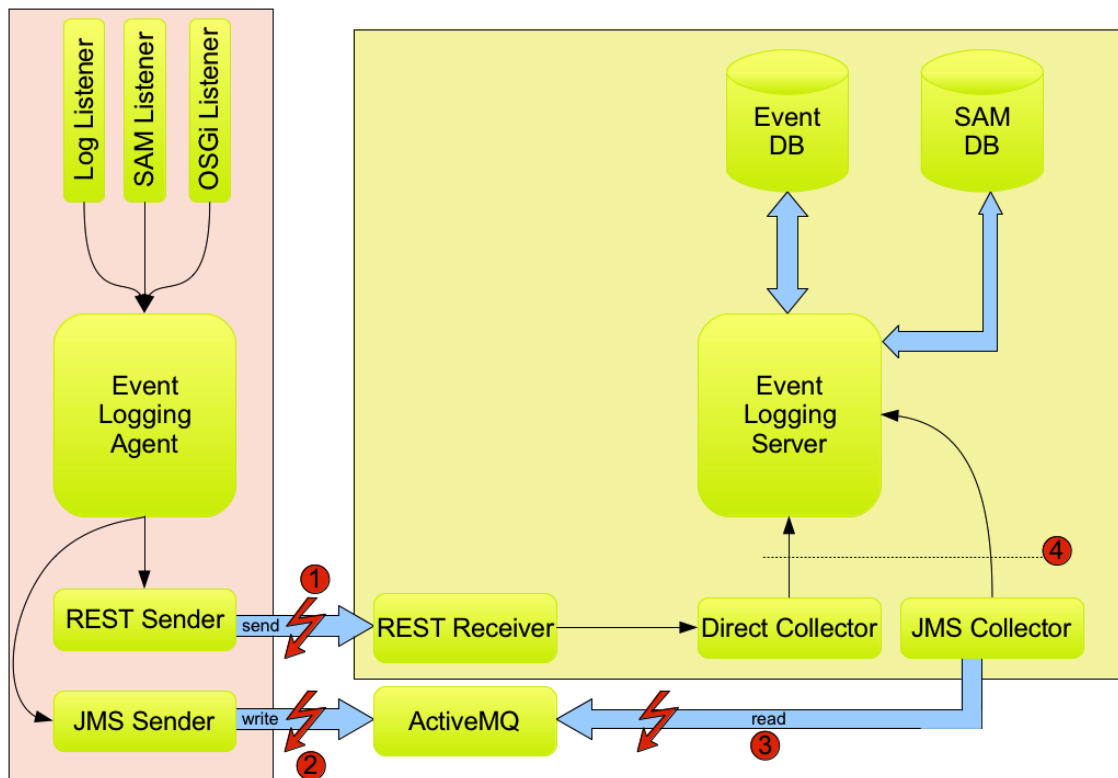
The Event Logging Web-based **Logging** page in Talend Administration Center is based on Kibana 3 (with Elasticsearch). Kibana is an open source (Apache licensed) browser-based analytics and search interface to Log data and other time-stamped data sets stored in Elasticsearch. Its point-and-click composition lets users easily design custom dashboards.

The **Logging** page allows users to explore and make use of a large amount of log data by optimized search and visualization support.

The use of the **Logging** page is optional and Kibana, on which it is based, will be deployed within a Web server.

## 4.10. Robust event processing

The primary focus of the Robust event processing is to make sure none of the events are lost or skipped in case the REST or the JMS brokers are down, or the JMS collector is not able to deliver events to the Event Logging server. As shown in the image below, these are the four critical paths where robust event processing is currently implemented.



To start the robust event processing feature, use the following command in the container:

```
tesb:start-el-dlq
```

This command will start a camel route which handles the robust re-delivery of messages to the Event Logging server. In case the REST receiver or the JMS broker are down, the REST and JMS senders will send the events to a DLQ (Dead letter Queue) configured in the `org.talend.eventlogging.dlq.cfg` file. The Robust processing camel route will monitor this DLQ.

Once the events arrive in DLQ, it will ping the JMS broker to check for its availability.

Once the broker is available, it will send the events from the DLQ to the JMS broker.

The configuration for this is as follows:

```
# JMS connection url
dlq.buffer.jms.url=vm://eventloggingbroker?create=true&broker.use
Jmx=false&broker.persistent=true
# Name of JMS queue
dlq.buffer.jms.queue=event.logging.dlq
# JMS Username
dlq.buffer.jms.username=tadmin
# JMS Password
dlq.buffer.jms.password=tadmin

dlq.timeout=10000

dlq.destination.jms.url=tcp://localhost:61616
dlq.destination.jms.queue=event.logging.server
dlq.destination.jms.username=tadmin
dlq.destination.jms.password=tadmin
```

As shown in the above image at Step 3, if the events arrive in the JMS broker but the JMS Collector is down, or for some reason not able to pick the events, the events will be stored in the JMS broker until the JMS collector picks them up.

In Step 4, if the JMS Collector is not able to send events to the Event Logging server, it will send the events to a DLQ configured in the `org.talend.eventlogging.dlq.cfg` file.

The events will be stored in this DLQ permanently and will not be processed by the JMS Collector or any other component of the Event Logging, or will not be send to the Event Logging server later. It will be the responsibility of the Administrator to manually handle them.

The detailed configuration for the Robust event processing feature is as follows:

# JMS connection url <code>dlq.buffer.jms.url=vm://eventloggingbroker?create=true&amp;broker.useJmx=false&amp;broker.persistent=true</code>	The address of the JMS broker which is used to store the events in the DLQ. The default configuration uses the embedded broker which is also used for internal event jms buffering at the agent side.
# Name of JMS queue <code>dlq.buffer.jms.queue=event.logging.dlq</code>	DLQ queue name
# JMS Username <code>dlq.buffer.jms.username=tadmin</code> # JMS Password <code>dlq.buffer.jms.password=tadmin</code>	DLQ queue Active MQ broker credentials
<code>dlq.timeout=60000</code>	Timeout in milliseconds in which the robust event processing feature will re-deliver events from DLQ queue to destination.
<code>dlq.destination.jms.url=tcp://localhost:61616</code>	Destination (Event Logging server) queue broker address
<code>dlq.destination.jms.queue=event.logging.server</code>	Destination (Event Logging server) queue name
<code>dlq.destination.jms.username=tadmin</code> <code>dlq.destination.jms.password=tadmin</code>	Destination (Event Logging server) broker credentials

## 4.11. Event Logging - API's and Data Structures

### 4.11.1. Event Data - Structure

The event structure is the primary data structure used by all components of the Event Logging feature and consists of the following attributes. The table also shows how the event structure is represented in the Apache Camel Exchange structure (header and body attributes).

Attribute Name	Type	Camel Exchange Mapping	Description
eventUUID	String	header.eventuuid	Unique ID associated with every event.
category	String	header.category	<p>Each event can be assigned to a particular category. Category will help to group and organize events which can be further searched and filtered.</p> <p><b>default</b>, <b>audit</b> and <b>ping</b> are reserved categories and cannot be used as category names.</p> <p>Categories that can be used to begin with might be <i>system</i>, <i>osgi</i>, <i>service</i>, <i>route</i>, <i>bam</i>, <i>sam</i> and <i>security</i>. But the user can easily define and use additional categories. New category names should comply the following: A-Z, a-z, 0-9 and '_' characters are supported, and only maximum 32 characters are allowed.</p>

Attribute Name	Type	Camel Exchange Mapping	Description
eventtype	String	header.eventtype	Attribute to identify the listener from which this event was captured. Pax listener for LOG Event, OSGI Listener for OSGI Event, and SAML Listener for SAM Event.
logmessage	String	body	The log message associated with this event.
severity	String	header.severity	Describes the severity of the event. For example, the log4j log levels.
logsource	Map	header.source	<p>The logsource is collection of data which identifies the source of the generated event. For example :</p> <pre> "logSource":{   "bundle.id": "124",   "bundle.name":     "org.apache.cxf.cxf-rt-management",   "bundle.version": "2.7.7",   "class.name": "org.apache.cxf.manage ment.jmx.InstrumentationManagerImpl",   "file.name":     "InstrumentationManagerImpl.java",   "host.name": "sopera",   "line.number": "329",   "logger.name": "org.apache.cxf.manag ement.jmx.InstrumentationManagerImpl",   "method.name":     "registerMBeanWithServer",   "process.id": "6468" } </pre>
agenttimestamp	Date	header.agenttimestamp	<p>The Timestamp applied on the event when the Event Logging Agent receives it.</p> <p>The timestamp is the date and time of the local machine, converted to UTC date / time with milliseconds and the local timezone of the machine. For example: 2013-07-23T08:45:30.453Z</p>
servertimestamp	Date	header.servertimestamp	The Timestamp applied on the event when the Event Logging Collector Service receives it. The timestamp is the date and time of the server machine, converted to UTC date / time with milliseconds and the timezone of the server. For example: 2013-07-23T08:45:30.453Z
logtimestamp	Date	header.logtimestamp	The Timestamp of the log when it was created. The format of the timestamp depends completely on the logging framework. It will not be converted or transformed by the agent. If no timestamp exists in the log, this attribute will be empty.
audit	boolean	header.audit	A flag to indicate if the event needs to be audited. The default value of this attribute is false.
agentid	String	header.agentid	The local agent ID which handled this event. Every agent should have a unique agent ID.
auditsequenceno	long	header.auditsequenceno	The sequence number is continuously incrementing by +1 on the agent for each new Audit Event, and only for Audit events. This prevents audit events from being repeated later in the EventLogging Database.
signedlogmessage	String	header.signedlogmessage	The log message in XML format including the signature information.
correlationid	String	header.correlationid	An ID used to correlate different events: the Business Correlation ID. In case of SAM Event, the technical FlowID will be stored as <i>eventlog_custominfo key / value</i> .
subject	String	header.subject	The User which created this event.
custominfo	Map	header.custominfo	<p>It is a collection of key / value pairs. This attribute can be used to propagate custom information. For example:</p> <pre> "customInfo":{   "activemq.broker": "eventloggingbrok er" } </pre>

## 4.11.2. Event Database Structure

The Event Logging Database schema consists of two primary tables:

- EVENTLOG
- EVENTLOG\_METADATA

For the complete list of **Compatible Databases**, see the *Talend Installation Guide*.

To create those tables, SQL scripts are provided to you for the supported databases.

1. Make sure your chosen database is installed properly and is accessible.
2. Log in with a user account with CREATE permissions and run the "init SQL" scripts for the corresponding database (see table below). There are two initial scripts for each database. The script with "\_ind" suffix is used to create indexes in database.

The script files for the corresponding databases are described in the following table. The SQL scripts can be found in the <TalendRuntimePath>/add-ons/event-logging/db directory.

SQL script filename	Database
create.sql	Apache Derby
create_ind.sql	
create_mysql.sql	MySQL
create_mysql_ind.sql	
create_oracle.sql	Oracle
create_oracle_ind.sql	
create_sqlserver.sql	SQL Server
create_sqlserver_ind.sql	
create_h2.sql	H2 Database Engine
create_h2_ind.sql	
create_db2.sql	IBM DB2
create_db2_ind.sql	
create_postgresql.sql	PostgreSQL
create_postgresql_ind.sql	

Once the scripts executed, the EVENTLOG and EVENTLOG\_METADATA tables are created in your database. Below is the data structure of those tables:

**For the EVENTLOG table:**

Field	Type
ID(Primary Key)	BIGINT
EVENT_UUID	CHAR(36)
CATEGORY	varchar(255)
EVENT_TYPE	varchar(255)
LOG_MESSAGE	CLOB(2147483647)
SEVERITY	varchar(255)
LOG_SOURCE	varchar(4000)
AGENT_TIMESTAMP	TIMESTAMP



Field	Type
SERVER_TIMESTAMP	TIMESTAMP
LOG_TIMESTAMP	TIMESTAMP
AUDIT	char(1)
AGENT_ID	varchar(255)
AUDIT_SEQUENCE_NO	varchar(255)
SIGNED_LOG_MESSAGE	BLOB(2147483647)
CORRELATION_ID	varchar(255)
SUBJECT	varchar(255)

**For the EVENTLOG\_METADATA table:**

Field	Type	Description
ID(Primary Key)	BIGINT	Unique id of the CustomInfo
EVENT_ID	BIGINT	Relative EVENT's ID value
METADATA_TYPE	varchar(255)	Metadata type
METADATA_KEY	varchar(255)	Metadata key
METADATA_VALUE	varchar(4000)	Metadata value

## 4.11.3. Event Logging Service API

### Resource and URI:

This section describes the Event Logging REST Service resources and URI. The base URI for the service will be:

```
http://{hostname}:{port}/services/eventlogging/
```

### GET /

Resource to check if Event Logging REST Service is online. On success, it will return an HTTP code 200.

Example request:

```
GET http://{hostname}:{port}/services/eventlogging/
```

### GET /events/{eventUUID}

It returns an event with the given uuid.

Example request:

```
GET http://{hostname}:{port}/services/eventlogging/events/fe5338b4-fc8a-451e-9d28-33c73cd1d828
```

Request Body:

```
{
  "eventUUID": "392c775b-8072-45b2-bf6b-falffb1ffc6c",
  "category": "system",
  "eventType": "LOGEvent",
  "severity": "INFO",
  "logMessage": "Total 3 routes, of which 3 is started.",
  "logSource": {
    "bundle.id": "170",
    "bundle.name": "org.apache.camel.camel-core",
    "bundle.version": "2.12.1",
    "class.name": "org.apache.camel.impl.DefaultCamelContext",
  }
}
```

```

    "file.name": "DefaultCamelContext.java",
    "host.name": "sopera",
    "line.number": "1533",
    "logger.name": "org.apache.camel.blueprint.BlueprintCamelContext",
    "method.name": "start",
    "process.id": "6468"
  },
  "logTimestamp": "2013-11-13T09:13:58.126+0000",
  "agentId": "agent1",
  "agentTimestamp": "2013-11-13T09:13:58.134+0000",
  "serverTimestamp": "2013-11-13T09:14:59.187+0000",
  "audit": false,
  "customInfo": {
    "activemq.broker": "eventloggingbroker"
  }
}

```

### GET /events/{eventUUID}/signature

If not empty, this request returns a signedlogmessage event attribute content for the event with the given uuid (response content-type: application/xml). If empty, you will get a 204 No content HTTP response.

Example request:

```

GET http://{hostname}:{port}/services/eventlogging/events/149edf25-7f94-4
90a-bc07-4fcb860cb9fe/signature

```

### GET /events?

It returns a collection of relevant events matching a specified search query. The search query supports FIQL (Feed Item Query Language) syntax for simple data types. FIQL provides a way to express complex search expressions using an intuitive and URI friendly language.

Currently, only the following FIQL operators are supported:

Operator	Description
Operator	Description
"=="	Equal
","	AND
" "	OR
"=lt="	Less Than
"=le="	Less or Equal
"=gt="	Greater Than
"=ge="	Greater or Equal

### Search parameters:

category	Specifies the category of the event to be searched.
optional	Example value: security
severity	Specifies the severity of the event to be searched.
optional	Example value: fatal
eventtype	Specifies the type of the event to be searched.
optional	Example values: LOGEvent, OSGiEvent, SAMEvent.
correlationid	Specifies the correlation ID of the event to be searched.
optional	Example value: 21760804-4961
subject	Specifies the subject associated with the event to be searched.
optional	Example value: Alice

agentid	Specifies the agent ID which is associated with the event.
optional	Example value: Agent3455
agenttimestamp	Returns all the events matching the given agent timestamp. Date should be formatted as UTC time format: YYYY-MM-DDThh:mm:ss.sTZD.
optional	Example value: 2013-10-10T12:22:06.060+0000
servertimestamp	Returns all the events matching the given server timestamp. Date should be formatted as UTC time format: YYYY-MM-DDThh:mm:ss.sTZD.
optional	Example value: 2013-10-10T12:22:06.060+0000
audit	Specified to return the events needs to be audited or not be audited.
optional	Example value: true/false
auditsequenceno	Specifies the auditsequenceno of the event to be searched.
optional	Example value: 1234

Examples of search query:

- `/events?_s=category==security;severity==ERROR`

The above search query will return all the events of the *security* category and *ERROR* severity.

- `/events?_s=category==security;(severity==ERROR,severity==WARN)`

The above search query will return all the events of the *security* category and with either *ERROR* or *WARN* severity.

- `/events?_s=category==system;agenttimestamp=ge=2013-10-10T12:22:06.060+0000;agenttimestamp=le=2013-10-10T12:22:06.076+0000`

The above search query will return all the events of the *system* category and the agenttimestamp greater than or equal to *2013-10-10T12:22:06.060+0000* and less than or equal to *2013-10-10T12:22:06.076+0000*.



FIQL queries must be URL encoded. This means, in particular, that if you are using FIQL with a Web browser, use "%2B" instead of "+" in date format.

For example: *2013-10-10T12:22:06.060%2B0000*

It is also possible to search on complexe log event data types like *logSource* and *customInfo*. However, only the equal "=" operations are supported for complex data types.

The following syntax can be used to define a filter for a complex event data type:

- `logsource.<key>=<value>`
- `custominfo.<key>=<value>`

Examples of complex data type search query:

- `/events?logsource.host.name=myserver`

The above search query will return all the events from a computer with the hostname "myserver".

- `/events?custominfo.mykey=myValue&logsource.file.name=LogEventHigh.java`

The above search query will return all the events that contain a "mykey" parameter of value "myValue" in its *customInfo* field (MDC property) and come from the "LogEventHigh.java" file.

It is also possible to combine FIQL search queries for simple data types with search parameter for complex data types.

Examples of combined search query:

```
/events?logsource.bundle.name=myservice&_s=audit==true;auditsequenceno=gt=5
```

The above search query will return all audit events from a bundle named "myservice" where the *auditsequenceno* is greater than 5.

### Controlling the response

The response of the search query can be controlled with the following parameters:

limit optional	Limits the result set to the first "n" number of rows (always ordered by agenttimestamp descending).  Example value: 100
include_logmessage optional	Specifies if the log message needs to be included in the returned result of events.  Example value: true/false
include_signedlogmessage optional	Specifies if the signed log message needs to be included in the returned result of events.  Example value: true/false
include_logsource optional	Specifies if the log source needs to be included in the returned result of events.  Example value: true/false
include_custominfo optional	Specifies if the custom info properties needs to be included in the returned result of events.  Example value: true/false
include_all optional	Specifies if all extra properties of the event described by the above mentioned include_* parameters needs to be included in the returned result of events.  Example value: true/false

Example Request:

```
GET /events?_s=category==system&count=2&includecustominfo=true
```

Request Body:

```
{
  "events": [
    {
      "eventUUID": "ad082036-a873-49dd-8fd8-f5f75a1a6763",
      "category": "system",
      "eventType": "LOGEvent",
      "severity": "INFO",
      "logMessage": "Route: route32 started and consuming from: Endpoint[paxlogging://eventloglisteneraudit]",
      "logSource": {
        "bundle.id": "170",
        "bundle.name": "org.apache.camel.camel-core",
        "bundle.version": "2.12.1",
        "class.name": "org.apache.camel.impl.DefaultCamelContext",
        "file.name": "DefaultCamelContext.java",
        "host.name": "sopera",
        "line.number": "2183",
        "logger.name": "org.apache.camel.blueprint.BlueprintCamelContext",
        "method.name": "doStartOrResumeRouteConsumers",
        "process.id": "6468"
      },
      "logTimestamp": "2013-11-13T09:13:58.123+0000",
    }
  ]
}
```

```

    "agentId": "agent1",
    "agentTimestamp": "2013-11-13T09:13:58.131+0000",
    "serverTimestamp": "2013-11-13T09:14:59.186+0000",
    "audit": false,
    "customInfo": {
      "activemq.broker": "eventloggingbroker"
    },
    {
      "eventUUID": "f75ae2a7-6cbc-4213-946a-a43cb62d7f70",
      "category": "system",
      "eventType": "LOGEvent",
      "severity": "WARN",
      "logMessage": "Can't find the the request for https://localhost:9001/services/XacmlRegistryAtom's Observer ",
      "logSource": {
        "bundle.id": "130",
        "bundle.name": "org.apache.cxf.cxf-rt-transport-http",
        "bundle.version": "2.7.7",
        "class.name": "org.apache.cxf.transport.servlet.ServletController",
        "file.name": "ServletController.java",
        "host.name": "sopera",
        "line.number": "175",
        "logger.name": "org.apache.cxf.transport.servlet.ServletController",
        "method.name": "invoke",
        "process.id": "6468"
      },
      "logTimestamp": "2013-11-13T09:17:55.894+0000",
      "agentId": "agent1",
      "agentTimestamp": "2013-11-13T09:17:55.896+0000",
      "serverTimestamp": "2013-11-13T09:18:56.473+0000",
      "audit": false,
      "customInfo": {}
    }
  ],
  "searchMetadata": {
    "count": 2,
    "totalCount": 83
  }
}

```

## POST /events

Adds a single or a collection of events to the Event Logging backend. On success, the resource invocation will result into HTTP code 204.

### Parameters

The following attributes in the event/events object should not be empty. The other attributes defined in the event structure above can be empty.

Attribute Name
id
category
agenttimestamp
agentid
auditsequenceno (required in case if it is an audit event)

Example request:

```
POST http://{hostname}:{port}/services/eventlogging/events/Content-Type:
application/json
```

Request Body:

```
[
  {
    "eventUUID": "ad082036-a873-49dd-8fd8-f5f75a1a6763",
    "category": "system",
    "eventType": "LOGEvent",
    "severity": "INFO",
    "logMessage": "Route: route32 started and consuming from: Endpoint[paxlogg
ing://eventloglisteneraudit]",
    "logSource": {
      "bundle.id": "170",
      "bundle.name": "org.apache.camel.camel-core",
      "bundle.version": "2.12.1",
      "class.name": "org.apache.camel.impl.DefaultCamelContext",
      "file.name": "DefaultCamelContext.java",
      "host.name": "sopera",
      "line.number": "2183",
      "logger.name": "org.apache.camel.blueprint.BlueprintCamelContext",
      "method.name": "doStartOrResumeRouteConsumers",
      "process.id": "6468"
    },
    "logTimestamp": "2013-11-13T09:13:58.123+0000",
    "agentId": "agent1",
    "agentTimestamp": "2013-11-13T09:13:58.131+0000",
    "serverTimestamp": "2013-11-13T09:14:59.186+0000",
    "audit": false,
    "customInfo": {
      "activemq.broker": "eventloggingbroker"
    }
  },
  {
    "eventUUID": "f75ae2a7-6cbc-4213-946a-a43cb62d7f70",
    "category": "system",
    "eventType": "LOGEvent",
    "severity": "WARN",
    "logMessage": "Can't find the the request for https://localhost:9001/servi
ces/XacmlRegistryAtom's Observer ",
    "logSource": {
      "bundle.id": "130",
      "bundle.name": "org.apache.cxf.cxf-rt-transport-http",
      "bundle.version": "2.7.7",
      "class.name": "org.apache.cxf.transport.servlet.ServletController",
      "file.name": "ServletController.java",
      "host.name": "sopera",
      "line.number": "175",
      "logger.name": "org.apache.cxf.transport.servlet.ServletController",
      "method.name": "invoke",
      "process.id": "6468"
    },
    "logTimestamp": "2013-11-13T09:17:55.894+0000",
    "agentId": "agent1",
    "agentTimestamp": "2013-11-13T09:17:55.896+0000",
    "serverTimestamp": "2013-11-13T09:18:56.473+0000",
    "audit": false,
    "customInfo": {}
  }
]
```



## Chapter 5. Service Registry

The Service Registry provides a repository for storing service WSDL and WS-Policy files. This product is available with Talend ESB; it is not included in the Talend Open Studio for ESB.

The Service Registry helps maintain consistency for your services and their Policy-based security and reliability requirements. The Service Registry itself is part of Talend Runtime, while provided interceptors are provided to clients (whether SOAP clients or Talend Runtime-hosted web service providers) to access the registry.

## 5.1. Introduction

Storing non-functional aspects of services such as security and reliability policies within each Container, or even within each application, poses several maintenance challenges:

- Distributing several copies of one and the same artifact, such as service descriptions and policies, over different runtimes.
- Enabling reuse of artifacts, especially policy assertions, across services.
- Keeping a consistent version of configurations and policies over all runtimes.
- Getting an overview of current settings in the different runtimes.
- Enforcing consistent authorization policies for the changing artifacts in the runtimes.

For larger deployments, a central registry becomes an increasingly important component. The purpose of the registry is to serve as a runtime directory advertising service descriptions, policies, and configuration. To manage the artifacts, the registry provides an administrative user interface which also enables browsing through the artifacts. For a lookup of the artifacts required by the runtime, the registry provides a corresponding service for the different artifact types. Instead of a pull style service, a push style may be offered to simplify updates of artifacts when the runtime is already active.

With a central registry being in place, a security architect for example would now be able to specify standard policy assertions for the different security aspects, authentication, signatures, and so on, and deploy them to the registry. The security assertions can then be reused to define a uniform policy for multiple services ensuring that security aspects are applied consistently.

Common use cases supported by the Service Registry:

- **Service Lookup** - When a CXF service or a client is created, it needs to retrieve the corresponding WSDL, either to make it accessible using HTTP GET requests via the '?wsdl' mechanism or to generate a suitable client proxy. Instead of retrieving it from a fixed location, the service or client will now look up the description from the registry. To lookup a service description, the component (service or client) needs to specify the fully qualified name of the service. In case of success, the registry returns a WSDL containing the service with the requested name, otherwise an error is returned.
- **Initial Policy Lookup** - When a CXF service or a client is created and the WSDL already retrieved, the component next needs to know what policies are to be applied for the service. Besides applying policies embedded in the service description or referenced from within the service description, the component needs to consult potential policy attachment documents bound to the service description. Instead of retrieving the policy attachment documents from a fixed location configured locally, the component looks up the policy attachment from the registry. To lookup a policy attachment, the component needs to specify the fully qualified name of the service for which to get the policy attachment. In case of success, the registry returns a policy attachment and the component further follows the references in the attachment, otherwise the registry returns an empty response.

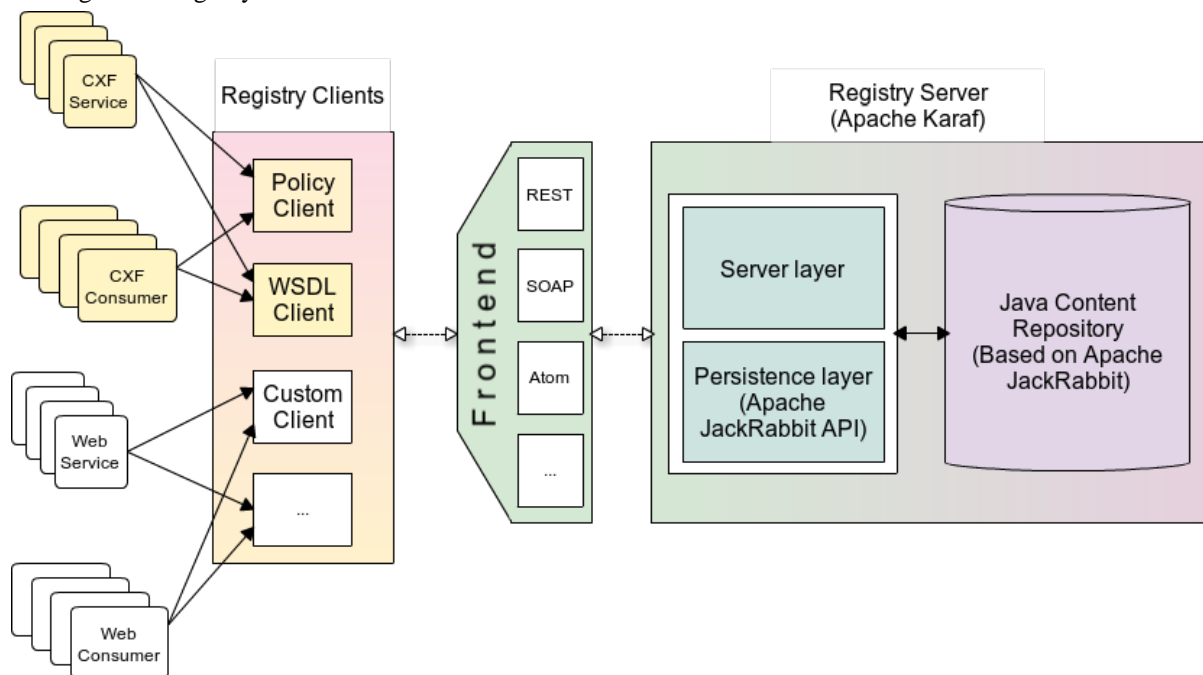
The list below describes the domain model for the Service Registry in order to provide a structural view of the system:

- **Registry** - a component that provides centralized storage for service metadata (WSDL, policy, policy attachment) and a possibility to work with it (upload, update, delete, lookup). It contains three layers: Persistence layer, Server layer, and Service layer.
- **Registry client** - a component that can use the Registry Service to request resources from Registry storage. There are two registry clients that can be used in TESB Runtime to provide metadata for CXF services: WSDL Registry client and Policy Registry client. Also browser, soapUI or any custom clients can be used to work with Registry.
- **Resource** - an entity that can be stored in the Registry.



- **Service metadata** - Carries customer-specific information related to the stored services, usually targeted rather to human readers than to machines. This information may cover aspects of service development, business aspects of service provision and consuming, or similar issues. The Service Registry expects XML documents as metadata resources where the root element, the `ServiceName` element, and the `metadata` element are defined, but not the content of the `metadata` element. It is up to you to provide a schema (XSD) which is used by the Service Registry to validate the metadata documents. Service metadata can be fed into ElasticSearch for full-text search and other advanced queries.
- **Service Description** - Specifies the interface of the service and corresponding data structures. Represented by WSDL and XML Schema documents.
- **WSDL** - a Service Description Resource providing the operations exposed by a web service provider, as well as binding and endpoint information.
- **Policy** - a Resource providing assertions about non-functional characteristics of a service, such as security aspects.
- **Policy Attachment** - A Resource used to bind policies to a service description.
- **Persistence layer** - a functional layer that consists of JCR (Java Content Repository), or more precisely Apache Jackrabbit, and wrapped Jackrabbit API for convenient usage of repository (PersistenceManagerFactory and PersistenceManager classes).
- **Server layer** - a functional layer that operates with higher-level abstractions than Persistence layer. The main abstractions on this level are resource, WSDL, policy and operations to work with these entities: upload, update, delete, lookup. This layer is presented by RegistryServer class.
- **Service layer** - a Registry frontend that can be presented by different service types (REST, SOAP, and so on). By default TESB Registry uses a REST service to expose the Registry API.

The Talend Service Registry consists of server and registry clients that can be used by CXF-based web-services to retrieve resources from the repository (for example: WSDL, policies, and so on). Please see the graphic model of the general Registry structure below:



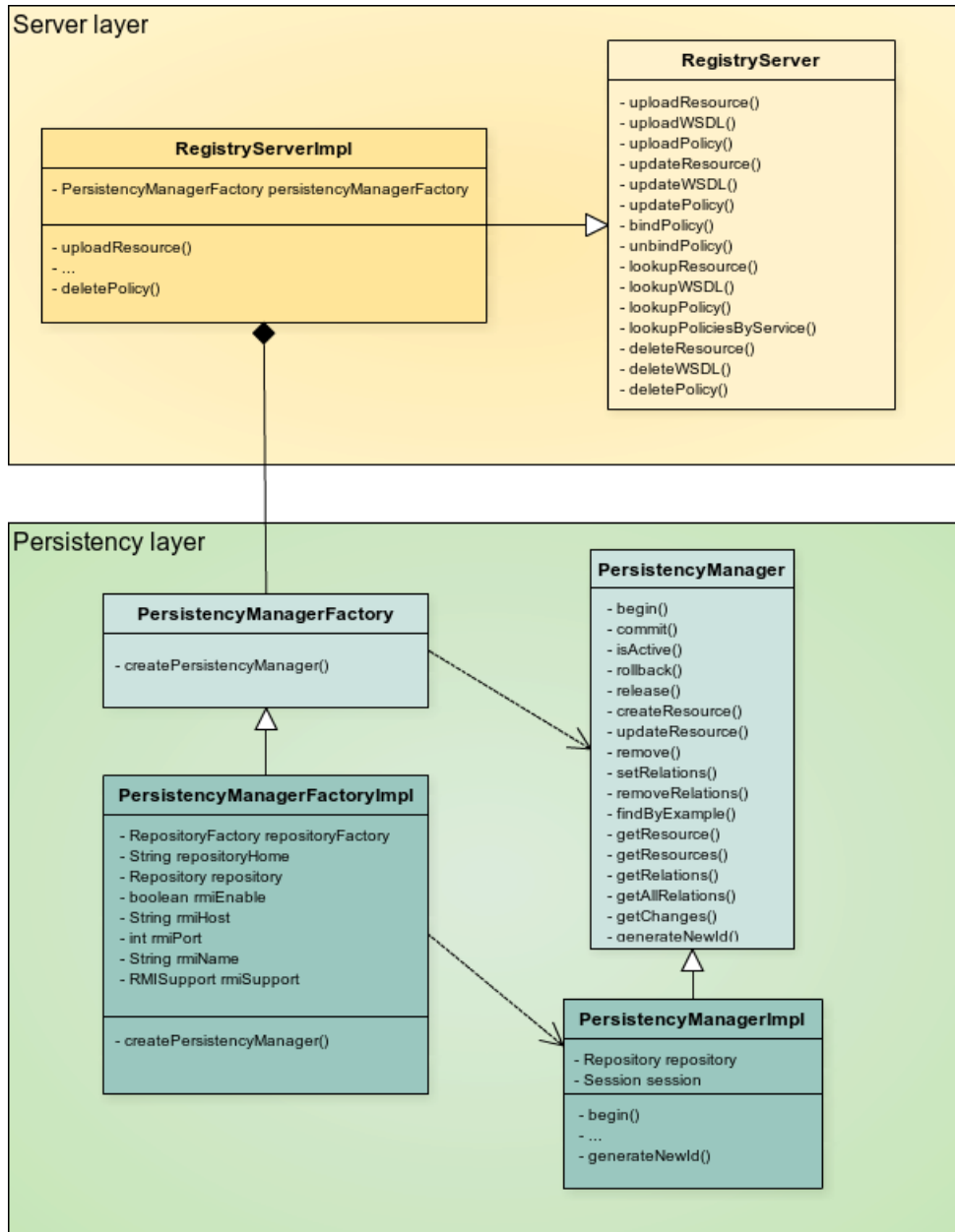
The Registry clients shown above are used to gain and process registry data. Two kinds of clients are provided as features in the Talend Runtime container: **WSDL Client** and **Policy Client**. These clients allow CXF services to dynamically change their WSDLs and policies, avoiding a need to manually alter the services. User can also

implement custom clients to work with one of existing frontends of the registry. Also SOAP UI or a browser can be used to make various manipulations with registry data.

The main components of the Server part of the Registry are:

- Java Content Repository (JCR), using the Apache Jackrabbit repository to store registry data.
- Backend, which has two layers: persistence and server layer, both of which have APIs to work with them.
- Frontend, that can be REST, SOAP or some other kind of service, which will expose the registry API to the clients.

The registry backend is kept independent of the frontend. The next picture shows this in more detail:



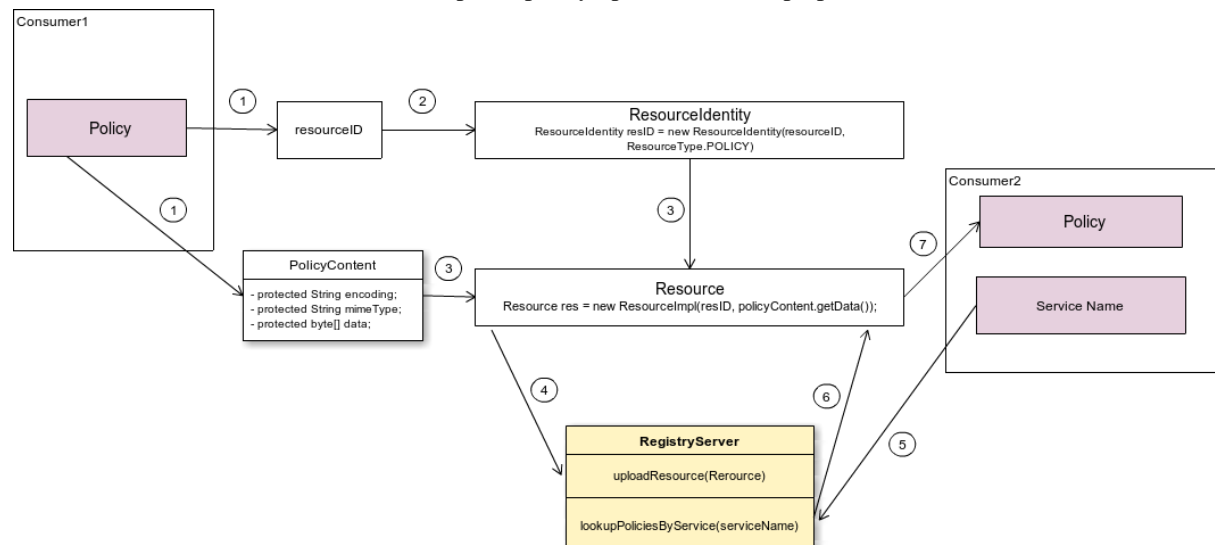
The Persistence layer API is a wrapper for the Jackrabbit API to manipulate resources in form of repository nodes, properties and relations between them (in JSR notions). The Server layer API provides the possibility to manipulate WSDLs and policy files.

The main interface of the domain model is **RegistryServer**. It contains a set of operations that can be used to manipulate Resources. Every resource has a **ResourceIdentity** and **Content** (for example, **PolicyContent** in

case of policies). ResourceIdentity can be of one predefined ResourceTypes and contains a unique resourceID. RegistryServer provides just a contract for registry operations and these operations can be implemented using various technologies. For example, instead of the JCR repository, a database can be used to store registry data. In this case, an implementation that works with database API should be provided and this approach ensures independence of domain model from underlying data access layer or backend. For more information on how to change the repository from the default file-based storage to a database-based one, see [Backend configuration](#).

Another advantage of this API is related to the well-defined contract and strict set of entities from the registry domain. Objects specific for some technology or architectural style are not used (for example: REST or SOAP), which provides the possibility to make an independent service layer. The only thing needed is to use the domain model API from whatever specific technology context chosen.

Below is the domain model with an example of policy upload and lookup operations:



As you can see from the image, these operations can be split into several steps.

First, if the consumer (Consumer1) wants to upload the policy to the Registry:

1. It collects information about certain policies (Step 1),
2. It instantiates the ResourceIdentity and Resource classes (Steps 2 and 3),
3. It can use the Resource object to upload a policy by methods from the RegistryServer interface. (Step 4).

Second, if another consumer (Consumer2) tries to look up policies from the Registry:

1. It will also have to use RegistryServer, by setting the service name (Step 5),
2. The Resource object that the consumer gets from the RegistryServer (Step 6), can easily get the policy content to the consumer (Step 7).

No classes specific to a particular frontend interface are used for these operations, keeping this scheme valid for different possible frontend implementations. In addition, by using the RegistryServer interface the domain model does not depend on any specific backend implementation.

## 5.2. Installing the Service Registry

In this section, you will see how to install the Service Registry into the Talend Runtime container and how to install Talend ESB Policies.

## 5.2.1. Activating the Service Registry

There are four available components to the Service Registry, as well as a common bundle for shared functionality.

The simplest way to install them within the Talend Runtime container is via the following command:

```
tesb:start-registry
```

They can also be uninstalled via:

```
tesb:stop-registry
```

Alternatively, each individual feature making up the Service Registry can be installed individually with the following commands:

- **feature:install tesb-registry-server**
- **feature:install tesb-registry-rest-service**
- **feature:install tesb-registry-rest-atom-service**
- **feature:install tesb-registry-server-commands**

Once installed, run the following command to show the activated features:

```
list | grep Registry
```

The installed features displayed should be as follows:

237	Active	80	6.4.1	Talend ESB Registry :: Client :: Common
238	Active	80	6.4.1	Talend ESB Registry :: Common
239	Active	80	6.4.1	Talend ESB Registry :: Client :: Policy
240	Active	80	6.4.1	Talend ESB Registry :: REST :: Security
241	Active	80	6.4.1	Talend ESB Registry :: Client :: WSDL
274	Active	80	6.4.1	Talend ESB Registry :: REST Atom Service
275	Active	80	6.4.1	Talend ESB Registry :: REST Lookup Service
276	Active	80	6.4.1	Talend ESB Registry :: Server
277	Active	80	6.4.1	Talend ESB Registry :: Server :: Commands

You can configure the Service Registry in the `etc/org.talend.esb.registry.server.cfg` file, with the following parameters. Note that the below values provided by default for those parameters are usually suitable.

**Table 5.1. Service Registry Configuration Settings**

Option	Description
<code>repository.home</code>	Jackrabbit repository home directory
<code>rmi.enable</code>	Whether to enable RMI access to Jackrabbit repository
<code>rmi.host</code>	The localhost interface for the RMI registry
<code>rmi.port</code>	The port on which the RMI registry is listening
<code>rmi.name</code>	The name to which the repository is to be bound in the registry
<code>checker.wsdl.enableWSIcheck</code>	Whether to enable the WS-I consistency check for WSDL resources

The Talend Service Registry service is exposed via the container HTTP(s) port which can be configured in the `org.ops4j.pax.web.cfg`. For more information, see the **HTTP Configuration** chapter in the *Talend ESB Container Administration Guide*.

The Registry WSDL client can be installed via the following commands within the Talend Runtime container:

```
feature:install tesb-registry-wsdl-client
```

The Registry Policy client can be installed via the following commands within the Talend Runtime container:

```
feature:install tesb-registry-policy-client
```

Once installed, run the activated features as follows:

```
list | grep Registry
```

The installed features displayed should be as follows:

```
[235] [Created] [80] Talend ESB Registry :: Client :: Policy
[236] [Created] [80] Talend ESB Registry :: Client :: WSDL
```

The WSDL client configuration is done in the `etc/org.talend.esb.registry.client.wsdl.cfg` and the Policy client configuration is done in the `etc/org.talend.esb.registry.client.policy.cfg`.

To use the Service Registry with SSL, change the `registry.url` parameter value from `http://localhost:8040/services/registry/lookup` to `https://localhost:9001/services/registry/lookup`.

The Service Registry WSDL and Policy clients support two authentication methods: BASIC and SAML. You can use BASIC or SAML authentication by enabling the corresponding settings or no authentication by enabling `registry.authentication = NO` in the configuration file.

To use BASIC authentication, enable the following settings. The user credentials can be found in `etc/users.properties`.

```
#BASIC authentication
registry.authentication.user = tesb
registry.authentication.password = tesb
```

To use SAML authentication, enable the following settings. Change the WS-Security and STS Client configuration according to your own use case. For more information, see the *Talend ESB Service Developer Guide* and *Talend ESB STS User Guide*.

```
#SAML authentication
security.username = tadmin
security.password = tadmin
security.sts.token.username = myclientkey
security.sts.token.properties = clientKeystore.properties
security.encryption.username = mystskey
security.encryption.properties = clientKeystore.properties

sts.wsdl.location = http://localhost:8040/services/SecurityTokenService/UT?wsdl
sts.namespace = http://docs.oasis-open.org/ws-sx/ws-trust/200512/
sts.service.name = SecurityTokenService
sts.endpoint.name = UT_Port
```

## 5.3. ESB Policies

Talend offers different standard and custom policies that can be used within the Talend ESB.

The standard WS-Security policies are the following ones:

- [Authentication](#)
- [Signing/Encryption](#)

The custom policies are the following:

- [Correlation ID Policy](#)
- [WSDL Schema Validation Policy](#)

- [Custom Schema Validation Policy](#)
- [Transformation Policy](#)
- [Compression Policy](#)
- [Service Activity Monitoring Policy](#)
- [Authorization](#)

However, those policies are only available in the Talend Enterprise and Talend Platform products. They can be found under the `add-ons\registry\policies` directory of the Talend ESB delivery, and can be uploaded to the Service Registry via the Talend Administration Center Web User Interface (for more information, see the *Talend Administration Center User Guide*), or via Service Registry commands directly from a Talend ESB Container (for more information, see [Using the Service Registry with Talend ESB](#)).

A set of default policy templates can be imported at once into the Talend Administration Center Service Registry by importing the `add-ons\registry\policies\tesb_template_policies.xml` file.

In addition to the Talend ESB system policies and the standard WS-Security policies supported by Talend ESB documented in the following sections, Talend ESB supports standard WS-Addressing policies. For more information on the WS-Addressing policies, please see the related standards.

## 5.3.1. WS-Security policies

Talend ESB supports the following standard WS-Security policies:

- [Authentication](#) via either UsernameToken or SAMLToken.
- [Signing/Encryption](#).

### 5.3.1.1. Authentication

#### UsernameToken

The UsernameToken authentication policy adds a WS-Security UsernameToken to the SOAP header of the request on the client side. On the service side, it enforces that a user has authenticated by sending a UsernameToken in the request. The UsernameToken contains a username and password.

Talend ESB provides a template policy called `wspolicy_authn_usernameToken.policy` and available in the `/add-ons/registry/policies` folder of the Talend ESB product.

It is also applied by default to your Talend ESB Container via the following policy file: `etc/org.talend.esb.job.token.policy`. So if you selected the UsernameToken for your Service in the Studio, when you deploy it on your container, this policy will be picked up automatically.

#### SAMLToken

The SAMLToken policy specifies a WS-SecurityPolicy Asymmetric Binding. This means that the client must secure the request using asymmetric keys (a private key for signature, and a public key for encryption). In addition, the "token" required for signature is an IssuedToken policy, which means that the client must contact the STS to

get a token (a SAML token according to the policy) and include it in the service request. If the client is signing the request, the client uses the private key associated with the SAML token.

Talend ESB provides a template policy called `wspolicy_authn_saml.policy` and available in the `/add-ons/registry/policies` folder of the Talend ESB product.

It is also applied by default to your Talend ESB Container via the following policy file: `etc/org.talend.esb.job.saml.policy`. However, the STS must be started before the client, otherwise the client will not be able to get a SAML token to access the service.

### 5.3.1.2. Signing/Encryption

Signing and Encryption use the same two policies, both available here:

- `/add-ons/registry/policies/wspolicy_authn_saml_crypto.policy`

This policy add the SAML token, and sign and encrypt the SOAP Body.

- `/add-ons/registry/policies/wspolicy_authn_authz_crypto.policy`

Same as the above, but with the authorization policy.

The SOAP Body is signed using the key associated with the SAML Token. The Body is encrypted using a certificate for the service obtained from the XKMS service.

However, some of the policies appear more than once, because in the Studio, you have four different options:

- Username / Password. It maps to the `org.talend.esb.job.token.policy` file.
- SAML Token. It maps to the `org.talend.esb.job.saml.policy` file, if you are not using any authorization or encryption.
- Authorization. It must be used in conjunction with SAML and it maps to the `etc/org.talend.esb.job.saml.authz.policy` file.
- Encryption/Signature body. It must also be used with SAML but it maps to either `org.talend.esb.job.saml.authz.crypto.policy` or `org.talend.esb.job.saml.crypto.policy` depending if authorization is selected or not.

### 5.3.2. Custom policies

In addition to WS-Addressing and WS-Security policies, Talend ESB also provides custom system policies:

- [\*Correlation ID Policy\*](#)
- [\*WSDL Schema Validation Policy\*](#)
- [\*Custom Schema Validation Policy\*](#)
- [\*Transformation Policy\*](#)
- [\*Compression Policy\*](#)
- [\*Service Activity Monitoring Policy\*](#)
- [\*Authorization\*](#)

### 5.3.2.1. Correlation ID Policy

The correlation ID feature provides support for setting a business correlation ID to services.

To allow chained service calls to be grouped under the same ID, you need to introduce Correlation ID as part of the Custom SOAP (HTTP) Header. Using this Correlation ID, it is possible to identify all calls in the chain.

To do so, you can use a custom correlation ID policy to activate the custom correlation ID feature, which is only available for SOAP services, or add the correlation ID feature to the endpoint features list, which is available for both SOAP and REST services.

If the custom correlation ID feature is enabled or present in the SOAP (HTTP) Header, the Service Activity Monitoring agent sets the ID in the custom properties as Correlation ID.

Two types of Correlation ID policy can be enabled via two policies:

- [Enabling the default Correlation ID policy](#)
- [Enabling Correlation ID with XPATH extraction from payload](#)

## Maven project dependency

To use the Correlation ID policy in your project, you have to implement the following dependency:

```
<dependency>
  <groupId>org.talend.esb.policies</groupId>
  <artifactId>correlationid-policy</artifactId>
</dependency>
```

## Enabling the default Correlation ID policy

The default correlation ID feature provides two options:

- Use of the custom correlation ID using a callback.
- Use of the custom correlation ID generated.

For more information on how to use these options, see the procedure below:

1. Make sure the Talend Runtime is running, and the Service Registry service has been started. For more information, see the *Talend ESB Container Administration Guide*.
2. Import the correlation ID policy to the Service Registry, either directly from the Talend Runtime. For more information, see [Using the Service Registry with Talend ESB](#) or via the Talend Administration Center, for more information, see the *Talend Administration Center User Guide*.

Talend ESB provides a template policy called `wspolicy_correlation_id.policy` and available in the `/add-ons/registry/policies` folder of the Talend ESB product.

```
<wsp:Policy Name="wspolicy_schema_correlation_id"
  xmlns:wsp="http://www.w3.org/ns/ws-policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <tpa:CorrelationID
        xmlns:tpa="http://types.talend.com/policy/assertion/1.0"
        type="callback" />
    </wsp:All>
  </wsp:ExactlyOne>
```



```
</wsp:Policy>
```

In this default example, the callback option is used: `type="callback"`.

But if you want to automatically generate the correlation ID, remove the `type="callback"` attribute and value from the policy, then the id will be generated automatically as system UID, and the value will be the same for request and response.

Example of Correlation ID policy without the callback option:

```
<wsp:Policy Name="wspolicy_schema_correlation_id" xmlns:wsp="http://www.w3.org/ns/ws-policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <tpa:CorrelationID xmlns:tpa="http://types.talend.com/policy/assertion/1.0" type="callback"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

3. If you use the callback option, you should specify the correlation ID callback handler that will produce your custom correlation ID:

```
<jaxws:properties>
  <entry key="correlation-id.callback-handler">
    <bean class="common.talend.CorrelationHandler" />
  </entry>
</jaxws:properties>
```

Where `common.talend.CorrelationHandler` is a custom class that implements the `org.talend.esb.policy.correlation.CorrelationIDCallbackHandler` interface. You need to create the class and change the name of the class to your own in the code above appropriately.

4. Assign the policy to the service which you want to enable the Correlation ID feature.

## Enabling Correlation ID with XPATH extraction from payload

The XPATH parser allows to build the correlation ID using JXPath expressions. For more information, see <http://commons.apache.org/proper/commons-jxpath/users-guide.html>.

To enable the Correlation ID policy with XPATH extraction from payload, upload the following XPATH Correlation ID policy to the Service Registry and attach it to a service:

```
<wsp:Policy Name="wspolicy_schema_correlation_id" xmlns:wsp="http://www.w3.org/ns/ws-policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <tpa:CorrelationID xmlns:tpa="http://types.talend.com/policy/assertion/1.0" type="xpath" name="customer">
        <tpa:Namespace prefix="ns2" uri="http://customerservice.example.com/" />
        <tpa:Part name="customerFirstName" xpath="/ns2:getCustomersByName/firstname"/>
        <tpa:Part name="customerLastName" optional="true" xpath="/ns2:getCustomersByName/lastname"/>
      </tpa:CorrelationID>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

When you apply it to the above SOAP message:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

```
<soap:Body>
  <ns2:getCustomersByName xmlns:ns2="http://customerservice.example.com/">
    <firstname>Alfred</firstname>
    <lastname>Nobel</lastname>
  </ns2:getCustomersByName>
</soap:Body>
</soap:Envelope>
```

You get the following SOAP message with the Correlation ID policy attachment:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <correlationId xmlns="http://www.talend.com/esb/sam/correlationId/v1">customer#customerFirstName=Alfred;customerLastName=Nobel</correlationId>
  </soap:Header>
  <soap:Body>
    <ns2:getCustomersByName xmlns:ns2="http://customerservice.example.com/">
      <firstname>Alfred</firstname>
      <lastname>Nobel</lastname>
    </ns2:getCustomersByName>
  </soap:Body>
</soap:Envelope>
```

## XPath-based CorrelationID String Syntax

```
{CorrelationName [Optional]}{Correlation Name Separator [Mandatory if CorrelationName is specified]}{CorrelationPartName[Optional]}{CorrelationPartValueNameSeperator[Mandatory if CorrelationPartName is specified]}{CorrelationPartValue [Mandatory/can not be empty]}
```

Where:

- *Correlation Name* is customer
- *Correlation Name separator* is #
- *Correlation Part Value separator* is =
- *Correlation Part separator* is ;

## Registering Namespaces

When using namespaces, it is important to remember that XPath matches qualified names (QNames) based on the namespace URI, not on the prefix. Therefore the XPath `"/foo:bar"` may not find a node named `"foo:bar"` if the prefix `"foo"` in the context of the node and in the execution context of the XPath are mapped to different URIs. Conversely, `"/foo:bar"` will find the node named `"biz:bar"`, if `"foo"` in the execution context and `"biz"` in the node context are mapped to the same URI.

### 5.3.2.2. WSDL Schema Validation Policy

From Talend ESB, you can use policies in the Service Registry to validate WSDL schemas. This validation will be performed using the service schema defined in the WSDL.

Talend ESB provides a template policy called `wspolicy_schema_validation.policy` and available in the `/add-ons/registry/policies` folder of the Talend ESB product. This default policy for schema validation is as follows:

```
<wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy"
  Name="wspolicy_schema_validation">
  <wsp:ExactlyOne>
    <wsp:All>
      <tpa:SchemaValidation xmlns:tpa="http://types.talend.com/policy/
assertion/1.0" type="WSDLSchema" appliesTo="provider" message="request"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Where:

- `type` - *WSDLSchema* (if not specified, assumed as *WSDLSchema*),
- `appliesTo` - *consumer/provider/always/none*,
- `message` - *request/response/all/none*.

To enable WSDL Schema Validation via policy:

1. Make sure the Talend Runtime is running, and the Service Registry service has been started. For more information, see the *Talend ESB Container Administration Guide*.
2. Import the WSDL schema validation policy to use, either directly from the Talend Runtime. For more information, see [Using the Service Registry with Talend ESB](#) or via the Talend Administration Center, for more information, see the *Talend Administration Center User Guide*.

This `wspolicy_schema_validation.policy` policy is by default applicable for provider's request, but you can modify it according to your need. For more information about the customization or creation of your own validation policy, refer to the [Custom Schema Validation Policy](#).

3. Assign the policy to the service you want to validate the schema of.

This way, if the WSDL of the service is using a specific restriction in its `xsd:schema`, for example:

```
<xsd:restriction base="xsd:string">
  <xsd:minLength value="20"></xsd:minLength>
  <xsd:maxLength value="30"></xsd:maxLength>
</xsd:restriction>
```

If this restriction is defined in the `<xsd:element>` of the request, then it will be used to validate the request message on the consumer, provider (or both) side. If this restriction is defined in the `<xsd:element>` of the response, then it will be used to validate the response message on the consumer, provider, or always.

In case the restriction is defined in the `<xsd:element>` of the request but you set your schema validation on the response element (the `message="response"` parameter in the schema validation policy), then the validation will not be taken into account.

In case the restriction is taken into account, the validity of the consumer, provider or both, request or response messages sent to the service will be checked at runtime when they will be deployed into the Talend Runtime container. For example, if you send an invalid request to the service, for example a message of less than 20 characters, you will get a Fault response with validation failed information.

### 5.3.2.3. Custom Schema Validation Policy

The validation will be performed using an external customer schema.

The supported attributes are the following:

- `type` - *CustomSchema* (if not specified, assumed as *WSDLSchema*),

- path - URL, absolute or relative path to the custom schema,
- appliesTo - *consumer/provider/always/none*,
- message - *request/response/all/none*.

To enable Custom Schema Validation via policy, upload the following Schema Validation policy to the Service Registry and attach it to a service.

- Remote URL schema location:

```
<wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy"
  Name="wspolicy_schema_custom_validation">
  <wsp:ExactlyOne>
    <wsp:All>
      <tpa:SchemaValidation xmlns:tpa="http://types.talend.com/policy/
assertion/1.0" type="CustomSchema" path="http://localhost:8080/CustomSchema.xsd"
  appliesTo="provider" message="request"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

- Relative path from the root of the Talend Runtime container:

```
<wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy"
  Name="wspolicy_schema_custom_validation">
  <wsp:ExactlyOne>
    <wsp:All>
      <tpa:SchemaValidation xmlns:tpa="http://types.talend.com/policy/
assertion/1.0" type="CustomSchema" path="CustomSchema.xsd" appliesTo="consumer"
  message="response"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

- Absolute path (not recommended, as it is operation system specific and requires that the path exists and is exactly equal on all machines.):

```
<wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy"
  Name="wspolicy_schema_custom_validation">
  <wsp:ExactlyOne>
    <wsp:All>
      <tpa:SchemaValidation xmlns:tpa="http://types.talend.com/policy/
assertion/1.0" type="CustomSchema" path="/opt/CustomSchema.xsd" appliesTo="consumer"
  message="response"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

## 5.3.2.4. Transformation Policy

The Transformation policy allows to apply XSLT transformation to message payloads. The implementation is based on CXF interceptors.

### Policy

To enable the Transformation via the Service Registry, upload the following Transformation policy to the Service Registry and attach it to a service:

```
<wsp:Policy Name="wspolicy_xslt" xmlns:wsp="http://www.w3.org/ns/ws-policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <tpa:Transformation xmlns:tpa="http://types.talend.com/policy/assertion/1.0"
        path="etc/responseTransformation.xsl"
        appliesTo="provider"
        message="response"
        type="xslt" />
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

- The `type` parameter only supports the `xslt` value. The `"xslt"` activates the XSLT transformation using a xslt script. For more information about the XSLT feature of Apache CXF, see <http://cxf.apache.org/docs/xslt-feature.html>.
- The `appliesTo` parameter supports following values: *consumer/provider/always/none*.
- The `message` parameter currently supports following values: *request/response/all/none*.

## XSLT path settings

The **path** attribute can also be specified through the context properties:

```
"org.talend.esb.transformation.xslt-path"
```

If the context properties are specified, they overwrite the corresponding policy attributes.

The **path** attribute can contain:

- HTTP URL's (for example: `http://example.org/xsl/requestTransformation.xsl`),
- Path to an XSL file, relative to the Talend Runtime container (for example: `etc/requestTransformation.xsl` ) or an absolute path,
- Classpath path to the XSL file.

## Dependencies

When running a participant in servlet-container or as a standalone application, the following dependency should be used in the participant's pom.xml file:

### pom.xml for servlet-based or standalone participants

```
<dependency>
  <groupId>org.talend.esb.policies</groupId>
  <artifactId>transformation-policy</artifactId>
  <version>${project.version}</version>
</dependency>
```

When running a participant in the Talend Runtime container, in the *Require-Bundle* section of Felix bundle plugin, the **transformation-policy** bundle should be mentioned:

### OSGi environment pom.xml

```
<plugin>
  <groupId>org.apache.felix</groupId>
```

```
<artifactId>maven-bundle-plugin</artifactId>
<configuration>
  <instructions>
    <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
    <Require-Bundle>
      ...
      transformation-policy
    </Require-Bundle>
  </instructions>
</configuration>
<extensions>true</extensions>
</plugin>
```

## XSLT examples

Here is an example of XSLT identity transformation that transforms a document to itself (which means, no transformation is performed):

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="node()|@"*>
    <xsl:copy>
      <xsl:apply-templates select="node()|@"*/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Here is an example of template which changes a message value to another value:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="node()|@"*>
    <xsl:copy>
      <xsl:apply-templates select="node()|@"*/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="LastName/text()[.='Icebear']">Panda</xsl:template>
</xsl:stylesheet>
```

### 5.3.2.5. Compression Policy

This section shows you how to use the Compression Feature in Talend ESB.

The compression feature of Talend ESB compresses the SOAP Body (and only the SOAP Body) if a certain threshold size is reached. The compression uses a GZIP algorithm with a following base64 encoding. The compressed data is still part of the SOAP Body, so the SOAP Message is still a valid SOAP Message without any changes to the header / http header fields. However, note that using Talend ESB Compression and CXF GZIP Compression together is not recommended.

The Talend ESB Compression feature is driven completely by the `threshold` attribute. So, the supported attribute is `threshold` - the value, in bytes, under which messages are not compressed. And its default value is "1024".

The Compression policy can be enabled via policy or by adding the feature, depending on the type of service (SOAP or REST).

To enable the Compression via policy (for SOAP services only), upload the following Compression policy to the Service Registry and attach it to a Service:

```
<wsp:Policy Name="wspolicy_compression" xmlns:wsp="http://www.w3.org/ns/ws-policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <tpa:Compression xmlns:tpa="http://types.talend.com/policy/assertion/1.0"
        threshold="1000" />
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

The policy must be applied to both Consumer and Provider.

### 5.3.2.6. Service Activity Monitoring Policy

From Talend ESB, you can enable the Service Activity Monitoring feature via a custom policy that the user can maintain and assign to a service via the Service Registry.

1. Make sure the Talend Runtime is running, and the Service Registry service has been started. For more information, see the *Talend ESB Container Administration Guide*.
2. Import the Service Activity Monitoring policy to use, either directly from the Talend Runtime. For more information, see [Using the Service Registry with Talend ESB](#) or via the Talend Administration Center, for more information, see the *Talend Administration Center User Guide*.

Talend ESB provides a default Service Activity Monitoring policy called `wspolicy_sam.policy` in its `/add-ons/registry/policies` folder. This policy is by default applicable for consumer, but you can modify it according to your need.

```
<wsp:Policy Name="wspolicy_sam"
  xmlns:wsp="http://www.w3.org/ns/ws-policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <tpa:ServiceActivityMonitoring
        xmlns:tpa="http://types.talend.com/policy/assertion/1.0"
        appliesTo="consumer" />
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Where `appliesTo` can be consumer, provider, or always (both consumer and provider).

This parameter is restricted to events that can be monitored by the Service Activity Monitoring.

3. Assign the policy to the service for which you want to activate the Service Activity Monitoring.

### 5.3.2.7. Authorization

The Authorization policy enforces that only an authorized user can invoke the request. It is used in conjunction with the SAML policies as defined in [Authentication](#). It asserts that a SAML Token must be present in the request, where the SAML token contains role attributes. The receiver validates the SAML token, and then uses the roles to create an XACML request to the PDP to authorize the user.

Talend ESB provides two template policies, depending on if you are also using Signature/Encryption. They are available here in the Talend ESB product:

- `/add-ons/registry/policies/wspolicy_authn_authz.policy` (Authorization only)

- /add-ons/registry/policies/wspolicy\_authn\_authz\_crypto.policy (Authorization with Signature/Encryption)

```
<tpa:Authorization xmlns:tpa="http://types.talend.com/policy/assertion/1.0"
type="XACML" />
```

These custom policies are also applied by default to your Talend ESB Container via the following policy files:

etc/org.talend.esb.job.saml.authz.policy

etc/org.talend.esb.job.saml.authz.crypto.policy

So if you select the **Authorization** option for your Service in the Studio, when you deploy it on your container, this policy will be pick up automatically.

### 5.3.3. Order of policy execution

The sequence in which those policies are applied to a message is as follows:

#### For outgoing message chain

1. Validation
2. Correlation ID
3. Transformation
4. WS-Security and Authorization
5. Service Activity Monitoring (sending of event to database)
6. Compression

#### For incoming message chain

1. Compression (decompression)
2. Service Activity Monitoring (sending of event to database)
3. WS-Security and Authorization
4. Transformation
5. Correlation ID
6. Validation

For more information on each of these policies, see their related section below.

## 5.4. Typical use cases

Once installed into the Talend Runtime container, the Service Registry can be used with the Talend ESB and with REST Services.



## 5.4.1. Using the Service Registry with Talend ESB

Once initialized, Service Registry offers the following commands within the Talend Runtime container to operate with the Registry:

**Table 5.2. Service Registry commands**

Command	Description
<b>registry:list</b> <type>	Lists Registry resources by type.
<b>registry:create</b> <type> <file>	Installs a Registry resource into the JCR repository.
<b>registry:read</b> <type> <name>	Views the content of a Registry resource.
<b>registry:update</b> <type> <name> <file>	Overwrites the content of a given Registry resource with the specified file.
<b>registry:delete</b> <type> <name>	Removes a Registry resource from the JCR repository.
<b>registry:export</b> [-a] type file	Exports Registry resource(s) to an AtomPub XML file.
<b>registry:import</b> [-o] type file	Imports Registry resource(s) from an AtomPub XML file.

For the above commands:

- `type` refers to the resource type (`wsdl` for WSDL, `metadata` for metadata, `ws-policy` for POLICY, or `ws-policy-attach` for POLICY ATTACHMENT),
- `name` identifies the resource,
- `file` refers to the filepath and filename of the object to upload,
- `-a` exports the policies attached to the WSDL to export,
- `-o` overrides the resources during the import.

To get more detailed information about each command, use the `help` as follows: **help registry:update** or **<registry:update --help**.

Examples of use of these commands:

```
trun> registry:list ws-policy
```

```
Talend ESB Registry :: Collection of ws-policy resources [size:2]
  Name
  - urn:uuid:87654321-abcd-bcde-cdef-123456789000
  - UsernameToken
```

```
karaf@trun> registry:create ws-policy E:/talend/TESB/demo/SAMLToken.policy
```

```
Create Registry ws-policy resource with name =
e12dee61-cbc6-4b22-9555-6b9edfa2dd90 : DONE
```

If synchronization with ElasticSearch is enabled, a set of additional commands becomes available:

**Table 5.3. Service Registry commands**

Command	Description
<b>registry:sync</b> <type>	Updates ElasticSearch with the current resource content of the Service Registry.
<b>registry:search</b> <type> <query>	Implements the specified search query of a Registry resource.

For the above commands:

- `type` refers to the resource type, which only supports metadata,

- `query` refers to a JSON string which is passed to ElasticSearch as query document.

An example of use of these commands:

```
tregistry:search metadata '{ "query": { "term": { "metadataExtension.ServiceName.raw" :
"http://www.talend.org/service/" }DemoService" } } }
```

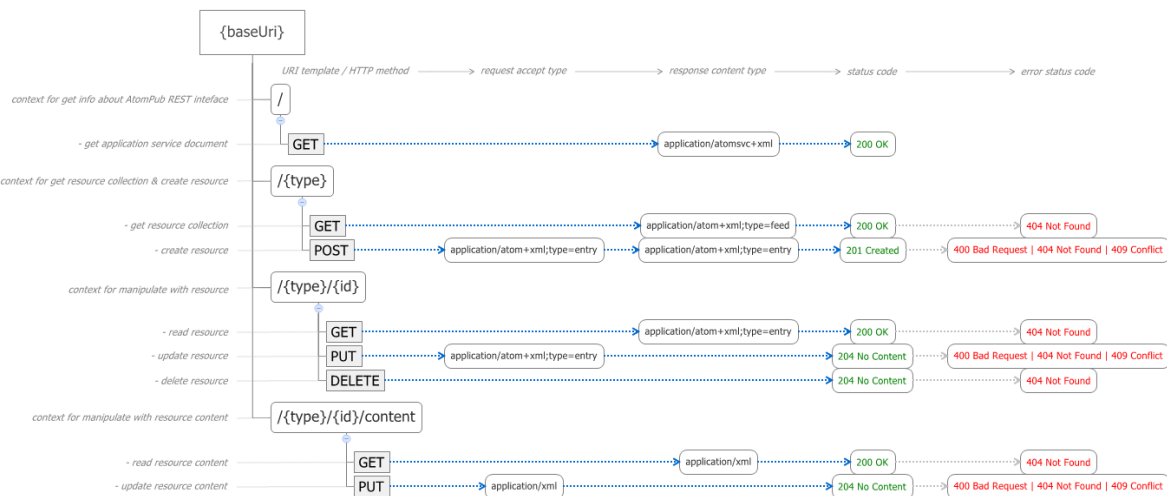
For more information about ElasticSearch, see [Event Logging](#).

Those operations can also be executed via:

- the Service Registry User Interface available in the Talend Administration Center. For more information, see *Managing Services and Policies* chapter in the *Talend Administration Center User Guide*.
- AtomPub REST services. For more information, see [AtomPub REST Service](#).

## 5.4.2. Using the Service Registry with REST Services

The Talend Service Registry REST interface provides access to the Service Registry service in a RESTful manner. This section describes the design for the Registry REST service, which acts as the interface to the Service Registry component. The Service Registry AtomPub REST interface is implemented based on [RFC 5023](#) and [RFC 4287](#) and provides a CRUD (create/read/update/delete) interface over resources in the Service Registry. It is illustrated in the below diagram:



In the above diagram:

- `{baseUri}` refers to the service base URI, `http://localhost:8040/services/registry/admin` by default with Talend Runtime,
- `{type}` - is the resource type (wsdl, ws-policy, or ws-policy-attach),
- `{id}` - resource unique identifier. A UUID is used here.

All AtomPub collections provided by the registry use paged feeds as described in [RFC 5005](#). The number of returned items in a feed is configured by the `atomservice.elementsOnPage` property in the `org.talend.esb.registry.service.admin.cfg` configuration file. By default it is set to 10.

The following resources are available for access: WSDL files, obtainable using `registry/lookup/wsdl/{serviceQName}` and Policy files which can be obtained via `registry/lookup/policy/{serviceQName}`.

### 5.4.2.1. AtomPub REST Service

The REST service WADL (accessible by default from [http://localhost:8040/services/registry/admin?\\_wadl](http://localhost:8040/services/registry/admin?_wadl)) is as below:

```
<application
  xmlns="http://wadl.dev.java.net/2009/02"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <grammars/>
  <resources base="http://localhost:8040/services/registry/admin">
    <resource path="/">
      <doc>Talend Service Registry AtomPub
        administration interface</doc>
      <method name="GET">
        <response>
          <representation mediaType="application/atomsvc+xml"/>
        </response>
      </method>
      <resource path="export">
        <method name="GET">
          <response>
            <representation mediaType="application/atom+xml"/>
          </response>
        </method>
      </resource>
      <resource path="export/policy">
        <param name="policies" style="matrix" repeating="true"
          type="xs:string"/>
        <method name="GET">
          <request/>
          <response>
            <representation mediaType="application/atom+xml"/>
          </response>
        </method>
      </resource>
      <resource path="export/wsdl">
        <param name="services" style="matrix" repeating="true"
          type="xs:string"/>
        <method name="GET">
          <request>
            <param name="attachedPolicies" style="query"
              default="false"
              type="xs:boolean"/>
          </request>
          <response>
            <representation mediaType="application/atom+xml"/>
          </response>
        </method>
      </resource>
      <resource path="import">
        <method name="POST">
          <request>
            <representation mediaType="application/atom+xml"/>
            <param name="override" style="query"
              default="false" type="xs:boolean"/>
          </request>
          <response status="204"/>
        </method>
      </resource>
      <resource path="import/policy">
        <method name="POST">
          <request>
            <representation mediaType="application/atom+xml"/>
            <param name="override" style="query"
              default="false" type="xs:boolean"/>
          </request>
          <response status="204"/>
        </method>
      </resource>
    </resources>
  </application>
```

```

        </method>
    </resource>
    <resource path="import/wsdl">
        <method name="POST">
            <request>
                <representation mediaType="application/atom+xml"/>
                <param name="override" style="query"
                    default="false" type="xs:boolean"/>
            </request>
            <response status="204"/>
        </method>
    </resource>
    <resource path="{type}">
        <param name="type" style="template" type="xs:string">
            <option value="wsdl"/>
            <option value="ws-policy"/>
            <option value="ws-policy-attach"/>
        </param>
        <method name="GET">
            <request/>
            <response>
                <representation mediaType="application/atom+xml"/>
            </response>
        </method>
        <method name="POST">
            <request>
                <representation
                    mediaType="application/atom+xml;type=entry"/>
            </request>
            <response>
                <representation
                    mediaType="application/atom+xml;type=entry"/>
            </response>
        </method>
    </resource>
    <resource path="{type}/{id}">
        <param name="type" style="template" type="xs:string">
            <option value="wsdl"/>
            <option value="ws-policy"/>
            <option value="ws-policy-attach"/>
        </param>
        <param name="id" style="template" type="xs:string"/>
        <method name="DELETE">
            <request/>
            <response status="204"/>
        </method>
        <method name="GET">
            <request/>
            <response>
                <representation
                    mediaType="application/atom+xml;type=entry"/>
            </response>
        </method>
        <method name="PUT">
            <request>
                <representation
                    mediaType="application/atom+xml;type=entry"/>
            </request>
            <response status="204"/>
        </method>
    </resource>
    <resource path="{type}/{id}/check">
        <param name="type" style="template" type="xs:string">
            <option value="wsdl"/>
            <option value="ws-policy"/>
            <option value="ws-policy-attach"/>
        </param>
        <param name="id" style="template" type="xs:string"/>

```

```

        <method name="GET">
            <request/>
            <response>
                <representation mediaType="application/xml"/>
            </response>
        </method>
    </resource>
    <resource path="{type}/{id}/content">
        <param name="type" style="template" type="xs:string">
            <option value="wsdl"/>
            <option value="ws-policy"/>
            <option value="ws-policy-attach"/>
        </param>
        <param name="id" style="template" type="xs:string"/>
        <method name="GET">
            <request/>
            <response>
                <representation mediaType="application/xml"/>
            </response>
        </method>
        <method name="PUT">
            <request>
                <representation mediaType="application/xml"/>
            </request>
            <response status="204"/>
        </method>
    </resource>
</resources>
</application>

```

## AtomPub sample requests

To obtain the AtomPub REST application service document, execute a GET request on address: <http://localhost:8040/services/registry/admin>.

The response will contain a description of all resource collections supported by the service:

```

<service xmlns="http://www.w3.org/2007/app"
  xmlns:atom="http://www.w3.org/2005/Atom">
  <workspace>
    <atom:title type="text">Talend ESB Registry</atom:title>
    <collection href="http://localhost:8040/services/registry/admin/wsdl">
      <atom:title type="text">
        WSDL (Web Service Definition Language)
      </atom:title>
      <accept>application/atom+xml; type=entry</accept>
    </collection>
    <collection href="http://localhost:8040/services/registry/admin/ws-policy">
      <atom:title type="text">
        WS-Policy (Web Services Policy)
      </atom:title>
      <accept>application/atom+xml; type=entry</accept>
    </collection>
    <collection href="http://localhost:8040/services/registry/admin/ws-policy-attach">
      <atom:title type="text">
        WS-PolicyAttachment (Web Services Policy Attachment)
      </atom:title>
      <accept>application/atom+xml; type=entry</accept>
    </collection>
  </workspace>
</service>

```

The below examples are described for the WS-Policy resource type `ws-policy` but also work for the other resource types listed above. For those, just change the resource type parameter accordingly in the request URL.

### To retrieve a collection of policy resources

Execute a GET request using address `http://localhost:8040/services/registry/admin/ws-policy`.

Sample response:

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <id>urn:uuid:021a3df1-037f-3bd6-a65d-9b19ab066063</id>
  <title type="text">
    Talend ESB Registry :: Collection of POLICY resources [size:1]
  </title>
  <author>
    <name>Talend ESB Registry</name>
  </author>
  <generator version="6.4.1">
    Talend ESB Registry AtomPub REST Service
  </generator>
  <updated>2013-03-12T07:59:22.658Z</updated>
  <link href="http://localhost:8040/services/registry/admin/ws-policy"
    rel="self"/>
  <entry xmlns:reg="http://www.talend.com/esb/registry/1.0">
    <author>
      <name>username</name>
    </author>
    <title type="text">title</title>
    <id>urn:uuid:296755cc-bf48-4b78-a8d1-5823566fade4</id>
    <updated>2013-03-12T07:59:22.658Z</updated>
    <link href="http://localhost:8040/services/registry/admin/  \
      ws-policy/296755cc-bf48-4b78-a8d1-5823566fade4/check"
      rel="related"/>
    <link href="http://localhost:8040/services/registry/admin/  \
      ws-policy/296755cc-bf48-4b78-a8d1-5823566fade4" rel="self"/>
    <published>2013-03-12T07:59:22.658Z</published>
    <summary type="text">summary</summary>
    <content type="application/wspolicy+xml"
      src="http://localhost:8040/services/registry/admin/ws-policy/  \
        296755cc-bf48-4b78-a8d1-5823566fade4/content"/>
    <reg:name>SAMLToken</reg:name>
  </entry>
</feed>
```

### To create a policy resource

Execute a POST request using address `http://localhost:8040/services/registry/admin/ws-policy`, with an HTTP header value of `ContentType: application/atom+xml;type=entry` added to the request.

Sample request body:

```
<entry xmlns="http://www.w3.org/2005/Atom">
  <author>
    <name>author</name>
  </author>
  <title>some policy title</title>
  <id></id>
  <updated>2012-09-12T12:53:44.512Z</updated>
  <summary type="text">policy description</summary>
  <content type="application/xml">
    <wsp:Policy Name="usernameToken"
      xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:sp=
        "http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401  \
        -wss-wssecurity-utility-1.0.xsd">
      <wsp:ExactlyOne>
        <wsp:All>
```

```

        <sp:SupportingTokens>
            <wsp:Policy>
                <sp:UsernameToken sp:IncludeToken=
                    "http://docs.oasis-open.org/ws-sx/ws-securitypolicy \\\
                    /200702/IncludeToken/AlwaysToRecipient">
                    <wsp:Policy />
                </sp:UsernameToken>
            </wsp:Policy>
        </sp:SupportingTokens>
    </wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
</content>
</entry>

```

Sample POST response:

```

<entry xmlns="http://www.w3.org/2005/Atom"
  xmlns:reg="http://www.talend.com/esb/registry/1.0">
  <author>
    <name>username</name>
  </author>
  <title type="text">some policy title</title>
  <id>urn:uuid:0989f1a9-d0c6-42df-a223-b41fff7c0395</id>
  <updated>2013-03-12T07:59:22.658Z</updated>
  <link href="http://localhost:8040/services/registry/admin/ \\\
    ws-policy/0989f1a9-d0c6-42df-a223-b41fff7c0395/check" rel="related"/>
  <link href="http://localhost:8040/services/registry/admin/ \\\
    ws-policy/0989f1a9-d0c6-42df-a223-b41fff7c0395" rel="self"/>
  <published>2013-03-12T07:59:22.658Z</published>
  <summary type="text">policy description</summary>
  <content type="application/wspolicy+xml"
    src="http://localhost:8040/services/registry/admin/ws-policy/ \\\
    0989f1a9-d0c6-42df-a223-b41fff7c0395/content"/>
    <reg:name>usernameToken</reg:name>
  </entry>

```

### To read a policy resource

Execute a GET request on address `http://localhost:8040/services/registry/admin/ws-policy/{resource id}` where `resource id` is a UUID similar to the `0989f1a9-d0c6-42df-a223-b41fff7c0395` retrieved from the previous example.

Sample GET response:

```

<entry xmlns="http://www.w3.org/2005/Atom">
  <author>
    <name>username</name>
  </author>
  <title type="text">{no title}</title>
  <id>urn:uuid:0989f1a9-d0c6-42df-a223-b41fff7c0395</id>
  <updated>2013-03-12T07:59:22.658Z</updated>
  <link href="http://localhost:8040/services/registry/admin/ \\\
    ws-policy/0989f1a9-d0c6-42df-a223-b41fff7c0395" rel="self" />
  <published>2013-03-12T07:59:22.658Z</published>
  <summary type="text">{empty summary}</summary>
  <content type="application/wspolicy+xml"
    src="http://localhost:8040/services/registry/admin/ws-policy/ \\\
    0989f1a9-d0c6-42df-a223-b41fff7c0395/content" />
</entry>

```

### To update a policy resource

Execute a PUT request on address `http://localhost:8040/services/registry/admin/ws-policy/{resource id}` with an HTTP header of `ContentType: application/atom+xml;type=entry`.

Sample PUT request:

```
<entry xml:base="http://policy" xmlns="http://www.w3.org/2005/Atom">
  <author>
    <name>author</name>
  </author>
  <title>new policy title</title>
  <id>urn:uuid:0989f1a9-d0c6-42df-a223-b41fff7c0395</id>
  <updated>2012-09-12T12:53:44.512Z</updated>
  <content type="application/xml">
    <!--
      modified policy document here
    -->
  </content>
</entry>
```

### To retrieve or delete a policy resource

Execute a GET or DELETE request, respectively, on address `http://localhost:8040/services/registry/admin/ws-policy/{resource id}`.

### To update a policy resource

Execute a PUT request on address `http://localhost:8040/services/registry/admin/ws-policy/{resource id}` with the new policy document as the request body.

### To create a Policy Attachment

Execute a POST request on address `http://localhost:8040/services/registry/admin/ws-policy-attach` with an HTTP header value of `ContentType: application/atom+xml;type=entry`.

Sample POST request body:

```
<entry xmlns="http://www.w3.org/2005/Atom">
  <author>
    <name>author</name>
  </author>
  <title>my title</title>
  <id>urn:uuid:5296755cc-bf48-4b78-bbbb-5823566fade4</id>
  <updated>2012-09-12T12:53:44.512Z</updated>
  <summary type="text">content summary</summary>
  <content type="application/xml">
    <wsp:PolicyAttachment xmlns:wsp="http://www.w3.org/ns/ws-policy">
      <wsp:AppliesTo>
        <wsp:URI>http://services.talend.org/ReservationService# \\
          wsdl11.service(ReservationServiceProvider)</wsp:URI>
      </wsp:AppliesTo>
      <wsp:PolicyReference URI="usernameToken" />
    </wsp:PolicyAttachment>
  </content>
</entry>
```

Sample POST response:

```
<entry xmlns="http://www.w3.org/2005/Atom"
  xmlns:reg="http://www.talend.com/esb/registry/1.0">
  <author>
    <name>username</name>
  </author>
  <title type="text">my title</title>
  <id>urn:uuid:8cb99f38-20e7-417a-99d3-9f66f45bf216</id>
  <updated>2013-03-12T10:16:31.312Z</updated>
  <link href="http://localhost:8040/services/registry/admin/ \\
    ws-policy-attach/8cb99f38-20e7-417a-99d3-9f66f45bf216" rel="self"/>
  <link href="http://localhost:8040/services/registry/admin/ \\
    ws-policy-attach/8cb99f38-20e7-417a-99d3-9f66f45bf216/check"
    rel="related"/>
  <published>2013-03-12T10:16:31.312Z</published>
  <summary type="text">content summary</summary>
```



```
<content type="application/wspolicy+xml"
  src="http://localhost:8040/services/registry/admin/  \
  ws-policy-attach/8cb99f38-20e7-417a-99d3-9f66f45bf216/content"/>
<reg:targetNamespace>
  http://services.talend.org/ReservationService
</reg:targetNamespace>
<reg:serviceName>
  {http://services.talend.org/ReservationService}  \
  ReservationServiceProvider
</reg:serviceName>
<reg:name>urn:uuid:75c75618-6847-477d-b8b9-d961b003dc56</reg:name>
</entry>
```

### To check the consistency of the resources

Execute GET request on address:

- `http://localhost:8040/services/registry/admin/wsdl/{id}/check` to check the consistency of a specific WSDL, including the policies assigned to it.
- `http://localhost:8040/services/registry/admin/ws-policy/{id}/check` to check the consistency of a specific policy.

Sample of successful response:

```
<resourceCheckerResultCollection>
  <resourceCheckerResult>
    <passed>true</passed>
  </resourceCheckerResult>
</resourceCheckerResultCollection>
```

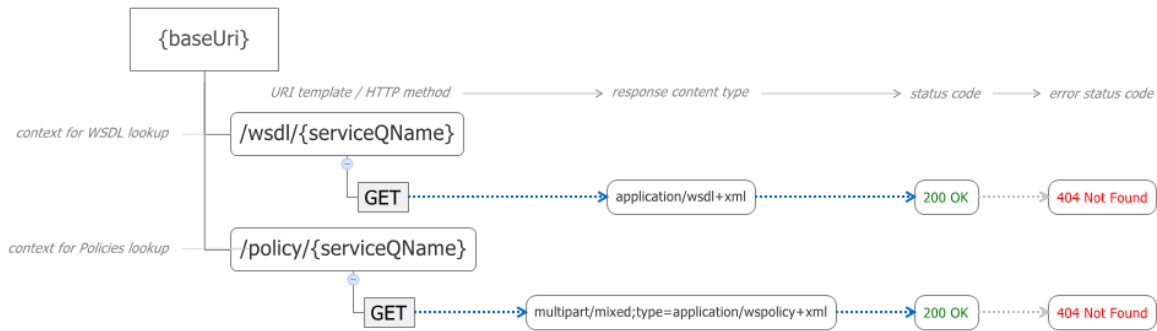
Sample of failed response:

```
<resourceCheckerResultCollection>
  <resourceCheckerResult>
    <failedInfo>
      <failedItems>
        <detailInfo>Cannot find policy resource referenced by uri -
        schemaValidation</detailInfo>
        <item>schemaValidation</item>
      </failedItems>
      <resName>urn:uuid:eed948da-5173-4b6e-b27e-4d7f8967abaf</resName>
    </failedInfo>
    <passed>false</passed>
  </resourceCheckerResult>
</resourceCheckerResultCollection>
```

## 5.4.2.2. Lookup REST Service

The Lookup REST interface diagrammed below retrieves from the Service Registry:

- WSDL documents for a service specified by name
- WSDL documents for a service specified by name, merged with provider policies
- WSDL documents for a service specified by name, merged with consumer policy specified by consumer policy alias
- WS-Policy resources applied to the WSDL subject under the scope of a specified service
- WS-Policy resources applied to the WSDL subject under the scope of a specified service and consumer policy alias



Where:

- **{baseUri}** refers to the service base URI, `http://localhost:8040/services/registry/lookup` by default with Talend Runtime
- **{serviceQName}** refers to the service QName in `{namespace}name` string format
- **{consumerPolicyAlias}** refers to the parameter which can be specified to request consumer policy
- **{withMergedPolicy}** refers to the parameter to use to lookup WSDL document merged with the applied policies

The default URL for accessing the service WADL is `http://localhost:8040/services/registry/lookup?_wadl`.

WSDL and WS-Policy lookups are both handled via GET requests using this interface.

- To lookup a WSDL by `{serviceQName}`, run a GET request on the following address:

```
http://localhost:8040/services/registry/lookup/wsdl/{serviceQName}
```

For example, to get the WSDL for service with name `{http://services.talend.org/ReservationService}ReservationServiceProvider`, execute a GET request using address:

```
http://localhost:8040/services/registry/lookup/wsdl/%257Bhttp%253A%252F%252Fservices.talend.org%252FReservationService%257DReservationServiceProvider
```

- WS-Policy Lookup returns policy documents as a multipart HTTP response. For example, to lookup policies for a service with name `{http://services.talend.org/CRMService}CRMServiceProvider`, execute a GET request on address:

```
http://localhost:8040/services/registry/lookup/policy/%257Bhttp%253A%252F%252Fservices.talend.org%252FCRMService%257DCRMServiceProvider
```

But in case, a non native ESB Java consumer is used (for example, a .NET consumer), it becomes more interesting to get the WSDL with all Policies merged into the WSDL from the Service Registry Lookup service. This enhancement of the Service Registry Lookup is provided as additional options for lookup which the consumer must explicitly call. It is also possible for the consumer to asked, via the consumer policy alias, to get an explicit consumer policy merged into the WSDL for the service.

Below are the new enhanced lookup services:

- To lookup a WSDL by `{serviceQName}` merged with provider policies, run a GET request on the following address:

```
http://localhost:8040/services/registry/lookup/wsdl/{serviceQName}?mergeWithPolicies=true
```

- To lookup a WSDL by `{serviceQName}` merged with consumer policies by `{consumerPolicyAlias}`, run a GET request on the following address:

```
http://localhost:8040/services/registry/lookup/wSDL/{serviceQName}?
mergeWithPolicies=true&consumerPolicyAlias={consumerPolicyAlias}
```

- To lookup provider policies by {serviceQName}, run a GET request on the following address:

```
http://localhost:8040/services/registry/lookup/policy/provider/{serviceQName}
```

- To lookup default consumer policies by {serviceQName}, run a GET request on the following address:

```
http://localhost:8040/services/registry/lookup/policy/consumer/{serviceQName}
```

- To lookup consumer policies by {serviceQName} and {consumerPolicyAlias}, run a GET request on the following address:

```
http://localhost:8040/services/registry/lookup/policy/consumer/{serviceQName}?
consumerPolicyAlias={consumerPolicyAlias}
```

### 5.4.2.3. Authenticating REST requests

To authenticate the requests made to the Service Registry service, enable the authentication parameter: `registry.authentication` in following two configuration files:

- `org.talend.esb.registry.service.admin.cfg` to authenticate the Service Registry AtomPub REST interface.
- `org.talend.esb.registry.service.lookup.cfg` to authenticate the Service Registry Lookup REST interface.

The following settings are possible for this parameter:

- `registry.authentication = NO` is the default configuration. No security is used.
- `registry.authentication = BASIC`. Basic security (Username and Password credentials) is enabled.

### 5.4.2.4. Looking up using consumer policy

With Talend ESB, you can use a custom attribute in the Policy Attachment and distinguish a consumer policy in the Service Registry REST Service. The Service API will be updated to support additional attributes. The Persistence layer is not changed.

The Service Registry REST Service API is extended to support the possibility to get consumer and provider policies:

```
@GET
@Produces({ "multipart/mixed;type=application/wspolicy+xml" })
@Path( "/policy/consumer/{serviceQName}" )
Response lookupConsumerPolicy(@PathParam("serviceQName") String serviceQName,
                             @QueryParam("consumerPolicyAlias") String
                             consumerPolicyAlias);
@GET
@Produces({ "multipart/mixed;type=application/wspolicy+xml" })
@Path( "/policy/provider/{serviceQName}" )
Response lookupProviderPolicy(@PathParam("serviceQName") String serviceQName);
```

The **lookupConsumerPolicy** method returns the consumer policy with `consumerPolicyAlias`, which is provided as a method parameter;

The **lookupProviderPolicy** method returns all provider policies for service which name is provider as method parameter;

The existing **lookupPolicy** method is changed to support new optional parameters: `policyType` and `consumerPolicyAlias`:

```
@GET
@Produces({ "multipart/mixed;type=application/wspolicy+xml" })
@Path("/policy/{serviceQName}")
Response lookupPolicy(@PathParam("serviceQName") String serviceQName,
                      @QueryParam("policyType") String policyType,
                      @QueryParam("consumerPolicyAlias") String consumerPolicyAlias);
```

The **lookupPolicy** method is changed (extended) to support receiving consumer, default, or provider policies. This new implementation of this method is compatible with the previous versions of the Service Registry clients:

policyType	consumerPolicyAlias	lookupPolicy
not set	any value or not set	provider policies (for backwards compatibility)
consumer	not set	default consumer policies (or provider policies, in case the default consumer policy does not exist)
consumer	some consumer policy alias	policies with specified consumer policy alias
provider	any value or not set	all provider policies

Note that the provider policies for the consumer policy type will be retrieved ONLY if consumer policy alias is not specified and no default consumer policies are registered for the whole service. Otherwise the consumer policy will be retrieved and applied independently on scope specified inside the policy.

## 5.4.3. Referencing WS-Policy resources within Service Registry

This section describes how WS-Policy resources (including Policy Attachments) can be associated to subjects (WSDLs), referenced from other Policy objects, and retrieved for a given subject.

### To associate a policy

To associate a WS-Policy resource with the subjects to which they apply in Talend Service Registry, an [External Policy Attachment](#) mechanism is used. Further, [URI Domain Expressions](#) are used to define the scope of the policy:

```
<wsp:PolicyAttachment xmlns:wsp="http://www.w3.org/ns/ws-policy">
  <wsp:AppliesTo>
    <wsp:URI>xs:anyURI</wsp:URI> *
  </wsp:AppliesTo>
  ( <wsp:Policy>...</wsp:Policy>
    | <wsp:PolicyReference>...</wsp:PolicyReference> )
</wsp:PolicyAttachment>
```

The following describes the format for the domain expression URI according to [WSDL 1.1 Element Identifiers](#):

```
<wsdl-target-namespace>#<pointer-part>
```

The possible Fragment Identifiers for `pointer-part` are:

- `wsdl11.service(service)`
- `wsdl11.bindingOperation(binding/operation)`

Note that different namespaces and operations are presently not supported for binding.



When a WSDL subject to which a WS-policy resource is associated gets deleted, this associated WS-Policy attachment will be removed too.

### To reference a policy

To reference an external WS-Policy document from another policy (either WS-Policy or PolicyAttachment) document, a standard [Policy References](#) mechanism is used:

```
<wsp:PolicyReference URI="http://some.domain/policy/samlToken.policy"/>
```

A Name attribute placed on the Policy root element is used to identify the WS-Policy resource within the registry, as shown below. In case a Name attribute was not provided at the time of being imported into the Talend Service Registry, one will be automatically generated with the urn:uuid schema.

```
<wsp:Policy Name="http://some.domain/policy/samlToken.policy">
  ...
</wsp:Policy>
```



When a policy to which a WS-policy document is referenced gets deleted, this referenced WS-Policy document will be removed too.

### To do a policy lookup

To have policy lookup operation performed successfully a resolvable in the Talend Service Registry URI should be provided in its <wsp:PolicyReference>.

Finally, a policy lookup operation contains all policy attachment documents associated with the service. All policies referenced directly (from the policy attachment document) or transitively (from referenced policies) will be embedded in the result policy attachment document. A fault is returned for the case where a referenced policy cannot be resolved.

## 5.4.4. Storing and retrieving Metadata within Service Registry

### 5.4.4.1. Creation and Configuration of the Custom Schema for Metadata

The service metadata structure is defined by two XML schema definition (XSD) resources. The internal XSD owned by the Service Registry describes the frame of the metadata documents and properties mandatory for being handled by the Service Registry. It imports a schema which is expected as file <TalendRuntimePath>/etc/org.talend.esb.registry.server.metadata.xsd. The outline of this schema looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
  targetNamespace="http://metadata.extension.registry.esb.talend.org/v1"
  elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://metadata.extension.registry.esb.talend.org/v1"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <element name="metadata" type="tns:metadata"></element>

  <complexType name="metadata">
    <sequence>
      <element name="CustomServiceType" type="string"></element>
      <element name="CustomCreationDate">
        <simpleType>
          <restriction base="date">
            <pattern value="[^\:Z]*"></pattern>
          </restriction>
        </simpleType>
      </element>
    </sequence>
  </complexType>
</schema>
```

```

        <element name="CustomDescription" type="string"></element>
        <element name="CustomServiceOwner" type="string"></element>
    </sequence>
</complexType>
</schema>

```

This sample schema shows what is expected as parts of a conversion-safe schema. The custom schema information needs to be added in `<sequence>...</sequence>`. In the example shown above it consists of all elements starting with `Custom`. Element names are chosen for demonstration purpose. There is no mandatory naming convention for custom element names. However, there are restrictions to the custom schema imposed by the fact that for import into ElasticSearch, metadata resources need to be unambiguously convertible to JSON documents:

- Custom namespaces are discarded on conversion. Therefore, XML nodes should have unique local names.
- It is recommended to have metadata as a flat sequence of elements or elements nested in a simple way.
- Inheritance and other advanced XML schema constructs are not supported.

When synchronization with ElasticSearch is configured, a JSON mapping template file must be created which matches the custom schema. The mapping file is `<TalendRuntimePath>/etc/org.talend.esb.registry.el.sync.template.json`.

#### 5.4.4.2. Synchronization of Service Registry Metadata with ElasticSearch

Service metadata can be exported to an ElasticSearch server for advanced search functionality. While the connection to ElasticSearch is active, the Service Registry keeps the metadata records in ElasticSearch in synchronized state with the Service Registry database. The ElasticSearch connection does not modify any data in the Service Registry. The data in the Service Registry are always treated as the master data, and the data stored in ElasticSearch are updated to reflect the current state of the Service Registry data.

### ElasticSearch Requirements

ElasticSearch is expected to be running as external server. The HTTPS access is supported in the subscription version of Talend ESB.

### Configuration of ElasticSearch Synchronization

Synchronization of metadata with ElasticSearch is configured by the following files in directory `<TalendRuntimePath>\etc`:

- `org.talend.esb.registry.el.sync.server.cfg`: This file controls activation of the ElasticSearch access by the service registry server.

The `el.enable` property must be set `true` after ElasticSearch access has been properly configured in order to really enable it.

- `org.talend.esb.registry.el.sync.client.cfg`: This file contains access information for ElasticSearch including credentials and SSL keystore information.

Property name	Description
<code>el.host</code>	Hostname of ElasticSearch client node.
<code>el.port</code>	HTTP or HTTPS Port of ElasticSearch client node.

Property name	Description
el.ssl.enable	Enable or disable HTTPS communication.
el.authentication.enable	Enable or disable authentication settings.
el.username	Username for BASIC authentication method.
el.password	Password for BASIC authentication method.
el.index.name	Elasticsearch index name for synchronization.
el.ssl.keystore.password	SSL Keystore password.
el.ssl.keystore.key.password	SSL Keystore Private key password.
el.ssl.keystore.path	SSL Keystore with private key.
el.ssl.keystore.type	SSL Keystore type.
el.ssl.truststore.password	SSL Truststore password.
el.ssl.truststore.path	SSL Truststore with Elasticsearch server certificate.
el.ssl.truststore.type	SSL Truststore type.
el.ssl.enable.cn.check	Enable or disable hostname verification.

- org.talend.esb.registry.server.metadata.xsd: This file is the schema XSD which is used to validate the input metadata XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
  targetNamespace="http://metadata.extension.registry.esb.talend.org/v1"
  elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://metadata.extension.registry.esb.talend.org/v1"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <element name="metadata" type="tns:metadata"></element>

  <complexType name="metadata">
    <sequence>
      <element name="ServiceType" type="string"></element>
      <element name="ServiceStatus">
        <simpleType>
          <restriction base="string">
            <enumeration value="active"></enumeration>
            <enumeration value="inactive"></enumeration>
          </restriction>
        </simpleType>
      </element>
      <element name="ServiceVersion" type="string"></element>
      <element name="CreationDate">
        <simpleType>
          <restriction base="date">
            <pattern value="^[^:Z]*"></pattern>
          </restriction>
        </simpleType>
      </element>
      <element name="Description" type="string"></element>
      <element name="Usage">
        <simpleType>
          <restriction base="string">
            <enumeration value="internal"></enumeration>
            <enumeration value="external"></enumeration>
          </restriction>
        </simpleType>
      </element>
      <element name="ServiceOwner" type="string"></element>
      <element name="ServiceDesigner" type="string"></element>
      <element name="ServiceDevelopmentTeam" type="string"></element>
      <element name="Projectname" type="string"></element>
      <element name="ApplicationName" type="string"></element>
      <element name="ServiceMaintenanceOwner" type="string"></element>
      <element name="ServiceMaintenanceContact" type="string"></element>
    </sequence>
  </complexType>
</schema>
```

```

        <element name="Tags" type="string" maxOccurs="unbounded"
            minOccurs="0">
        </element>
    </sequence>
</complexType>
</schema>

```

- `org.talend.esb.registry.el.sync.template.json`: This file contains a JSON template with information for preparing and indexing of the metadata for search in ElasticSearch. This template must be consistent with the schema of the metadata.

```

{
  "order" : 1,
  "template" : "talendregistry",
  "settings" : {
    "index.number_of_replicas" : "1",
    "index.number_of_shards" : "1",
    "analysis" : {
      "analyzer" : {
        "servicename" : {
          "type" : "pattern",
          "pattern" : "\\{.*\\}",
          "lowercase" : false
        },
        "namespace" : {
          "type" : "pattern",
          "pattern" : "(.*\\{\\}|(\\}\\.)*",
          "lowercase" : false
        }
      }
    }
  },
  "mappings" : {
    "metadata" : {
      "properties" : {
        "metadataExtension" : {
          "properties" : {
            "@schemaLocation" : {
              "type" : "string",
              "index" : "not_analyzed"
            }
          }
        },
        "ServiceName" : {
          "type" : "string",
          "analyzer" : "servicename",
          "fields" : {
            "raw" : {
              "index" : "not_analyzed",
              "type" : "string"
            },
            "namespace" : {
              "index" : "analyzed",
              "analyzer" : "namespace",
              "type" : "string"
            }
          }
        }
      }
    },
    "metadata" : {
      "properties" : {
        "CustomServiceType" : {
          "type" : "string",
          "index" : "not_analyzed"
        },
        "CustomCreationDate" : {
          "type" : "date",
          "format" : "yyyy-MM-dd"
        },
        "CustomDescription" : {
          "type" : "string",

```



```
        "index" : "analyzed"  
    },  
    "CustomServiceOwner" : {  
        "type" : "string",  
        "index" : "analyzed",  
        "fields" : {  
            "raw" : {  
                "index" : "not_analyzed",  
                "type" : "string"  
            }  
        }  
    },  
},  
}  
  
}  
  
}  
  
}
```

Most of the template file needs to be taken over as it is. Only the content of the innermost metadata properties that starts with `Custom` need to be replaced by the properties matching the elements from the corresponding custom schema.

For the present example, matches are as follows:

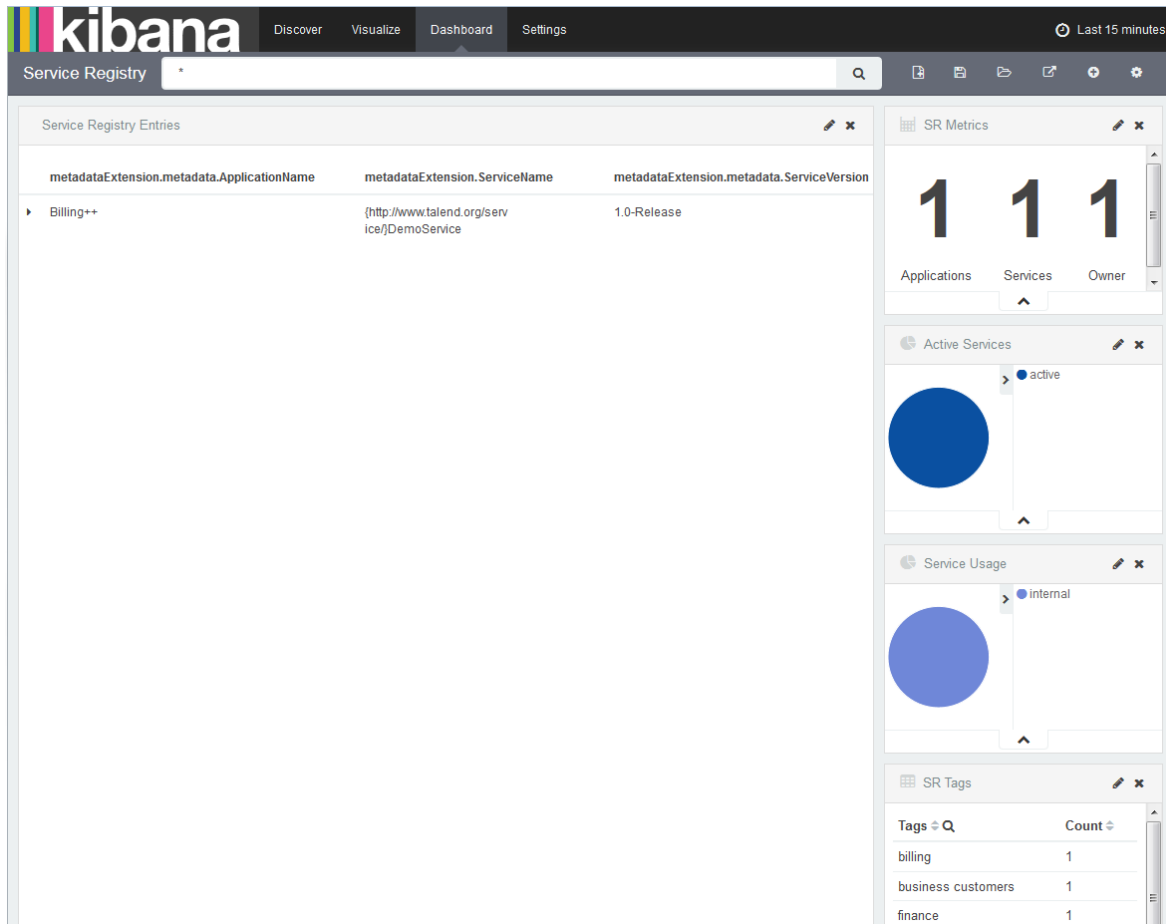
- CustomServiceType: a simple string which should be indexed for search as it is without cutting it into further details.
- CustomCreationDate: a stringified date where the template contains the format for converting the string back into a date.
- CustomDescription: a string which should be cut into words for full-text search.
- CustomServiceOwner: a string which should be cut into words for full-text search, but at the same time the raw string should be searchable as a whole.

#### 5.4.4.3. Browsing Service Metadata in Kibana dashboard

A Kibana dashboard definition `Service-Registry-Dashboard.json` is provided in the `/add-ons/ registry/ dashboard` folder for browsing metadata stored in ElasticSearch with Kibana. This dashboard is designed for the example metadata schema which comes with the Talend ESB installation. For custom metadata, the dashboard definition must be modified to reflect the properties of the custom metadata.

To install the dashboard:

1. Open Kibana. For more information, see the *Talend Administration Center User Guide*.
2. Create an index pattern `talendregistry` from the **Settings** > **Indices** page.
3. Switch to the **Discover** page and select the `talendregistry` index pattern. Make sure there are Service Registry metadata JSON data shown on the right side of the page.
4. Switch to the **Settings** > **Objects** page, select the **visualizations** tab view, then click the **Import** button. Select the file `Service-Registry-Dashboard.json`. You will see four visualizations, one search, and one dashboard are added.
5. Switch to the **Dashboard** page, click **Load Saved Dashboard**, then select **Service Registry**. You will see the Service Registry metadata dashboard is shown:



#### 5.4.4.4. Migration of Service Registry Metadata after Custom Schema Update

Generally, the metadata schema is supposed to be created before any metadata are uploaded to the Service Registry. But you can still modify the original metadata schema after service metadata are uploaded to the Service Registry.

For a compatible schema change, proceed as follows:

1. Add optional elements to the schema definition and the ElasticSearch template like the following:

- In the `<TalendRuntimePath>/etc/org.talend.esb.registry.server.metadata.xsd` file, add

```
...
<element name="CustomNewField" type="string" minOccurs="0"></element>
...
```

- In the `<TalendRuntimePath>/etc/org.talend.esb.registry.el.sync.template.json` file, add

```
...
  "CustomNewField" : {
    "type" : "string",
    "index" : "not_analyzed"
  },
...
```

2. Copy the changed schema and ElasticSearch template to the `<TalendRuntimePath>/etc` folder to replace the old files.

- Restart the Runtime container, or at least run the following commands to restart the Service Registry:

```
tesb:stop-registry
tesb:start-registry
```

- Re-synchronize ElasticSearch with the new template:

```
tregistry:sync metadata
```

For incompatible schema changes, a full-fledged migration is needed as described in the following steps:

- Export the existing metadata:

```
tregistry:export metadata <PATH>/metadata_export.xml
```

- List all the metadata:

```
tregistry:list metadata
```

- Delete each metadata, for example:

```
tregistry:delete metadata '{http://www.talend.org/service/}DemoService'
```

- Update the schema configuration `<TalendRuntimePath>/etc/org.talend.esb.registry.server.metadata.xsd` and index template `<TalendRuntimePath>/etc/org.talend.esb.registry.el.sync.template.json` to the new files.

- Restart the Runtime container, or at least run the following commands to restart the Service Registry:

```
tesb:stop-registry
tesb:start-registry
```

- Re-synchronize the Service Registry to enable the new index template. This will result in dropping and recreating the index in Elasticsearch.

```
tregistry:sync metadata
```

- Update the old metadata to fit the new structure in `<PATH>/metadata_export.xml`.

- Import the updated metadata:

```
tregistry:import metadata <PATH>/ metadata_export.xml
```

- Check Elasticsearch for the updated data, for example:

```
tregistry:search metadata '{ "query": { "term":
{ "metadataExtension.ServiceName.raw" : "{http://www.talend.org/
service/}DemoService" }}}'
```





## Chapter 6. Talend Identity and Access Management

Talend Identity and Access Management, based on Apache Syncope, is a system that allows you to manage the user access to all the Talend web applications. For Talend ESB, it is used to manage users and groups within the ESB Runtime Environment. So, Talend Identity and Access Management is mandatory to use authentication and authorization with Talend ESB, especially as authorization is only supported in combination with the Talend Identity and Access Management.

This product module is only available with the Talend subscription versions of the Talend ESB; it is not included in the Talend Open Studio for ESB.

For Talend Open Studio for ESB, the JAAS Login module is still used by default to handle authentication via the SAML Token (as in the previous versions).

So the users of Talend subscription products need to be created and maintained in the Talend Identity and Access Management to be authenticated and authorized to access your company's ESB resources, however, the use of the JAAS Login module is still possible in the Security Token Service (instead of the default Talend Identity and Access Management). For more information on how to use JAAS Login module instead of Talend Identity and Access Management as authentication handler in the Security Token Service, see the *Talend ESB Container Administration Guide*.

For information about the installation of Talend Identity and Access Management, refer to the *Talend Installation Guide*.

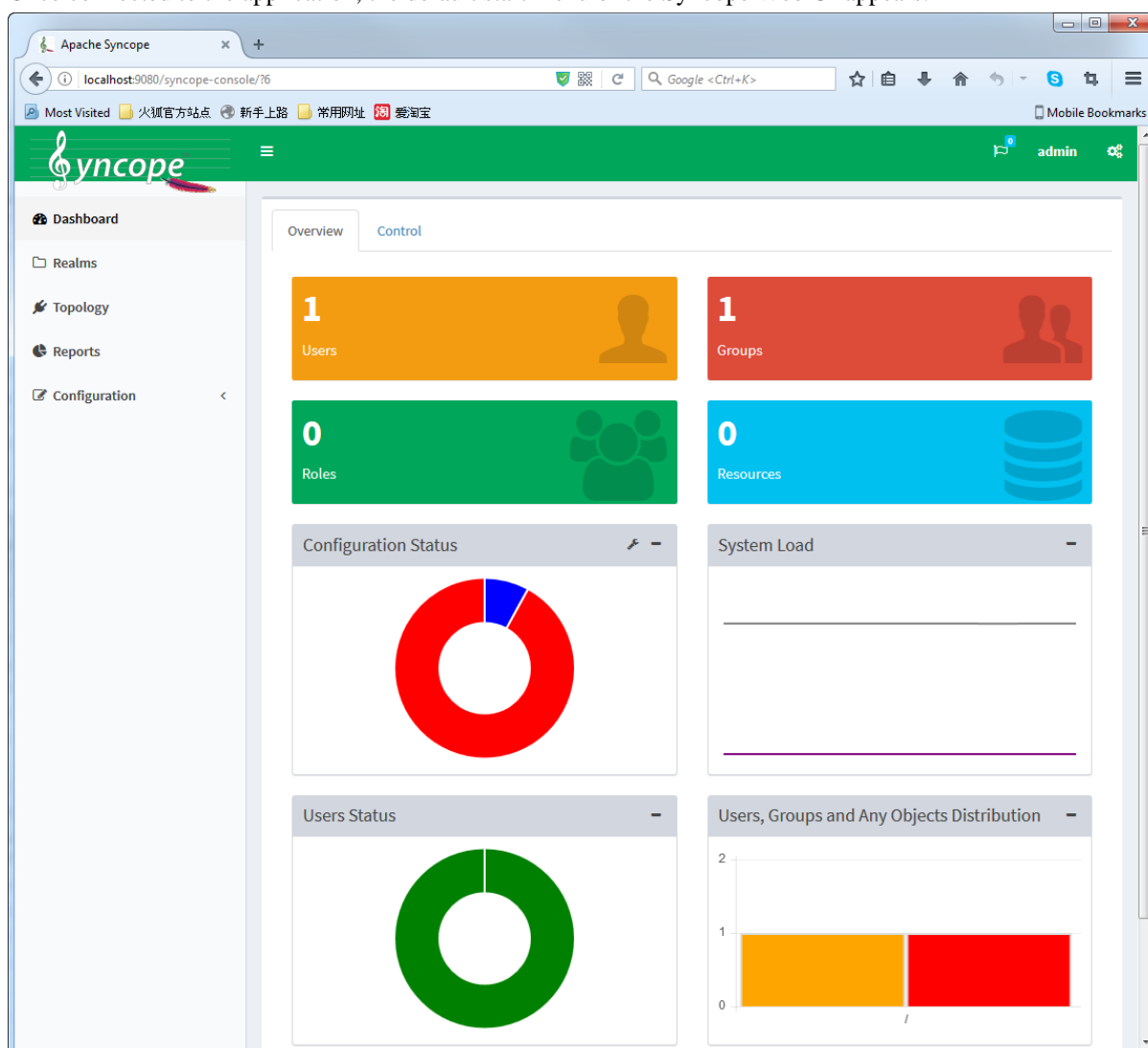
## 6.1. Accessing Talend Identity and Access Management

Talend Identity and Access Management is a system that allows you to manage digital identities in enterprise environments. For Talend ESB, it is used to manage users authentication and users and groups authorization.

Once installed, Talend Identity and Access Management can be accessed at `http://localhost:9080/syncope-console/` with the following default credentials:

Username	Password
admin	password

Once connected to the application, the default start menu of the Syncope Web UI appears.



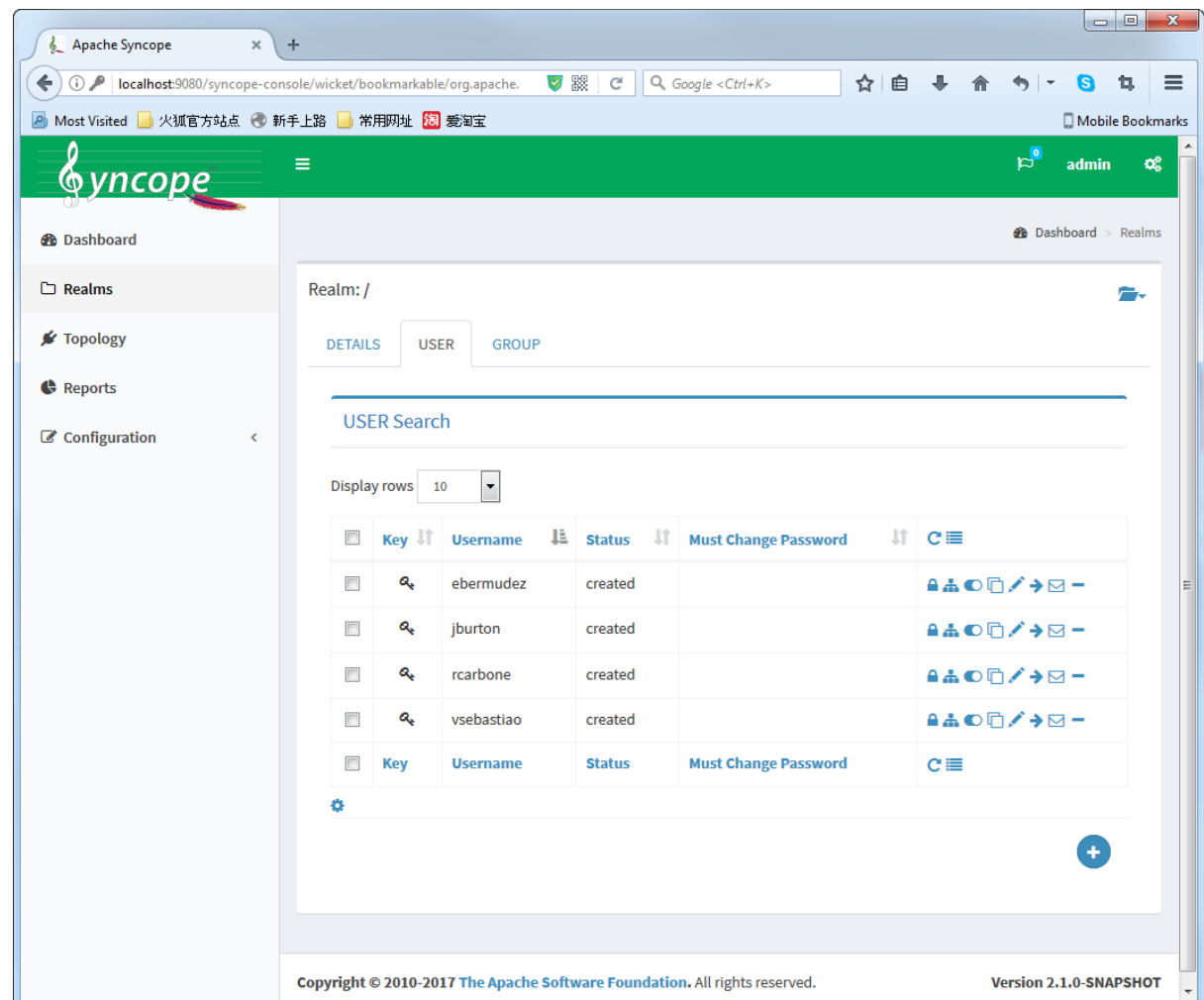
Only the use of the **Users** and **Groups** functionalities will be described in this chapter. Furthermore, the Talend Identity and Access Management can be used as is but it can also be used in synchronisation with other identity management systems or sources of your company. For an advanced use of Apache Syncope functionalities, please refer to its online documentation: <http://syncope.apache.org/docs/index.html>.

## 6.2. Managing user authentication

To be authenticated in Talend ESB, a user must have an account in Talend Identity and Access Management with at least a Username and a Password.

### 6.2.1. Creating a new user

From the main menu of Talend Identity and Access Management, open the **Realms** page and click **Users** to display the list of users.



When you access Talend Identity and Access Management for the first time, no user record is found.

1. To create a new user, click the [+ ] button on the lower-right corner of this tab.
2. The [New User] dialog box appears.

In the **Username** and **Password** fields, type in the username and password that will be used to authenticate Talend ESB users. You have to type in the password a second time to confirm it.



*OpenID authentication only supports usernames in lower case. If you want to use OpenID authentication, the username can not contain capital letters.*

New USER

Destination realm  
/

Username \*  
rcarbon

Password management

Password  
.....

Password (confirm)  
.....

Cancel < Prev Next > Finish

- Keep the default settings of the other options and click **Next** to go to the **Groups** page.

In the **Groups** page, select a group from the **Available** box you want to assign the user to and add it to the **Selected** folder.

When you access Talend Identity and Access Management for the first time, no group is available. You can assign the user to a group later. For more information, see [Assigning a user to a group](#).

New USER

Groups

Search

Available

- employee
- manager

Selected

- admin

Dynamic groups

Cancel < Prev Next > Finish

- Keep the default settings of the other options and click **Next** and then **Finish** to validate your settings and create the new user.

Repeat this operation to create as many users as needed.

Users having an account in Talend Identity and Access Management can now be authenticated if security is enabled in Talend ESB. Furthermore, users credentials can now be used to authenticate them in the Talend Studio when creating Data Services or Routes for example. For more information about authentication in the Talend Studio, see the *Talend Studio User Guide*.

However, if you want to use the authorization functionality in Talend ESB, you should complete the user account by assigning them a group.



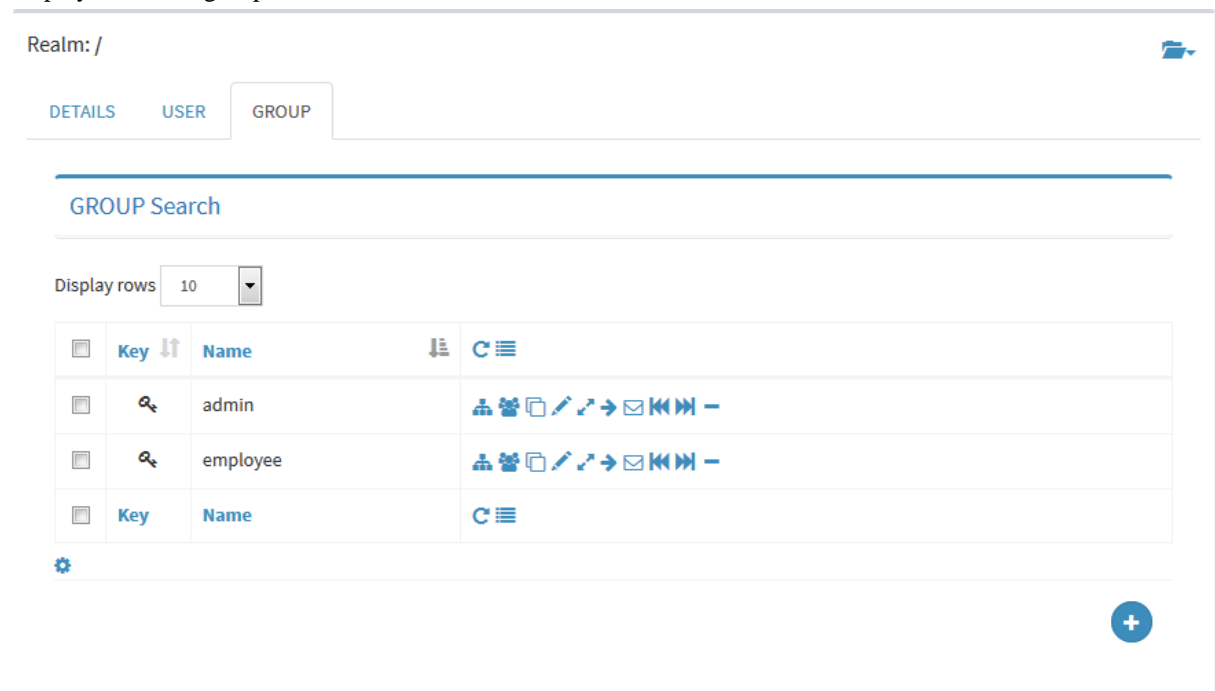
## 6.3. Managing user authorization

Authorization is given to a group of users, so to give authorization to a user, you first have to define the group. Talend Identity and Access Management allows to create and manage those groups and assign users to them.

### 6.3.1. Creating a new group

To manage authorization rights of users already authenticated, you have to assign them a group.

From the main menu of Talend Identity and Access Management, open the **Realms** page and click **Group** to display the list of groups.



Realms: /

DETAILS USER **GROUP**

GROUP Search

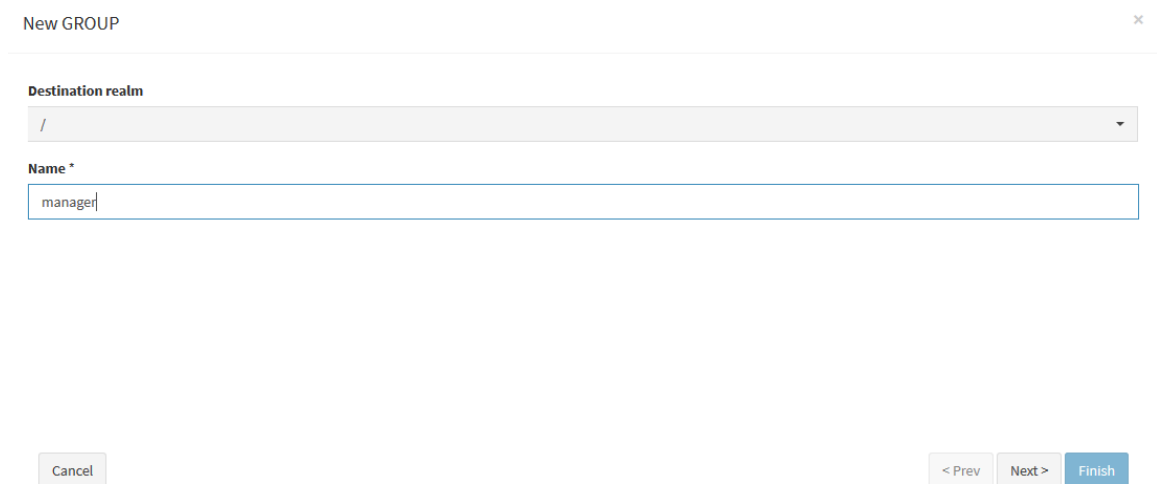
Display rows 10

<input type="checkbox"/>	Key	Name	
<input type="checkbox"/>	admin		
<input type="checkbox"/>	employee		
<input type="checkbox"/>	Key	Name	

+

When you access Talend Identity and Access Management for the first time, no group is found.

1. To create a new group, click the [+] button on the lower-right corner of this tab. to the left of the page. A **New Group** dialog box appears.
2. In the **Name** field, type in the name of the group. Click **Next**.



New GROUP

Destination realm

/

Name \*

manager

Cancel < Prev Next > Finish

- Keep the default settings of the other options and click **Next** and then **Finish** to create the new group.


You can create as many groups as needed.



*Existing Talend ESB users, please note that we now use Syncope **Groups** where we used Syncope **Roles** in previous versions of the Talend ESB and Apache Syncope. With the new Major version of Syncope and the way we use it within Talend Identity and Access Management this is the more appropriate way. The Talend ESB Authorization page in Talend Administration Center and the remaining documentation on the ESB Authorization still uses the term **Roles** by this in the ESB use case the term **Groups** and **Roles** is used currently for the same purpose.*

## 6.3.2. Assigning a user to a group

To assign a user to a group:

- Go to the **Users** page of Talend Identity and Access Management.
- Edit the user by clicking the  icon.
- In the **Edit User** dialog box, click **Next** to go to the **Groups** page.

Edit USER ebermudez

Groups

Search

Available

- employee
- manager

Selected

- admin

Dynamic groups

Cancel < Prev Next > Finish

- Select a group from the **Available** box you want to assign the user to and add it to the **Selected** folder.
- Click **Finish** to finalize the assignment.

You can also assign a user to a group during the creation of the user. For more information, see [Creating a new user](#).

Once users credentials have been created and the users are assigned to groups, authorizations can be provided to those users if the authorization functionality is enabled in Talend ESB. For more information on how to manage users' authorization, see the *Talend Administration Center User Guide*.



## Chapter 7. Authorization with Talend ESB

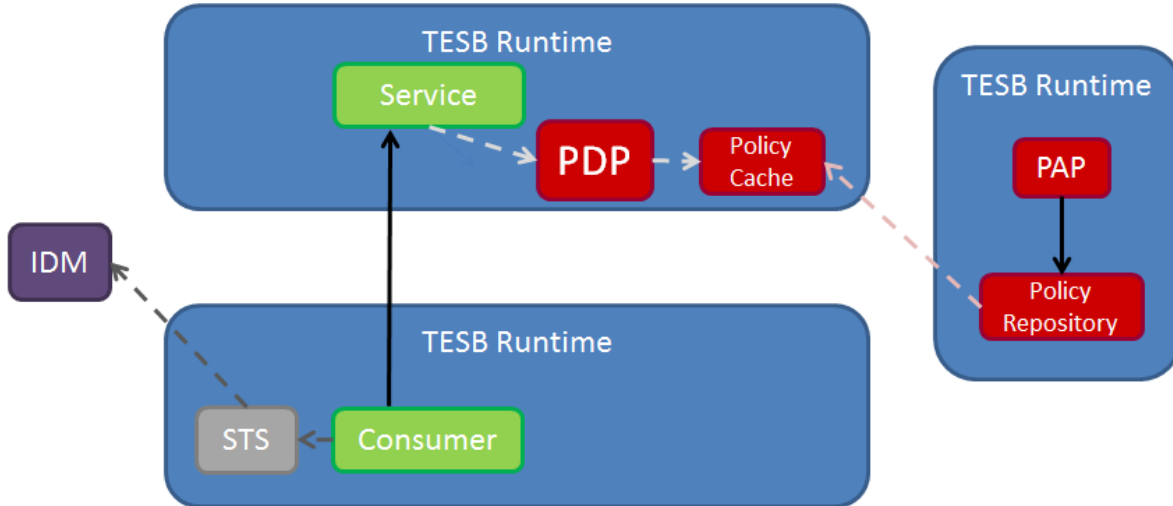
This chapter describes the Talend ESB authorization solution. This product is available with Talend ESB; it is not included in the Talend Open Studio for ESB.

Talend ESB Authorization uses the XACML standard to specify access control. Talend ESB Authorization components are based on this standard and use the [HERAS\\_AF core](#) as the basis of its implementation. As of this version of Talend ESB, the Talend ESB Authorization components support the following:

- PEP (Policy Enforcement Point): A CXF interceptor which intercepts access requests to a resource and enforces the authorization decision of the PDP. This will be described in the next chapter.
- PDP (Policy Decision Point): Requests the needed XACML policies from a policy repository and evaluates the request.
- Policy Repository/Registry: Stores XACML policies. The Talend XACML Registry is based on JCR (Apache Jackrabbit) and is accessed via one front end, a ATOM-based rest interface. It supports deployment, retrieval, and deletion of XACML policies.
- PAP (Policy Administration Point): A user interface for the administration of policies, described in the Talend Administration Center User Guide.
- PIP (Policy Information Point): Supply external policy context and attributes: subject credentials and attributes verification.

## 7.1. TESS Client and Endpoint

This section focuses on requirements for TESS clients and endpoints. It does not describe any of the requirements of the PDP and policy repository infrastructure, beyond describing the XACML interface of the PDP. The following image describes the architecture of the authorization solution as a whole.



The scenario of interest in this section is that of a TESS client invoking a TESS service. In the case of a JAX-WS service endpoint, the WSDL of the service will contain an IssuedToken policy. The RequestSecurityTokenTemplate policy of the IssuedToken policy will require a SAML (1.1 or 2.0) token with a specific Claim corresponding to the role of the client. For example, the following policy requires a SAML 2.0 Token, with an embedded PublicKey and an Attribute containing a role:

```

<sp:RequestSecurityTokenTemplate>
  <t:TokenType>http://.../oasis-wss-saml-token-profile-1.1#SAMLV2.0
</t:TokenType>
  <t:KeyType>
    http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey
  </t:KeyType>
  <t:Claims Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity"
    xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity">
    <ic:ClaimType Uri=
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"/>
    </t:Claims>
</sp:RequestSecurityTokenTemplate>
  
```

The TESS service provider will throw an Exception if the received SAML Token does not contain the corresponding Claim (e.g. role). Note that this means that it is not a requirement for the TESS service provider to obtain the roles of the authenticated principal itself, although this could be added at a later stage.

The TESS client parses the WSDL of the service provider, and creates a RequestSecurityToken element to send to the STS after parsing the IssuedToken policy. The Claim is passed to the STS as part of this Element, which essentially means that the client is requesting that the STS insert the specified Claim into the issued (SAML) token.

The STS must authenticate the client's credentials and obtain the roles of the client to insert into the SAML Token to fulfill the claims that the client has requested. For the case that the client has sent a WS-Security username and password over (one-way) TLS, then the STS endpoint must be instantiated with a custom WSS4J Validator that will validate the given credentials. For example, a JAAS Validator can be used if the IDM backend is an LDAP store like Apache DS. Alternatively, a Validator can be created to invoke on the REST APIs of a product like Apache Syncope.

The SAML Token is then returned to the client, which includes it in the security header of the service request. CXF will automatically extract the principal and roles from the SAML Token, and populate a CXF

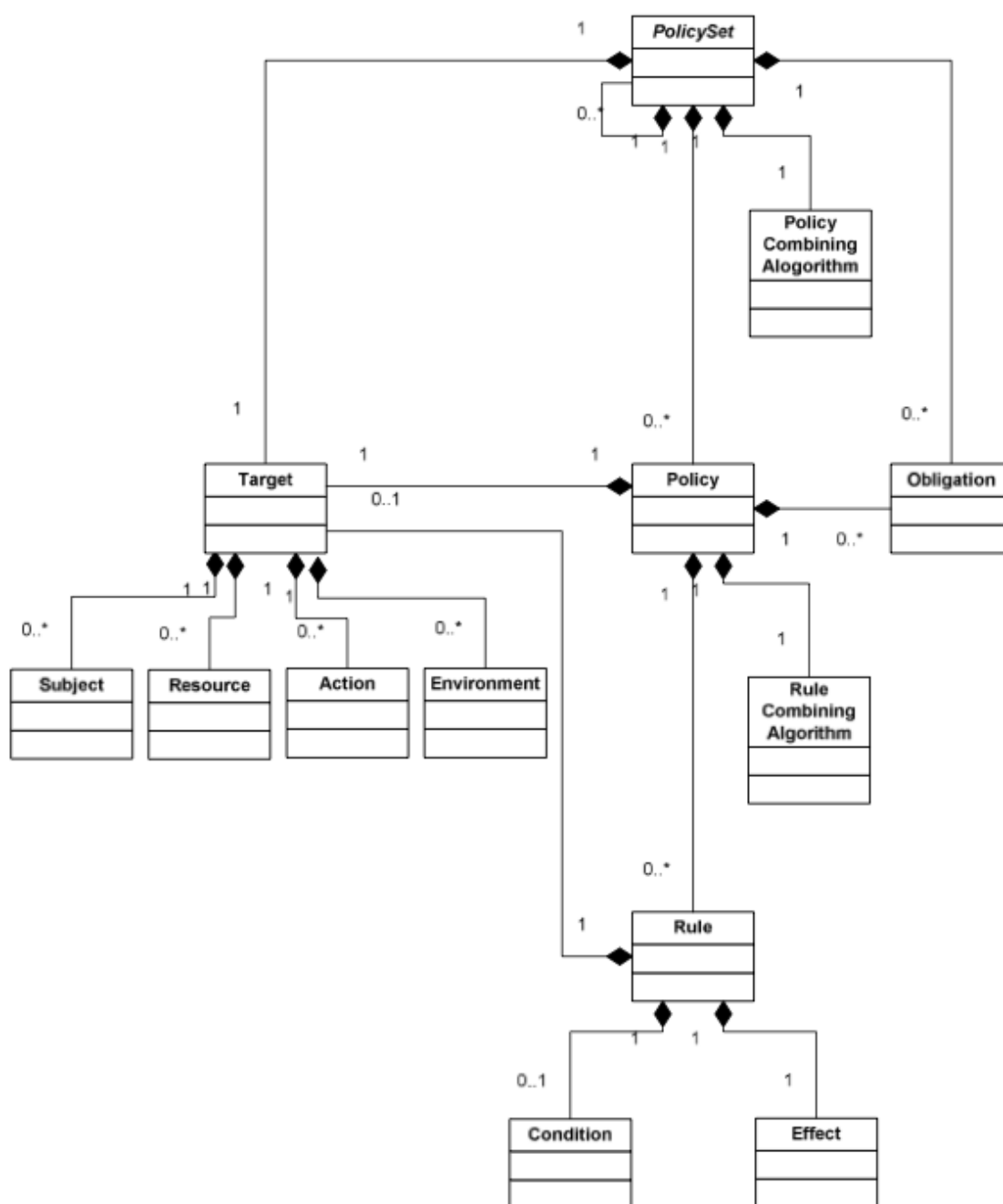
SAMLSecurityContext object. This latter object was added as part of CXF 2.7.0 and is shared between the JAX-WS and JAX-RS runtimes. This means that the XACML interceptors work independently of how the SAML Token was actually received. A CXF interceptor will then create an XACML request from the SecurityContext and dispatch it to a PDP for authorization. The interceptor must then enforce the returned authorization decision, by throwing an exception if access is denied.

A JAX-RS service endpoint does not have a way of advertising what Claims etc. are required as for JAX-WS. In this case, the JAX-RS client must be aware of what the JAX-RS service provider is expecting (e.g. SAML 1.1 or 2.0 etc.) and use these values when obtaining a SAML Token from the STS. An interceptor is provided which wraps CXF's STSClient to obtain a SAML Token from the STS. It then uses any of the standard CXF JAX-RS SAML capabilities (e.g. message pay-load, authorization header, etc.) to insert the SAML Token into the request. The JAX-RS (service) must be instantiated with a CXF JAX-RS provider which will handle the parsing of the SAML Token, and populate the CXF SAMLSecurityContext.

The scenario described above will be broken down into its various tasks in the following sections.

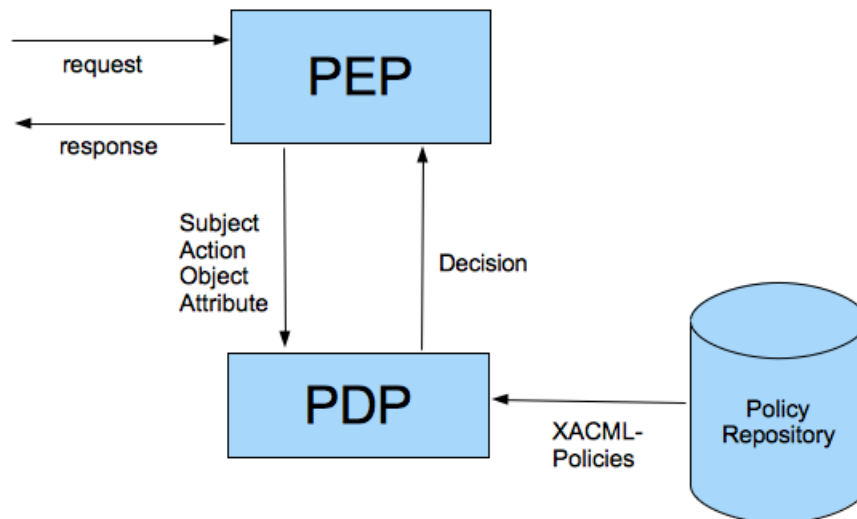
## 7.2. XACML Standard

XACML is a XML based OASIS standard for access control rules called policies. XACML allows a combination of policies and access privileges to be assigned based on attributes assigned to users, roles and other objects. XACML policies are independent from the concrete implementation of the access control. This means, policies can be generated and enforced by different services in a distributed environment. See the below model for a general XACML diagram.



As shown above, the XACML policy consists of policy sets including other policy sets or policy elements. A policy element contains a target and a rule. The target specifies where to apply the policy checking the conditions specified by the rule. Rule elements contain subject, resource and action elements and specify which subject can perform which actions for which resources.

The below diagram further clarifies the interaction between the PEP and the PDP:



Access control based on XACML is specified as follows:

- If access to a resource is required, all related policies are collected and evaluated and based on the result of the evaluation a decision is made whether access is allowed.
- The client requesting the resource interacts only with the PEP, the policy enforcement point. The PEP enriches the client request by additional attributes and forwards it then to the PDP, the policy decision point. The PDP requests the needed policies from a policy store, evaluates the request using the policies and tells the PEP whether access is allowed.

## 7.2.1. Role-Based Access Control

XACML supports RBAC - role based access control - by mapping users and roles on XACML subjects, objects on resources and actions on XACML actions. User-role relations and access control are expressed using policies. Roles and access rights are specified in different types of policies. We call the policies specifying the roles as role policies referring via policy references to its access rights specified in permission policies.

## 7.2.2. XACML policies

For its Authorization feature, Talend ESB is using three types of XACML policies: the Role Policies, the Permission Assignment Policies, and the Permission Policies. Their role can be summarized as follows:

1. A PDP receives a request from a PEP, which contains the resource, action, role, date, and some other optional data.
2. The PDP first goes through the Role Policies it has to try to match the given role name.
3. If it finds a match, then it finds the Permission Policies that are referenced via the Permission Assignment Policy associated with the Role Policy.
4. It matches these policies against the request: the resource and the action name.
5. If they all match then the authorization decision is "permit".

Otherwise, it is "deny" or "indeterminate".

## 7.2.2.1. Permission Policies

The Permission Policy is a <PolicySet> that contains the actual permissions associated with a given role. It contains <Policy> elements and <Rules> that describe the resources and actions that subjects are permitted to access, along with any further conditions on that access, such as time of day. For example:

```
<PolicySet PolicySetId="org.talend.xacml.permissions.boss.doubleit"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-
algorithm:permit-overrides" xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" >

  <Target/>

  <Policy PolicyId="doubleit" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-
combining-algorithm:permit-overrides">

    <Target/>
    <Rule RuleId="doubleit" Effect="Permit">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch
              MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal ">
                <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">{http://www.example.org/contract/DoubleIt}DoubleItService#DoubleIt</
AttributeValue>
                <ResourceAttributeDesignator
                  DataType="http://www.w3.org/2001/XMLSchema#string" AttributeId="
urn:oasis:names:tc:xacml:1.0:resource:resource-id "/>
                </ResourceMatch>
              </Resource>
            </Resources>
            <Actions>
              <Action>
                <ActionMatch
                  MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                    <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">execute</AttributeValue>
                    <ActionAttributeDesignator DataType="http://www.w3.org/2001/
XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
                    </ActionMatch>
                  </Action>
                </Actions>
              </Target>
            </Rule>
          </Policy>
        </PolicySet>
```

In this case, the resource is the {SOAP Target namespace}SOAP Service name#SOAP Operation name: {http://www.example.org/contract/DoubleIt}DoubleIt.

And the action is execute.

So, this permission policy associates the above resource with the execute action. It does not say anything about who is allowed to access this resource, simply that a particular resource is grouped with an action.

For REST, you match against the request URL of the service, and also the HTTP Verb that was used to access the service. For example:

```
<PolicySet PolicySetId="org.talend.xacml.permissions.boss.doubleit-rest"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-
overrides" xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" >

  <Target/>

  <Policy PolicyId="doubleit-rest"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-
overrides">
```



```

    <Target/>
    <Rule RuleId="doubleit-rest" Effect="Permit">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch
              MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
                <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">(/services)?/numberservice/doubleit/(\d)*</AttributeValue>
                <ResourceAttributeDesignator
                  DataType="http://www.w3.org/2001/XMLSchema#string"
                  AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
                </ResourceMatch>
              </Resource>
            </Resources>
            <Actions>
              <Action>
                <ActionMatch
                  MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                    <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">GET</AttributeValue>
                    <ActionAttributeDesignator DataType="http://www.w3.org/2001/
XMLSchema#string" AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
                  </ActionMatch>
                </Action>
              </Actions>
            </Target>
          </Rule>
        </Policy>
      </PolicySet>

```

### 7.2.2.2. Permission Assignment Policy

The Permission Assignment Policy or PolicySet is a <Policy> or <PolicySet> that defines which permissions can be enabled or assigned to which subjects. It may also specify restrictions on combinations of permissions or total number of permissions assigned to or enabled for a given subject. For example:

```

<PolicySet PolicySetId="org.talend.xacml.permissions.assignment.boss"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-
overrides" xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os">

  <Target/>

  <PolicySetIdReference>org.talend.xacml.permissions.boss.doubleit</
PolicySetIdReference>
  <PolicySetIdReference>org.talend.xacml.permissions.boss.doubleit-rest</
PolicySetIdReference>
  <PolicySetIdReference>org.talend.xacml.permissions.boss.quadrupleit</
PolicySetIdReference>
  <PolicySetIdReference>org.talend.xacml.permissions.boss.quadrupleit-rest</
PolicySetIdReference>
</PolicySet>

```

### 7.2.2.3. Role Policies

The Role PolicySet or RPS is a <PolicySet> that associates holders of a given role attribute and value with a Permission <PolicySet> that contains the actual permissions associated with the given role. The <Target> element of a Role <PolicySet> limits the applicability of the <PolicySet> to subjects holding the associated role attribute and value. Each Role <PolicySet> references a single corresponding Permission <PolicySet> but does not contain or reference any other <Policy> or <PolicySet> elements.

A Role Policy associates a Subject with a Permission Assignment Policy. For example:

```
<PolicySet PolicySetId="org.talend.xacml.permissions.role.boss"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-
overrides" xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" >
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-
equal">
          <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#anyURI">boss</AttributeValue>
          <SubjectAttributeDesignator DataType="http://www.w3.org/2001/
XMLSchema#anyURI" AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <PolicySetIdReference>org.talend.xacml.permissions.assignment.boss</
PolicySetIdReference>
</PolicySet>
```

So in this case, a Subject of "boss" is associated with the given permission assignment policy Id.

## 7.3. TESB Authorization XACML PolicyDecisionPoint

Talend ESB ships with a PDP implementation to provide authorization decisions for a TESB endpoint. The TESB PDP is an extension of the [HERAS-AF SimplePDP](#).

There are two ways to access the Talend ESB PDP.

- JAX-RS. The PDP is exposed as a JAX-RS service that allows a JAX-RS client the ability to see whether a given request is authorized or not. The user must POST a XACML Request to `/pdp/authorize`. The next chapter describes how to configure a Policy Enforcement Point (PEP), which takes care of invoking on the PDP and enforcing the authorization decision.
- Co-located. The PDP can be retrieved as a service from the OSGi registry in the container. This allows the PEP to make an authorization request without the overhead of a remote call. See the next chapter for more details.

### 7.3.1. Policy Retrieval Point

The PDP uses a PolicyRetrievalPoint (PRP) implementation to retrieve XACML Policies for evaluation against a request. The TESB PDP ships with a default PRP implementation which retrieves role and permission policies from the XACML Policy Registry. The PRP implementation caches XACML policies to avoid costly calls to the XACML Policy Registry. The default caching mechanism is based on [Ehcache](#). The default cache configuration is specified in the "pdp-ehcache.xml" file. This configuration can be overwritten by specifying a different cache configuration file, as detailed in the next section. The default cache configuration in "pdp-ehcache.xml" is as follows. This describes a cache where policies are not persisted to disk, or overflow to disk, and where policies do not expire in the cache. A separate cache is configured for both role and permission policies:

```
<defaultCache
  maxEntriesLocalHeap="10000"
  eternal="false"
  timeToIdleSeconds="0"
```

```

timeToLiveSeconds="0"
overflowToDisk="false"
maxElementsOnDisk="20000"
diskPersistent="false"
diskExpiryThreadIntervalSeconds="120"
memoryStoreEvictionPolicy="LRU"
/>

```

In addition to the ability to configure how policies are cached via a caching configuration file, it is possible to select a common caching strategy in the PDP configuration file. Three options are supported:

- **InMemory:** XACML policies are kept in memory and not written to disk
- **OverflowToDisk:** XACML policies are kept in memory, but will overflow to disk if the cache is full
- **PersistToDisk:** XACML policies are persisted to disk

When the PDP is started for the first time, it will retrieve all role policies from the XACML Policy Registry. Permission policies are only retrieved as needed from the XACML Policy Registry. So in the course of evaluating a request against the set of role policies, if a role policy matches the request, then the relevant permission policy will be retrieved from the Policy Registry. This policy will then be cached to avoid having to retrieve it again. It is possible to configure the PDP to also retrieve all permission policies on startup.

The PDP is configured with an interval to reload XACML policies from the registry. After the initial policy retrieval, a scheduler is started to retrieve policies from the registry. The policy caches are cleared once this interval elapses, and new policies are downloaded.

## 7.3.2. Policy Information Point (PIP)

It is also possible to add a Policy Information Point (PIP) implementation to the TESB PDP. A PIP can be used to supply information to the PDP runtime which may not be in a request. For example, a policy may only be valid in a particular time interval, which may not be supplied in the request. In this scenario, a PIP implementation could be used to supply the missing attribute. To configure a PIP in the PDP, you must implement the HerasAF "org.herasaf.xacml.core.api.PIP" interface. This must then be registered as an OSGi service in the registry, where it will get automatically picked up by the PDP. In addition to this, you must edit the PDP configuration file (as described in the next section), and set the "usePIP" value to "true".

## 7.3.3. Deployment/Configuration

The PDP can be deployed and started in the ESB container via 'tesb:start-authz-pdp'. The PDP can be configured by `container/etc/org.talend.esb.authorization.pdp.cfg`:

- **registryAtomUrl:** The URL of the XACML Policy Registry to retrieve policies from. The default is 'https://localhost:9001/services/XacmlRegistryAtom'.
- **policyCachingStrategy:** The PolicyCachingStrategy of the PDP (see previous section). The default is "InMemory".
- **cacheConfiguration:** The cache configuration file (see previous section). The default is "pdp-ehcache.xml".
- **loadPermissionPoliciesOnInit:** Whether to load permission policies on startup or not. The default is "false", meaning that they are retrieved (and subsequently cached) when required.
- **policyReloadInterval:** How often to reload policies (in minutes). The default is "10". If set to "0", policies are initially retrieved, and are not reloaded.
- **usePIP:** Whether to use a PIP or not to retrieve attributes that are missing in the request. The default is "false".

## 7.3.4. Using a custom PDP implementation

In some cases the standard PDP service implementation can be changed to a custom PDP implementation. This section describes how to change the bundled PDP by a custom PDP module in the ESB container.

The OSGi bundles related to the PDP rendering are available in the authorization framework in system/org/talend/esb/authorization:

Bundle name	Functions
tesb-xacml-pdp-rt	Talend ESB XACML PDP Runtime which is include the PDP implementation
tesb-xacml-pdp-config	Talend ESB XACML PDP CONFIG which related to tesb-xacml-pdp-rt and includes the configuration for the PDP implementation
tesb-xacml-pdp-api	Talend ESB XACML PDP API includes the interface which should be used for PDP customization
tesb-xacml-pdp-service	Talend ESB XACML PDP Service is a REST service which is using to make a calls to PDP

### 7.3.4.1. Preparing the custom PDP bundle

1. The interface for the PDP customization is included to the following tesb-xacml-pdp-api bundle:

```
package org.talend.esb.authorization.xacml.pdp.api;
import javax.xml.transform.Source;
/**
 * An interface that describes a PolicyDecisionPoint (PDP).
 */
public interface PolicyDecisionPoint {

    /**
     * Evaluate an XACML Request and return a Response
     * @param request an XACML Request as a Source
     * @return the XACML Response as a Source
     */
    Source evaluate(Source request);

}
```

So first of all, make sure this bundle is installed and accessible.

2. The custom PDP bundle is an OSGi bundle which should import the authorization API resources and implement the **org.talend.esb.authorization.xacml.pdp.api.PolicyDecisionPoint** interface.

So, create this OSGi bundle via Maven.

3. Import the ESB XACML PDP API as a dependency to the Maven pom.xml:

```
<dependency>
  <groupId>org.talend.esb.authorization</groupId>
  <artifactId>tesb-xacml-pdp-api</artifactId>
  <version>${project.version}</version>
  <scope>compile</scope>
</dependency>
```

4. When using Spring for the description and rendering of the beans for the PDP implementation, create the beans.xml file in src/main/resources/META-INF/spring if it not exist, and add the PDP interface implementation. For example:

```
<bean id="pdpBean"
  class="org.talend.esb.authorization.xacml.pdp.herasaf.HerasAFPolyDecisionPoint">
```

```
.....
</bean>
```

The **HerasAFPPolicyDecisionPoint** class mentioned in the example above should implement the **PolicyDecisionPoint** interface.

5. Register the custom PDP as an OSGI service:

```
<osgi:service ref="pdpBean"
  interface="org.talend.esb.authorization.xacml.pdp.api.PolicyDecisionPoint" />
```

### 7.3.4.2. Using custom PDP bundle

To use of a custom PDP bundle:

1. Stop and uninstall the standard predeployed PDP bundles: `tesb-xacml-pdp-rt` and `tesb-xacml-pdp-config`.
2. Deploy the prepared custom PDP bundle to the ESB container.
3. Request the PDP service to check how it works. The WADL for the service should be accessible via `http://localhost:8040/services/pdp?_wadl`

## 7.4. TESB Authorization XACML Policy Registry

The XACML registry stores XACML policies using JCR/Jackrabbit, which means all backends supported by Jackrabbit can be configured. As default a file based repository is used, but it can be changed to a database-based repository, for more information see [Backend configuration](#).

The XACML registry rest interface is used by:

- The PDP which retrieves the policies needed to evaluate an authorization request.
- The PAP which supports administration of XACML policies.

The XACML registry distinguishes two types of XACML policies:

- Role policies - used to specify roles.
- Permission policies referred to by the role policies used to specify access rules.

The XACML policy registry client used by the PDP loads all role policies into the memory in advance and supports lazy loading of permission policies.

### 7.4.1. Deployment/Configuration

The XACML registry is deployed as two features in the TESB runtime Karaf container. There is one frontend feature: (`tesb-registry-rest-atom-service`) feature which installs the REST-ATOM frontend to the XACML registry. Make sure that `featuresBoot` in `container/etc/org.apache.karaf.features.cfg` includes `cxfr-abdera`:

```
featuresBoot=config,ssh,management,kar,webconsole, //
  spring,spring-dm,cxfr-abdera,cxfr,camel,...
```

The Apache Jackrabbit XACML repository location is defined in `container/etc/org.talend.esb.authorization.xacml.registry.server.cfg`. The default location is `container/xacmlrepository`. Its Jackrabbit configuration is `container/xacmlrepository/repository.xml`. Here the storage type (file/database based) and repository access rights are specified.

## 7.4.2. Atom REST interface

The Atom REST interface is defined as follows:

```
@GET
@Produces({ "application/atom+xml" })
Response getAtomApplicationDocument();

@GET
@Path("/{type}")
@Produces({ "application/atom+xml" })
ResourceCollection getResources(@PathParam("type") ResourceType type);

@GET
@Path("/{type}/{id}")
@Produces({ "application/atom+xml;type=entry" })
Resource getResource(@PathParam("type") ResourceType type,
    @PathParam("id") String id);

@POST
@Path("/{type}")
@Consumes({ "application/atom+xml;type=entry" })
@Produces({ "application/atom+xml;type=entry" })
@CreateResourceResponse
Resource createResource(@PathParam("type") ResourceType type,
    Resource resource);

@PUT
@Path("/{type}/{id}")
@Consumes({ "application/atom+xml;type=entry" })
@Produces({ "application/atom+xml;type=entry" })
void updateResource(@PathParam("type") ResourceType type,
    @PathParam("id") String id, Resource resource);

@DELETE
@Path("/{type}/{id}")
void deleteResource(@PathParam("type") ResourceType type,
    @PathParam("id") String id);

@GET
@Path("/{type}/{id}/content")
@Produces({ "application/xml" })
Response getResourceContent(@PathParam("type") ResourceType type,
    @PathParam("id") String id);

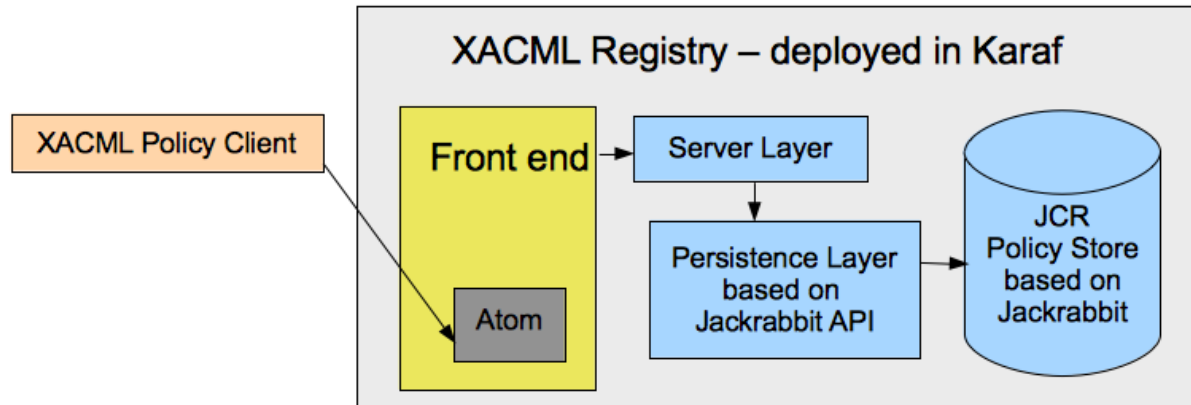
@PUT
@Path("/{type}/{id}/content")
@Consumes({ "application/xml" })
void updateResourceContent(@PathParam("type") ResourceType type,
    @PathParam("id") String id, InputStream body);
```

The XACML Registry Atom REST interface is very similar to the Talend Registry Service. Instead of the WSDL and Policy resource types, the XACML Registry supports resource types ROOT (for roles) and XACML (for access control definitions), but the provided service operations are the same.

The XACML Registry content model is defined analogously to the Talend Registry Content model. The XACML Registry uses Jackrabbit to persist policies. The Registry root has two subnodes, one for roles (labeled "ROOT")

and one for access control definitions referenced by the roles (labeled "XACML"). Other than Talend Registry these subnodes define flat sets of policies, they are itself not hierarchical. A JCR relation between roles and access control definitions is supported by the XACML registry REST frontend but not used by the current PDP implementation.

The XACML Registry Domain Model is shown as below:



## 7.5. Starting and stopping the Authorization service in the Talend Runtime container

After starting the Talend Runtime container, to start the Authorization service, enter the following commands at the console prompt:

1. `tesb:start-authz-repo`
2. `tesb:start-authz-pdp`

You can also shutdown the Authorization service by entering:

1. `tesb:stop-authz-pdp`
2. `tesb:stop-authz-repo`

For more information about how to start the Talend Runtime container, see the *Talend ESB Container Administration Guide*.

## 7.6. XACML Request creation

An [interface](#) is provided in CXF with a method to return an XACML request given a number of parameters. Only [XACML 2.0](#) is considered for Talend ESB, as XACML 3.0 is not supported. It is designed in such a way that the parameters encapsulate all useful information for making an authorization request on either the client or endpoint side. The method parameters are as follows:

- A Principal corresponding to the Subject of the request
- A list of roles corresponding to the roles of the principal
- A CXF Message object describing the current request

The method implementation creates a XACML request using these parameters and marshals it into an [OpenSAML RequestType](#) object. OpenSAML contains some functionality to handle XACML Requests, Responses and Policies, which can be marshalled to DOM Elements, and so it makes sense to re-use this functionality.

A [default implementation](#) is also provided of the interface defined above, that provides a XACML request that will be accepted by the TESB PDP, as well as standard third-party PDPs. The implementation constructs the request from the given parameters by associating the following values with the following (standard) XACML attributes:

- Principal name is mapped to `urn:oasis:names:tc:xacml:1.0:subject:subject-id`
- Each Principal role is mapped to `urn:oasis:names:tc:xacml:2.0:subject:role`
- An Action String is mapped to `urn:oasis:names:tc:xacml:1.0:action:action-id`
- A Resource String is mapped to `urn:oasis:names:tc:xacml:1.0:resource:resource-id`
- The current DateTime is mapped to `urn:oasis:names:tc:xacml:1.0:environment:current-dateTime`

The Principal name and role attributes additionally have an `Issuer` attribute corresponding to the Issuer of the SAML Assertion, as it may be that the PDP requires the knowledge of who provided the roles of the authenticated principal.

The Action String describes the Action being performed, which the XACML specification defines as “an Operation on a Resource”. It is configured differently for both a JAX-RS and JAX-WS service:

- JAX-WS: The action is a statically configured String, defaulting to `execute`.
- JAX-RS: The action is the HTTP verb, e.g. “GET”.

The “Resource” String which describes the JAX-RS or JAX-WS endpoint is extracted from the CXF Message object. The default is as follows:

- JAX-WS: `{Service namespace}Operation` (via CXF's `Message.WSDL_OPERATION`)
- JAX-RS: The REST URI (via CXF's `Message.REQUEST_URI`)

Note that for JAX-RS, the REST URI obtained via `Message.REQUEST_URI` does not include the “https://<ip-address>” prefix. In general, the policy will not care about how the service is deployed. However, this is configurable via a boolean property on the `XACMLRequestBuilder`. If set to true (the default is false), the full Request URL will be sent for both a JAX-WS and JAX-RS service.

Typically, a JAX-RS request includes a variable parameter, which you might not care about for authorization. XACML is flexible enough to handle this using regular expressions. For example, the following is a resource in an XACML request as sent by CXF:

```
<xacml-context:Attribute
  AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
  DataType="http://www.w3.org/2001/XMLSchema#string">
  <xacml-context:AttributeValue
    /numberservice/doubleit/20
  </xacml-context:AttributeValue>
</xacml-context:Attribute>
```

A policy that will successfully match this resource is as follows:

```
<Resources>
  <Resource>
    <ResourceMatch MatchId=
      "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
      <AttributeValue DataType=
        "http://www.w3.org/2001/XMLSchema#string">
        /numberservice/doubleit/(\d)*
      </AttributeValue>
      <ResourceAttributeDesignator
        DataType="http://www.w3.org/2001/XMLSchema#string"
```



```

        AttributeId=
            "urn:oasis:names:tc:xacml:1.0:resource:resource-id" />
    </ResourceMatch>
</Resource>
</Resources>

```

An example of a XACML request for a JAX-WS service using the definitions given above is listed below.

```

<xacml-context:Request
  xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:os">
  <xacml-context:Subject SubjectCategory=
    "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <xacml-context:Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
      DataType="http://www.w3.org/2001/XMLSchema#string"
      Issuer="STSIssuer">
      <xacml-context:AttributeValue>
        alice
      </xacml-context:AttributeValue>
    </xacml-context:Attribute>
    <xacml-context:Attribute
      AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
      DataType="http://www.w3.org/2001/XMLSchema#anyURI"
      Issuer="STSIssuer">
      <xacml-context:AttributeValue>
        manager
      </xacml-context:AttributeValue>
    </xacml-context:Attribute>
  </xacml-context:Subject>
  <xacml-context:Resource>
    <xacml-context:Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <xacml-context:AttributeValue>
        {http://www.example.org/contract/DoubleIt}DoubleIt
      </xacml-context:AttributeValue>
    </xacml-context:Attribute>
  </xacml-context:Resource>
  <xacml-context:Action>
    <xacml-context:Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <xacml-context:AttributeValue>
        execute
      </xacml-context:AttributeValue>
    </xacml-context:Attribute>
  </xacml-context:Action>
  <xacml-context:Environment>
    <xacml-context:Attribute AttributeId=
      "urn:oasis:names:tc:xacml:1.0:environment:current-dateTime"
      DataType="http://www.w3.org/2001/XMLSchema#dateTime">
      <xacml-context:AttributeValue>
        2012-10-09T14:36:07.003Z
      </xacml-context:AttributeValue>
    </xacml-context:Attribute>
  </xacml-context:Environment>
</xacml-context:Request>

```

## 7.7. XACML Response validation

Once the XACML Request described in the previous section has been created, it must be dispatched to the PDP (as covered in the next few sections). The PDP evaluates the Request, and constructs a XACML Response and returns it to the client.

The PDP can return a decision of `Permit`, `Deny`, `NotApplicable` or `Indeterminate`. Access is allowed only if the decision of the PDP is `Permit`. For any other decision, the PEP will throw a `CXF AccessDeniedException`. The PDP can also return an `Obligations` Element that is defined in the relevant policy as part of the request. The PEP is supposed to only grant access on a `Permit` decision if it can satisfy all `Obligations`. The TESB PEP does not support `Obligations` by default, but does have a pluggable way of handling an `Obligations` element if required.

An example of a XACML response is given below.

```
<Response
  xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
  <Result>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok" />
    </Status>
  </Result>
</Response>
```

## 7.8. TESB service provider PEP

A CXF interceptor (see [here](#) for its implementation) is available that provides the base functionality of a Policy Enforcement Point (PEP) for a TESB service provider. It uses the XACML creation and processing functionality described earlier. It has a reference to the XACML creation interface which uses the default implementation, but also has accessor methods so that a custom implementation can be configured instead.

The interceptor obtains the Principal name and roles via a CXF `SAMLSecurityContext` object on the CXF message. For the case of a JAX-WS service endpoint that receives a SAML Token, the `WSS4JInInterceptor` will automatically create a `SAMLSecurityContext` using the principal corresponding to the Subject of the SAML Token, and the roles extracted using a URI from the Attributes. This URI can be configured on the endpoint via a JAX-WS property. For the REST case, a `SAMLSecurityContext` is also created with the same information.

Once the request has been created, it must be dispatched to the PDP. The TESB PEP implementation, which wraps the basic CXF interceptor, provides functionality to send the request to the TESB PDP (described in the previous chapter). The PDP request can happen in one of two different ways:

- A remote HTTP request to the TESB JAX-RS PDP using POST
- A local request to the co-located PDP (that could have been obtained from the OSGi registry, for example)

### 7.8.1. Enabling and configuring the TESB PEP

To enable authorization on a TESB service endpoint, it is necessary to install the TESB PEP interceptor. This can be done in a number of different ways. The easiest way for a JAX-WS based endpoint is to use the following WS-Policy expression:

```
<tpa:Authorization xmlns:tpa="http://types.talend.com/policy/assertion/1.0"
  type="XACML" />
```

This will automatically install the `PolicyEnforcementPoint` interceptor and ensure that only authorized requests invoke on the endpoint. When the PEP is installed in this way, an additional property ("`tesb.pdp.address`") is needed to tell the PEP where to find the PDP. This can be done in the "`etc/org.talend.esb.authorization.pdp.cfg`" configuration file, by setting a value for the "`tesb.pdp.address`" property. Alternatively, it can be set as a property on the endpoint, e.g.:

```
<jaxws:server ...>
```

```

<jaxws:properties>
  <entry key="tesb.pdp.address"
    value="https://localhost:9001/services/pdp/authorize"/>
</jaxws:properties>
</jaxws:server>

```

See the 'syncpe-esb-xacml' example for more information on adding the PolicyEnforcementPoint to a JAX-WS service endpoint. It is also possible to create the PEP interceptor and add it directly to the CXF interceptor chain for the endpoint. For example:

```

<bean
  class="org.talend.esb.authorization.xacml.rt.pep.CXFXACMLAuthorizingI
nterceptor"
  id="XACMLInterceptor">
  <property name="pdpAddress"
    value="https://localhost:9001/services/pdp/authorize"/>
</bean>

```

This can then be added to the Interceptor chain of a JAX-WS endpoint via:

```

<jaxws:endpoint ...>
  <jaxws:inInterceptors>
    <ref bean="XACMLInterceptor"/>
  </jaxws:inInterceptors>
</jaxws:endpoint>

```

The PEP can also be added to the Interceptor chain of a JAX-RS endpoint via:

```

<jaxrs:server ...>
  <jaxrs:inInterceptors>
    <ref bean="XACMLInterceptor"/>
  </jaxrs:inInterceptors>
</jaxws:endpoint>

```

See the 'syncpe-esb-xacml-rest' example for more information on adding the PolicyEnforcementPoint to a JAX-RS service endpoint. An example of how to use the co-located PDP is given in the 'syncpe-esb-xacml-coloc' example. In this example, the service provider obtains the PDP from the OSGi registry via:

```

<reference id="pdpBean"
  interface="org.talend.esb.authorization.xacml.pdp.PolicyDecisionPoint"/>

<bean class="org.talend.esb.authorization.xacml.rt.pep.CXFXACMLAuthorizingI
nterceptor"
  id="XACMLInterceptor">
  <property name="policyDecisionPoint" ref="pdpBean"/>
</bean>

```

## 7.9. TESB client REST STS Interceptor

A CXF interceptor is required to communicate with the STS and to obtain a security token. For the WS-\* case, the IssuedTokenInterceptorProvider is automatically triggered in the CXF WS-Security runtime by an IssuedToken policy associated with the service provider. All that is required in this case is that the STSClient bean is Spring configured.

For the REST case, there is no such interceptor in CXF. A new interceptor STSRESTOutInterceptor is provided in the ESB to communicate with the STS and store the received token on the security context. It must be configured with an STSClient object as per the WS-\* case. The resulting token is stored on the security context using the SAMLConstants.SAML\_TOKEN\_ELEMENT tag. This is picked up automatically by either the SamlFormOutInterceptor or the SamlHeaderOutInterceptor in the JAX-RS security runtime, depending on which has been configured. The SAML Token will then get written out as part of a Form or in the Authorization Header to the JAX-RS service.

See the 'syncope-esb-xacml-rest' example to see how to use the `STSRESTOutInterceptor` to obtain a SAML Token from the STS from a JAX-RS client, and how to use it in conjunction with the JAX-RS security runtime to send the issued token to the JAX-RS endpoint.



## Chapter 8. XKMS Service

This chapter describes the Talend ESB XKMS implementation. This service is only available with the subscription version of the Talend ESB; it is not included in the Talend Open Studio for ESB.

The XML Key Management Specification (XKMS) is an XML-based protocol that is used for the distribution and registration of public keys, and Talend ESB uses it for encryption and signing of messages.

## 8.1. Overview

The Public Key Infrastructure (PKI) is a system for encrypting, decrypting, signing, authorizing and verifying the authenticity of information transmitted over the Internet, or of people's identity, using public-key cryptography. In a PKI system, a user or business has two keys, a public key and a private key. The public key is used to encrypt information by those who want to send private information to the user and only the private key of the user can decrypt the information.

To manage Public Key Infrastructure, Talend ESB is using XML Key Management Specification (XKMS) which handles the distribution and registration of public keys in conjunction with XML Signature [XML-SIG] and XML Encryption [XML-ENC] to decouple PKI complexity.

XKMS does not handle the actual work of managing public and private key pairs and other PKI details. Instead, it outsources the jobs of key registration, validation, and similar processes to an XKMS trust utility. The XKMS trust utility works with any PKI system, passing the information back and forth between it and the Web service.

XKMS itself is made of two standards:

- XML Key Information Service Specification (X-KISS) which performs location and validation of keys.
- XML Key Registration Service Specification (X-KRSS) which supports the key registration and management functionality.

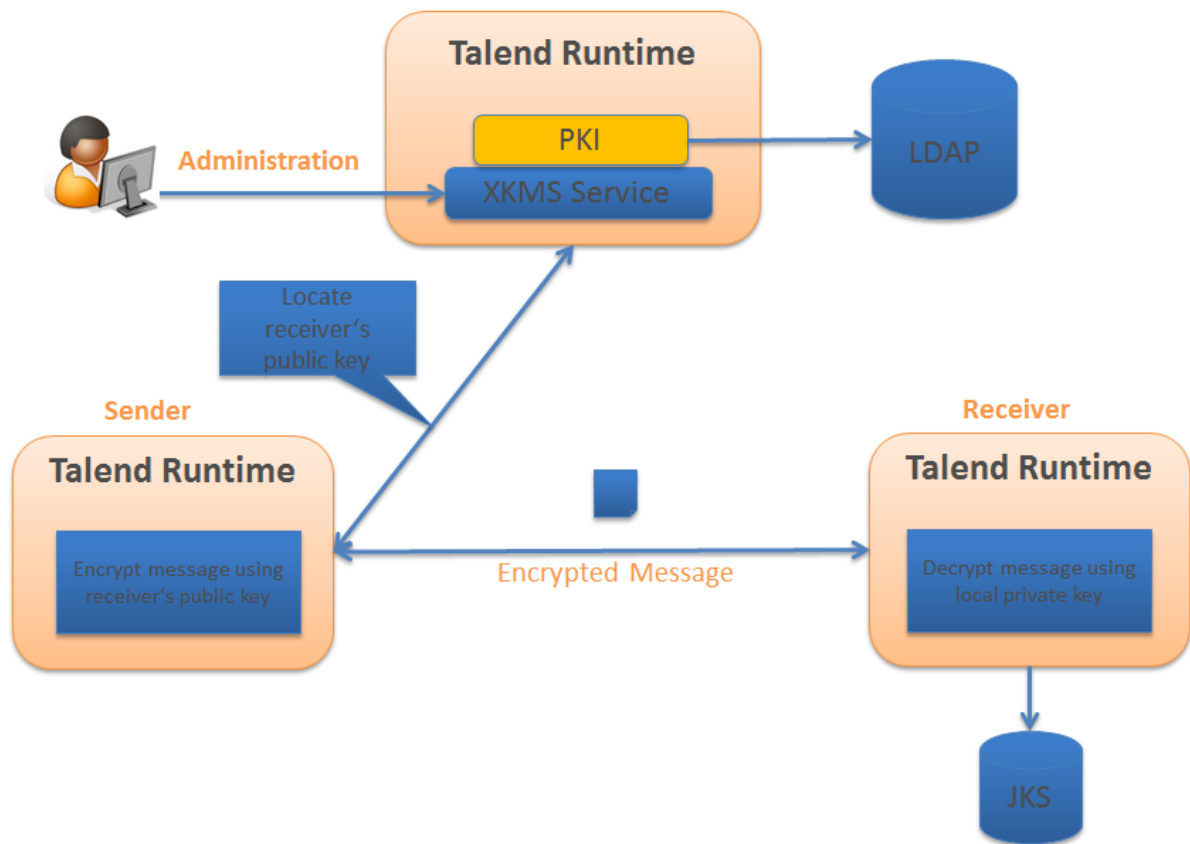
The X-KISS protocol provides the two following operations:

- Locate, which resolves a `<ds:KeyInfo>` element that may be associated with XML encryption or XML signature, but it does not determine the validity of the binding between the data and the `<ds:KeyInfo>` element and therefore does not certify that the binding information can be trustworthy. It may relay the request to other services or act as a gateway to the PKI.
- Validate, which does all that the Locate operation does: it looks for the public key that corresponds to the `<ds:KeyInfo>` element, and in addition, it determines the validity and trustworthiness of the binding between the data and the returned key.

So, Talend ESB employs the XML Signature [XML-SIG] for the purpose of providing message security in the form of authentication and integrity. With the help of the XKMS service, the use of XML Signature by the Talend ESB is simplified, as it minimize the complexity and syntax of the underlying public key infrastructure used to establish trust relationships.

And Talend ESB also employs the XML Encryption [XML-ENC] for the purpose of sending secured message to a receiver. This way, even if a client does not know the public key of a receiver, it can still query the XKMS service for it as XKMS is based on the use of the `<ds:KeyInfo>` element as a means of transporting key information used as templates for the various operations it specifies.

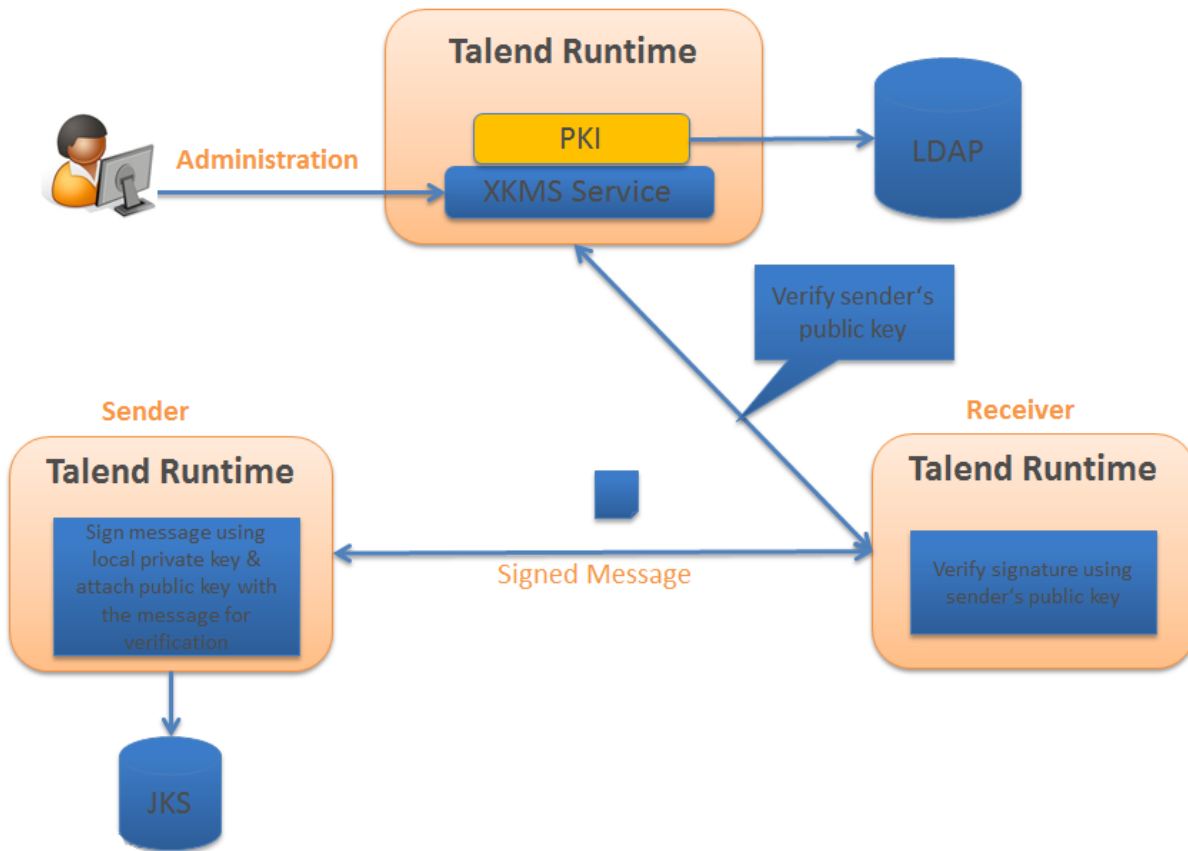
## 8.1.1. Encryption functional architecture



To send encrypted data to a receiver, senders locate the public key of the receiver in the XKMS repository via Service QName, and use this public key to encrypt the data.

The receiver will now be able to decrypt the data with the private key that corresponds to the public used for encryption.

## 8.1.2. Signature functional architecture



To send encrypted data to a receiver, senders encrypt the data with their own certificate (they sign the data). This way, the data are sent to the receiver associated with their own public key.

The receiver receives the signed data, validate that the public key is in the XKMS repository, and if the public key is valid, the receiver will be able to access the data.

## 8.2. Configuring the XKMS Service

First of all, you need to start the XKMS service in the Talend Runtime container. To do so, after starting the Talend Runtime container, enter the following command at the console prompt:

```
tesb:start-xkms
```

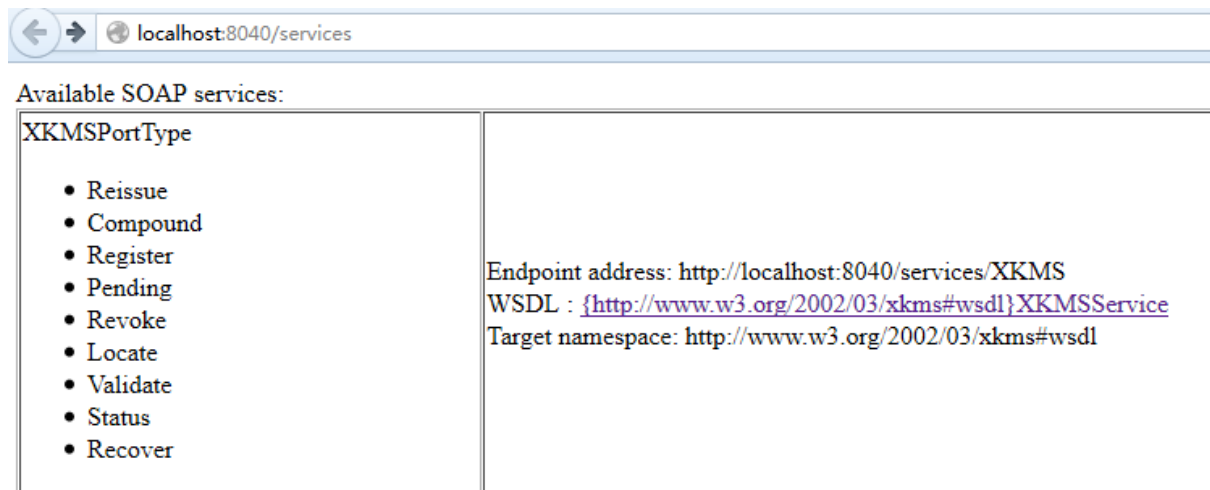
You can also shutdown the XKMS service by entering:

```
tesb:stop-xkms
```

For more information about how to start the Talend Runtime container, see the *Talend ESB Container Administration Guide*.

Once started, the XKMS service provides the following operations at <http://localhost:8040/services/>: Reissue, Compound, Register, Pending, Revoke, Locate, Status, Recover, and Validate.





Once the XKMS Service started, you can configure it by editing the *etc/org.apache.cxf.xkms.cfg* file. By default, it will use LDAP as backend repository but you can change it to File backend repository for testing purpose.

Below is the example of the default configuration for the use of File backend repository:

```
# XKMS configuration properties
xkms.enableXKRSS=true

# Certificate repository ldap or file
xkms.certificate.repo=file

# Filesystem backend
xkms.file.storageDir=${karaf.home}/esbrepo/xkms/certificates
```

To use your own public certificates (\*.cer files), copy them to the XKMS File backend repository in `${karaf.home}/esbrepo/xkms/certificates/trusted_cas`.

## 8.3. Generating key pairs for Signing and Encryption with ESB

1. Generate a keypair for client:

```
keytool -genkey -keystore myclientstore.jks -alias myclientalias -dname "client" -
keyalg RSA -sigalg SHA1withRSA -validity 3650 -storepass myclientstorepass -keypass
myclientkeypass
```

```
keytool -export -keystore myclientstore.jks -alias myclientalias -file
myclientcertificate.cer -storepass myclientstorepass
```

2. Generate a keypair for service:

```
keytool -genkey -keystore myservicestore.jks -alias myservicealias -dname "{http://
services.talend.org/ReservationService}ReservationServiceProvider" - keyalg
RSA -sigalg SHA1withRSA -validity 3650 -storepass myservicestorepass -keypass
myservicekeypass
```

```
keytool -export -keystore myservicestore.jks -alias myservicealias -file
myservicecertificate.cer -storepass myservicestorepass
```

3. Register public certificate into the XKMS repository:

For encryption and signing, the public certificates from the client and the service should to be located or validated by the xkms service. To enable this, copy the .cer files under <TalendRuntimePath>/container/esbrepo/xkms/certificates/trusted\_cas.

4. Configure the Service consumer and providers for signing and encryption.

#### **For the Service consumer configuration:**

1. Edit the etc/org.talend.esb.job.client.cfg configuration file:

```
security.signature.properties = file:${tesb.home}/etc/keystores/
clientKeystore.properties
security.signature.username = myclientkey ( configure the alias name of your key
in your keystore) as e.g. Above myclientalias
security.signature.password = ckpass ( configure the password of your key in
your keystore) as e.g. Above myclientkeypass
```

2. Edit the properties file defined in the security.signature.properties = file:\${tesb.home}/etc/keystores/clientKeystore.properties parameter of the etc/org.talend.esb.job.client.cfg configuration file as follows:

```
org.apache.ws.security.crypto.merlin.keystore.type=jks
org.apache.ws.security.crypto.merlin.keystore.password=cspass ( as eg above
myclientstorepass)
org.apache.ws.security.crypto.merlin.keystore.alias= myclientalias (as eg above
myservicealias)
org.apache.ws.security.crypto.merlin.keystore.file=./etc/keystores/
mykeystore.jks (location of the myclientstore.jks file)
```

#### **For the Service provider configuration:**

1. Edit the etc/org.talend.esb.job.service.cfg configuration file:

```
security.signature.properties = file:${tesb.home}/etc/keystores/
serviceKeystore.properties (as eg above myservicestore.jks)
security.signature.username = myservicekey ( as e.g. Above myservicealias)
security.signature.password = skpass (as e.g. Above myservicekeypass)
```

2. Edit the properties file defined in the security.signature.properties = file:\${tesb.home}/etc/keystores/serviceKeystore.properties parameter of the etc/org.talend.esb.job.service.cfg configuration file as follows:

```
org.apache.ws.security.crypto.merlin.keystore.type=jks
org.apache.ws.security.crypto.merlin.keystore.password=sspass
(mysevicestorepass)
org.apache.ws.security.crypto.merlin.keystore.alias=myservicekey
(mysevicealias)
org.apache.ws.security.crypto.merlin.keystore.file=./etc/keystores/
servicestore.jks (myservicestore.jks)
```

## **8.4. Configuring encryption for multiple service providers on the same container**

The default configuration nfile used for service encryption is the following one: etc/org.talend.esb.job.service.cfg.

To encrypt a particular or mutliple service providers, create a new configuration file as follows: etc/org.talend.esb.job.service.[escapeURL{service\_qname}].cfg.

If the provider specific configuration is not provided, the default configuration file will be used.

Example:

For the following Service QName: *{http://services.talend.org/ReservationService}ReservationServiceProvider*

The URL escaped Service QName is as follows:  
`http_services_talend_org_ReservationService_ReservationServiceProvider`

So, its Provider specific configuration file should be named as follows:  
**`org.talend.esb.job.service.http_services_talend_org_ReservationService_ReservationServiceProvider.cfg`**





## Chapter 9. Using STS with the Talend Runtime

This chapter describes the deployment and configuration of STS with a Talend Runtime container, how to configure the Data Services to use the STS. It also discusses creating keys and certificates for STS and clients.



The term `<TalendRuntimePath>` is used for the directory where Talend Runtime is installed. This is typically the full path of either `Runtime_ESBSE` or `Talend-ESB-V`, depending on the version of the software that is being used. Please substitute appropriately.

## 9.1. Deploying the STS into the Talend Runtime container



*For production use, the sample keys used here will need to be replaced with your project's own keys, usually signed by a third-party CA.*

To enable Security Token Service in the Talend Runtime, you need to deploy it into a Talend Runtime container:

1. Replace the STS' sample keystore/truststore called `stsstore.jks` located in the `<TalendRuntimePath>/container/etc/keystores` folder with your own keystore. See [Security Token Service Configuration](#) for more information.
2. `cd <TalendRuntimePath>/container/bin` directory, enter **trun** to start Talend Runtime, a Talend Runtime container (Karaf) console window will open.
3. In the console, type **tesb:start-sts** to install the Security Token Service feature. Or type **feature:install tesb-sts** if you are using a generic Karaf container instead of Talend Runtime
4. Type **list | grep STS** in the console. You should see the following output:

```
ID      State      Blueprint  Spring      Level  Name
[ 203] [Active ] [          ] [started ] [ 60]  Apache CXF STS Core (2.5.0)
Fragments: 204
[ 204] [Resolved ] [          ] [        ] [ 60]  Talend :: ESB :: STS :: CONFIG
(6.4.1)
```

The above shows that the Security Token Service feature is enabled in the Talend Runtime container. The Fragment Bundle 204: Talend :: ESB :: STS :: CONFIG (6.4.1) provides the custom configuration about the Security Token Service, which will be described in [Security Token Service Configuration](#).

## 9.2. Deploying the STS into a Servlet Container (Tomcat)



*For production use, the sample keys used here will need to be replaced with your project's own keys, usually signed by a third-party CA.*

To enable Security Token Service using a servlet container (here Tomcat is used as an example), follow the below steps:

1. Extract the `<TalendRuntimePath>/add-ons/sts/SecurityTokenService.war` file and replace the `stsstore.jks` STS sample keystore/truststore with your own keystore. Alter the `stsKeystore.properties` file with any different configuration information based on your new keystore. Recompress the extracted WAR into a new WAR file.
2. Deploy the new WAR file created in the previous step into the Tomcat container.
3. Start Tomcat and open a browser with the follow url: `http://{tomcat}host:port/SecurityTokenService/`. You will see several Security Token Service available, such as Username Token service (UT), X.509 Token service, and so on.
4. Enter URL: `http://{tomcat host}:port/SecurityTokenService/UT?wsdl`, the displayed WSDL file will describe the details about the Security Token Service.

## 9.3. Security Token Service Configuration

The Security Token Service provides the following methods as described in the below snippet, which is defined in `SecurityTokenService.war/WEB-INF/wsdl/ws-trust-1.4-service.wsdl`

```
<wsdl:service name="SecurityTokenService">
  <wsdl:port name="UT_Port" binding="tns:UT_Binding">
    <soap:address location=
      "http://localhost:8080/SecurityTokenService/UT" />
  </wsdl:port>
  <wsdl:port name="X509_Port" binding="tns:X509_Binding">
    <soap:address location=
      "http://localhost:8080/SecurityTokenService/X509" />
  </wsdl:port>
  <wsdl:port name="Transport_Port" binding="tns:Transport_Binding">
    <soap:address location="/Transport" />
  </wsdl:port>
  <wsdl:port name="UTEncrypted_Port" binding="tns:UTEncrypted_Binding">
    <soap:address location="/UTEncrypted" />
  </wsdl:port>
</wsdl:service>
```

As above snippet shows, the Security Token Service can issue (or validate) UserName Token or X509 Token, and so on.

In Talend Runtime container, the configuration of Security Token Service can be defined in the file `<TalendRuntimePath>/etc/org.talend.esb.sts.server.cfg`:

```
stsServiceUrl=/SecurityTokenService/UT
stsX509ServiceUrl=/SecurityTokenService/X509
loginModule=TIDM
jaasContext=karaf
signatureProperties=file:${tesb.home}/etc/keystores/stsKeystore.properties
signatureUsername=mystskey
bspCompliant=false
tidmServiceUrl=http://localhost:8080/syncope/cxf/
tidmUsername=admin
tidmPassword=password
useMessageLogging=false
samlTokenLifetime=1800
```

By default, Security Token Service is configured to use JAAS interface to verify the user credentials and perform authentication. As shown above, Security Token Service uses `karaf` JAAS Context which is the default context configured for Talend Runtime container and uses `PropertiesLoginModule` of Karaf. This login module uses the file located in `/etc/users.properties` which contains a list of users and their password, hence the users which are needed to be authenticated via the Security Token Service should be listed here. A different login module can be configured for the Security Token Service by updating the `jaasContext` parameter in the above configuration. A Talend Runtime container comes with several login modules that can be used to integrate into your environment, the modules are listed below:

- `PropertiesLoginModule`
- `OsgiConfigLoginModule`
- `JDBCLoginModule`
- `LDAPLoginModule`

The `signatureProperties` file, which is located in: `/etc/keystores/stsKeystore.properties`, defines the signature configuration as shown below:

```
org.apache.ws.security.crypto.provider=org.apache.ws.security.components.cr
ypto.Merlin
```

```
org.apache.ws.security.crypto.merlin.keystore.type=jks
org.apache.ws.security.crypto.merlin.keystore.password=stsspass
org.apache.ws.security.crypto.merlin.keystore.alias=mystskey
org.apache.ws.security.crypto.merlin.keystore.file=./etc/keystores/stsstore.jks
```

The keystore file name can be changed by altering its value in the `stsKeystore.properties` file. With the default configuration as shown above, the Talend Runtime container will expect the STS' private key to have the alias of `mystskey`, this can be changed by altering the `alias` and `signatureUsername` values in the two configuration files listed above.

The `samlTokenLifetime` property in the `<TalendRuntimePath>/etc/org.talend.esb.sts.server.cfg` file allows you to set the lifetime of the SAML token. The default is 1800 seconds. You can change it as needed.

## 9.4. Setting up the security management system in Security Token Service

The Security Token Service is provided with all versions of Talend ESB, however the security management system behind it is different in the community and in the subscription version. For the community version, Talend Open Studio for ESB, the security service is managed via the JAAS authentication handler, whereas for the subscription version, Talend ESB, the security service is, by default, managed by the Talend Identity and Access Management, based on Apache Syncope. The use of the JAAS is also possible, within Talend ESB, by switching the module used from Talend Identity and Access Management to JAAS.

So, if you are using the subscription version of Talend ESB, you are able to either use the Talend Identity and Access Management or the JAAS security management systems. To switch between those two systems, you have to change the `loginModule` value in the `<TalendRuntimePath>/container/etc/org.talend.esb.sts.server.cfg` configuration file:

- To use Talend Identity and Access Management, set the parameter as follows: `loginModule=TIDM`. You also need to set the `tidmServiceUrl`, `tidmUsername`, and `tidmPassword` properties in the configuration file. For more information about how to install the Talend Identity and Access Management, see the *Talend Installation Guide*.
- To use JAAS, set the parameter as follows: `loginModule=JAAS`.

This way, when executing the **tesb:start-sts** command, Talend Runtime container checks which module is used and then installs either the **tesb-sts** or the **tesb-sts-tidm** feature. If the `loginModule` property does not exist, by default, **tesb-sts-tidm** will be installed.

To switch from one security management system to the other, simply use the following commands:

- **tesb:switch-sts-jaas**

If the Security Token Service is not started yet, this command only changes the configuration file property to `loginModule=JAAS`.

If the Security Token Service using the Talend Identity and Access Management is started, this command stops it and starts the JAAS module instead.

- **tesb:switch-sts-tidm**

If the Security Token Service is not started yet, this command only changes the configuration file property to `loginModule=TIDM`.

If the Security Token Service using the JAAS module is started, this command stops it and starts the Talend Identity and Access Management instead.



## 9.5. Setting up logging parameters in Security Token Service

If you are using Talend Identity and Access Management with STS to manage authorization accesses to services in Talend ESB (only available in Talend subscription products), you can decide whether to log the communication between the modules involved.

To do so, you have to edit the following configuration files:

- In `<TalendRuntimePath>/container/etc/org.talend.esb.sts.server.cfg`, set `useMessageLogging=true` to indicate whether the communication between STS and Talend Identity and Access Management should be logged. Dynamic reconfiguration at runtime is supported.

By default, the option is disabled: `useMessageLogging=false`.

- In `<TalendRuntimePath>/container/etc/org.talend.esb.authorization.pdp.cfg`, set `useMessageLogging=true` to indicate whether the communication between PDP and the Talend Authorization XACML Repository should be logged. Dynamic reconfiguration at runtime is supported.

By default, the option is disabled: `useMessageLogging=false`.

## 9.6. Data Service Configuration for using STS

In the Talend Runtime container, the configuration used by Data Service Consumers for using Security Token Service (STS) can be defined in the file: `<TalendRuntimePath>/container/etc/org.talend.esb.job.client.sts.cfg`.

```
#STS endpoint configuration
sts.wsdl.location = \
    http://localhost:8040/services/SecurityTokenService/UT?wsdl
sts.namespace = http://docs.oasis-open.org/ws-sx/ws-trust/200512/
sts.service.name = SecurityTokenService
sts.endpoint.name = UT_Port

#STS properties configuration
security.sts.token.username = myclientkey
security.sts.token.usecert = true
ws-security.is-bsp-compliant = false
security.sts.token.properties = \
    file:${tesb.home}/etc/keystores/clientKeystore.properties
```

The STS endpoint used by the consumer is defined by `sts.wsdl.location`. This configuration should be changed in case the STS service is running on a different host and port. The keystore configuration described above is used for signing the timestamp sent in the request by the consumer to the provider. The Talend ESB-supplied sample keystores and certificates above are not meant for production use. Be sure to use your own keys (with different passwords) and configure them as discussed below.

A Data Service consumer can use two types of authentication mechanisms: **Username token** and **SAML token**.

- When using **Username token**, the consumer sends the credentials as a part of the request to the provider and authentication is performed on the provider side. The policy used by the consumer for Username token authentication is defined in the file `<TalendRuntimePath>/etc/org.talend.esb.job.token.policy`.
- For **SAML token**, the consumer makes a SAML token issue request to the STS passing its credentials and on successful authentication the STS issues a SAML token. This SAML token is sent as a part of the request to the provider and the provider verifies the validity of the SAML token. The policy used by the consumer for SAML token authentication is defined in the file `<TalendRuntimePath>/etc/org.talend.esb.job.saml.policy`.

When using **Username token**, a Data Service provider receives credentials from the consumer and performs authentication locally. By default a Data Service provider is configured with JAAS authentication handler and uses the default JAAS context `karaf` configured for the Talend Runtime container. The login module configured for this context uses the file located in `/etc/users.properties`, which contains a list of users and their password. Thus, the user which needs to be authenticated should be listed here.

In the case of a SAML token, the provider locally verifies the integrity of the token using a certificate, the configuration for it is defined in the file `<TalendRuntimePath>/etc/org.talend.esb.job.service.cfg`.

```
security.signature.properties = \  
    file:${tesb.home}/etc/keystores/serviceKeystore.properties  
security.signature.username = myservicekey  
security.signature.password = skpass
```

## 9.7. Creating keys for the Security Token Service

This section describes how to create keys for the Security Token Service. We highly recommend that you use third-party signed CA's (certificate authorities) or create your own Certificate Authority, but the following instructions can be used to create self-signed keys.

### 9.7.1. Using OpenSSL to create certificates

First, create the keys.



*Replace "`<PW-Sk>`", "`<PW-Sk>`", "`<PW-Cs>`" and "`<PW-Ck>`" in the example below with your own passwords.*

#### 9.7.1.1. Creating the service keystore

Note: given the **rm** commands below, it is probably best to create a new directory and navigate to it before running these commands from a terminal window.

```
rm *.p12 *.pem *.jks *.cer  
openssl req -x509 -days 3650 -newkey rsa:1024 -keyout servicekey.pem -out  
    servicecert.pem -passout pass:<PW-Sk>
```

When running this **openssl** command, enter any geographic and company information desired, the key password in passout, and a common name of your choice (perhaps `servicecn` for the service and `clientcn` for the client).

```
openssl pkcs12 -export -inkey servicekey.pem -in servicecert.pem -out  
    service.p12 -name myservicekey -passin pass:<PW-Sk> -passout  
    pass:<PW-Sk>
```

This creates a pkcs12 certificate. Note the `<PW-Sk>` value will be used both for the keystore and the private key itself.

```
keytool -importkeystore -destkeystore servicestore.jks -deststorepass  
    <PW-Sk> -deststoretype jks -srckeystore service.p12 -srcstorepass  
    <PW-Sk> -srcstoretype pkcs12 # See Note 3
```

This places the certificate in a new JKS keystore. The keystore's password is changed here to `<PW-Sk>`, but the private key's password retains the earlier value of `<PW-Sk>`. Also note we're using Java 6 instead of Java 5 `keytool` commands (see [changes](#) between the two.)

```
keytool -list -keystore servicestore.jks -storepass <PW-Sk> -v
```

The list command is just to show the keys presently in the keystore.

```
keytool -exportcert -alias myservicekey -storepass <PW-Sk> -keystore
  servicestore.jks -file service.cer
keytool -printcert -file service.cer
rm *.pem *.p12
```

### 9.7.1.2. Creating the client keystore

```
openssl req -x509 -days 3650 -newkey rsa:1024 -keyout clientkey.pem
  -out clientcert.pem -passout pass:<PW-Cs>
openssl pkcs12 -export -inkey clientkey.pem -in clientcert.pem
  -out client.p12
  -name myclientkey -passin pass:<PW-Cs> -passout pass: <PW-Ck>
keytool -importkeystore -destkeystore clientstore.jks -deststorepass
  <PW-Cs> -deststoretype jks -srckeystore client.p12
  -srcstorepass <PW-Ck> -srcstoretype pkcs12
keytool -list -keystore clientstore.jks -storepass <PW-Cs> -v
keytool -exportcert -alias myclientkey -storepass <PW-Cs> -keystore
  clientstore.jks -file client.cer
keytool -printcert -file client.cer
rm *.pem *.p12
```

## 9.7.2. Deploying and Using a Security Token Service (STS)

You have created the service and client keystores as in the previous section. Now create the STS keystore as follows:



Replace <PW-Ts>, <PW-Tk> in the example below with your own passwords.

```
openssl req -x509 -days 3650 -newkey rsa:1024 -keyout stskey.pem -out
  stscert.pem -passout pass:<PW-Ts>
openssl pkcs12 -export -inkey stskey.pem -in stscert.pem -out sts.p12
  -name mystskey -passin pass:<PW-Ts> -passout pass:<PW-Tk>
keytool -importkeystore -destkeystore stsstore.jks -deststorepass <PW-Ts>
  -srckeystore sts.p12 -srcstorepass <PW-Tk> -srcstoretype pkcs12
keytool -list -keystore stsstore.jks -storepass <PW-Ts>
keytool -exportcert -alias mystskey -storepass <PW-Ts> -keystore
  stsstore.jks -file sts.cer
keytool -printcert -file sts.cer
rm *.pem *.p12
```

To fix any issues with fixed paths to the keystore and truststore locations within the WSDLs, the source code download uses Maven resource filtering to allow for a relative path to the project base directory to be used instead.

Next, the service keystore will need to have the STS public key added so it trusts it, and vice-versa. Also, the client will need to have the STS' and WSP's certificates added to its truststore, as it relies on symmetric binding to encrypt the SOAP requests it makes to both:

```
keytool -keystore servicestore.jks -storepass <PW-Sk> -import -noprompt
  -trustcacerts -alias mystskey -file sts.cer
keytool -keystore stsstore.jks -storepass <PW-Ts> -import -noprompt
  -trustcacerts -alias myservicekey -file service.cer
keytool -keystore clientstore.jks -storepass <PW-Cs> -import -noprompt
```

```
-trustcacerts -alias mystskey -file sts.cer  
keytool -keystore clientstore.jks -storepass <PW-Cs> -import -noprompt  
-trustcacerts -alias myservicekey -file service.cer
```

If you plan on using X.509 authentication of the WSC to the STS (instead of UsernameToken), the former's public key will need to be in the latter's truststore. This can be done with the following commands:

```
keytool -exportcert -alias myclientkey -storepass <PW-Cs> -keystore  
clientstore.jks -file client.cer  
keytool -keystore stsstore.jks -storepass <PW-Ts> -import -noprompt  
-trustcacerts -alias myclientkey -file client.cer
```

Since the service does not directly trust the client (the purpose for our use of the STS to begin with), we will not add the client's public certificate to the service's truststore as normally done with message-layer encryption.



## Chapter 10. Provisioning Service

This chapter describes the steps to install and run the Provisioning Service. The Provisioning Service is a technical service that helps maintain consistency for your resources and configurations throughout all your Talend Runtime containers. It is used in Talend ESB to distribute feature descriptions and configuration resources throughout several Talend Runtimes, and potentially in Apache Tomcat based application server which are used for ESB Java based Consumer and Provider. However for Apache Tomcat, only the use of resources and placeholders are possible, not the use of features. For more information on what are resources, placeholders and features for the Provisioning Service, see [Introduction](#).

When using the Provisioning Service, you will be able to:

- distribute several copies of one and the same configuration artifact over different runtimes.
- reuse the same configuration artifacts.
- consistently update the configurations already in place over all runtimes.
- get an overview of the current configuration settings in the different runtimes.
- enforce consistent authorization for changing configuration artifacts in the runtimes, auditing or logging.

The Provisioning Service supports the lookup of configuration artifacts required by the runtime for the different artifact types. To support this, the Provisioning Services provides a lookup interface which the distributed agents will use to pull for new updates. The Service itself does not do any updates, it only manages the central configuration while the agents apply changes to the local container.

This functionality is available with Talend ESB; it is not included in the Talend Open Studio for ESB.

A User Interface is also available in Talend Administration Center, the **Provisioning** page, to manage the artifacts provided by the Provisioning Service which also enables to browse through the artifacts. For more information, see the *Talend Administration Center User Guide*.

## 10.1. Introduction

Talend Provisioning Service allows you to distribute resources and features in several containers via profiles. Two types of profiles are available: System profile, to distribute configuration artifacts required by the system, and Application profile, to distribute configuration artifacts required by applications.

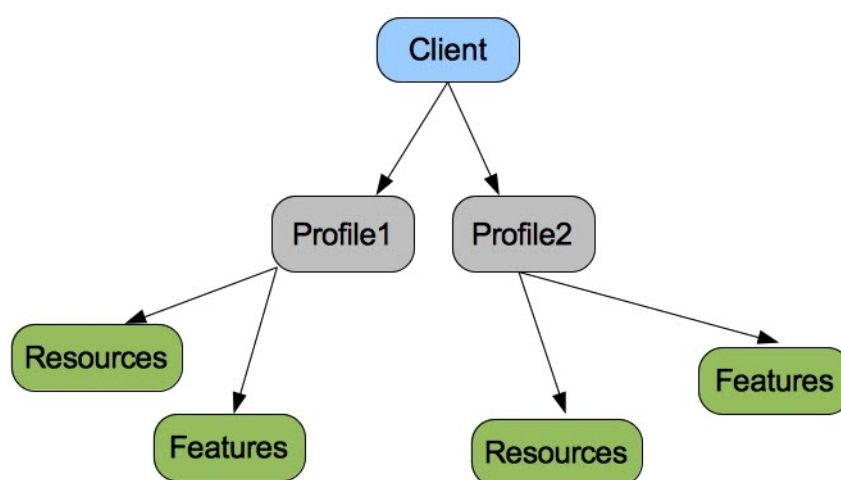
With the Provisioning Service, you can distribute configuration files, activate several services in several containers by applying profiles to them.

In Provisioning Service, two different profiles are available:

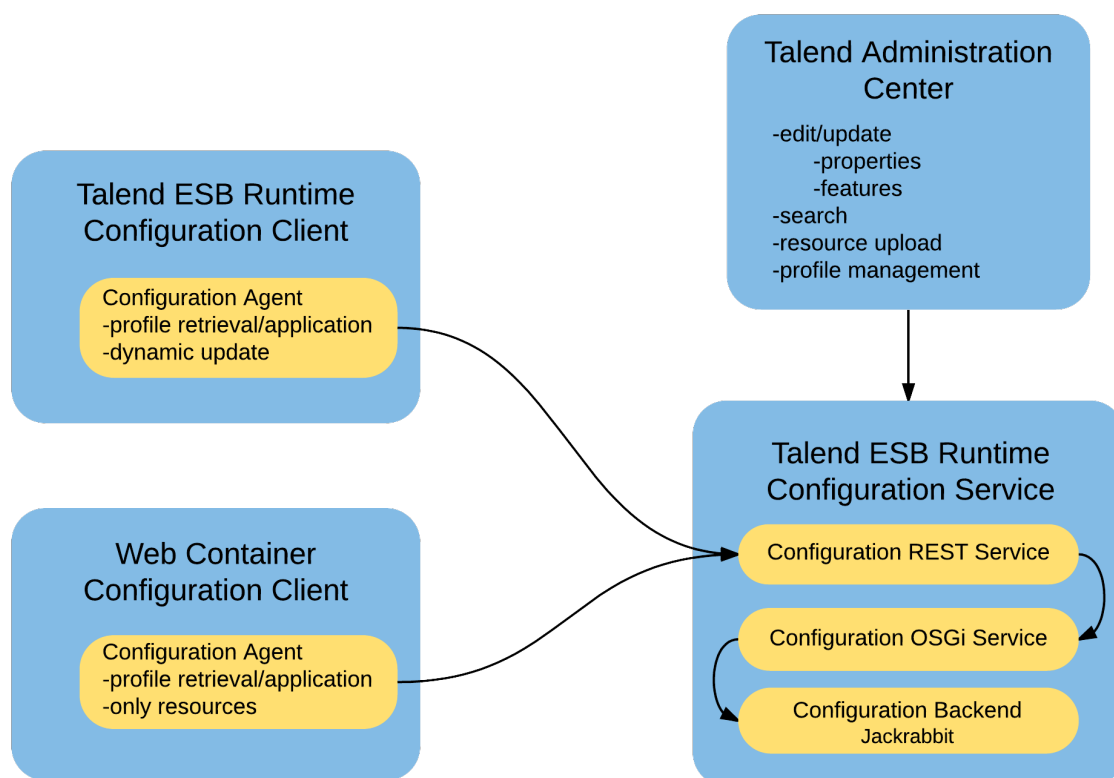
- System profile: a collection of configuration artifacts required by the system, for example, to configure the ESB services in the different Talend Runtime container.
- Application profile: a collection of configuration artifacts required by an application.

Each profile is made of:

- resources, that correspond to a list of destination paths. And for each path corresponds a list of resource files.
- features, that correspond to a list of features, with the following information: feature name, feature URL and Maven feature version.
- placeholders, that correspond to a list of variables, defined via the syntax `#{<placeholder>}` that, combined with resources, can help you dynamically configure your systems or applications.



## 10.1.1. Architecture diagram



Two types of clients are supported by the Talend Provisioning Service: Talend Runtime containers and Web Containers (only for configuration files via resources, and placeholders). A configuration agent accessing the central Provisioning Lookup Service are responsible for:

- Retrieval of the initial configuration corresponding to the configured configuration profiles consisting of resources and features (OSGI only). This happens in the bootstrap phase so that the profile is applied to the starting client container.
- Dynamic update of configuration property files (which are viewed as resources) for the OSGi Runtime. The local OSGi configuration administration service will trigger the configuration update method of the managed services. If dynamic update is enabled for a profile, the configuration agent will poll the central Provisioning Service for a new version of the profile, and then trigger the download/update if necessary.

An administrator accesses and uses Talend Administration Center to edit and update resources and features, and releases profiles and placeholders. If the administrator releases a new version of a profile or placeholder, they should increase the version number of the released profile or placeholder.

## 10.2. Installing and running the Provisioning Service

By default, the Provisioning Service is embedded as an OSGi feature in the Talend Runtime container, so to install it you just need to configure and start it into the container:

1. Start the container:

- **container/bin/trun.bat** for Windows,
- **container/bin/trun** for Linux.

2. Start the Provisioning Service.

If you started all the Talend Infrastructure Services via the **tesb:start-all** command, the Provisioning Service (server and agent) will be started automatically as well.

To only start the Provisioning Service, use the following commands:

1. `tesb:start-provision-server`

To start the Provision server that is used to manage profiles, features, resources, and placeholders.

2. `tesb:start-provision-agent`

To start the Provision agent, used by clients to apply the available profiles.

3. To check that the bundles have been correctly installed on the container, type the following command:

```
list | grep Provisioning
```

The following bundles will be listed:

```
269 | Active | 80 | 6.4.1 | Talend ESB Provisioning :: Common
270 | Active | 80 | 6.4.1 | Talend ESB Provisioning :: Server :: Core
275 | Active | 80 | 6.4.1 | Talend ESB Provisioning :: Agent
276 | Active | 80 | 6.4.1 | Talend ESB Provisioning :: Agent :: Commands
271 | Active | 80 | 6.4.1 | Talend ESB Provisioning :: Server :: Commands
272 | Active | 80 | 6.4.1 | Talend ESB Provisioning :: Server :: Admin REST
Service
273 | Active | 80 | 6.4.1 | Talend ESB Provisioning :: Server :: Lookup REST
Service
273 | Active | 80 | 6.4.1 | Talend ESB Provisioning :: Server :: Lookup REST
Service :: Client
```

## 10.3. Stopping the the Provisioning Service

If you need to stop the Provisioning Service, run the following commands:

1. `tesb:stop-provision-agent`

2. `tesb:stop-provision-server`

## 10.4. Starting the Provisioning Web agent

Apart from Talend Runtime containers, the Provisioning agent works on Apache Tomcat as well, but only for profiles made of resources (configuration files) and placeholders.

To install and start the Provisioning Web agent:

1. Copy the following file from Talend Runtime container:

```
add-ons/provisioning/provisioning-web-agent-6.4.1.war
```

to `<TomcatPath>/webapps` to deploy it to Tomcat.



2. Launch the Tomcat server.

The Provisioning Web agent is accessible at:

`http://localhost:8080/provisioning-web-agent-6.4.1/web-agent`

3. If the Provisioning server is installed on a different machine, change its URL in the `webapps/provisioning-web-agent-6.4.1/WEB-INF/web.xml` file.

```
<context-param>
  <param-name>provision.server.url</param-name>
  <param-value>localhost:8040/services/provision/lookup</param-value>
</context-param>
```

## 10.5. Managing profiles via Provisioning commands

Once installed and running, the Provisioning Service gives you access to a list of commands helping you manage your profiles directly from the Talend Runtime container:

271 | Active | 80 | 6.4.1 | Talend ESB Provisioning :: Server :: Commands

Type **tprovision:** and press the **TAB** keyboard to access them:

```
tprovision:export
tprovision:feature-delete
tprovision:features-list
tprovision:placeholder-create
tprovision:placeholder-update
tprovision:placeholders-export
tprovision:placeholders-list
tprovision:profile-create
tprovision:profile-info
tprovision:profile-update
tprovision:resource-create
tprovision:resource-export
tprovision:resources-list
tprovision:feature-create
tprovision:feature-update
tprovision:import
tprovision:placeholder-delete
tprovision:placeholders-categories-list
tprovision:placeholders-import
tprovision:placeholders-release
tprovision:profile-delete
tprovision:profile-release
tprovision:profiles-list
tprovision:resource-delete
tprovision:resource-update
tprovision:service-help
```

### 10.5.1. Provisioning Service Commands

Commands: **tprovision:**

Command	Parameter	Usage
<b>profile-create</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> <li>• <b>profile-category</b> (for example: "system")</li> </ul> <b>Optional:</b> <ul style="list-style-type: none"> <li>• <b>auto-update</b>: autoupdate in seconds</li> <li>• <b>pool-interval</b>: poolinterval in seconds</li> </ul>	Primary command to create a new profile
<b>profile-delete</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> </ul>	Deletes the entire profile (including all its existing versions).
<b>profile-info</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> </ul> <b>Optional:</b>	Shows information about the specified Profile. If the version is given, it shows the information for a specific version of the profile.

Command	Parameter	Usage
	<ul style="list-style-type: none"> <li>• <b>profile-version</b>, with format ('&lt;number&gt;.&lt;number&gt;')</li> </ul>	
<b>profile-release</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> <li>• <b>release-version</b>, with format ('&lt;number&gt;.&lt;number&gt;')</li> </ul> <p><b>Optional:</b></p> <ul style="list-style-type: none"> <li>• <b>placeholders-category</b></li> <li>• <b>placeholders-version</b></li> </ul>	<p>Releases a Profile (at the same time a placeholder category and version can also be set to replace placeholders used in this profile by the related values at lookup time).</p> <p>Only released profiles can be used by agents. If autoupdate is activated for a profile, agents will automatically start to apply the new version once it is released.</p>
<b>profile-update</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> <li>• <b>auto-update</b></li> </ul> <p><b>Optional:</b></p> <ul style="list-style-type: none"> <li>• <b>pooling-interval</b></li> <li>• <b>new-profile-name</b></li> <li>• <b>placeholders-category</b></li> </ul>	Can be used to update the optional parameters of the profile: auto-update, pool-interval and placeholders-category.
<b>profiles-list</b>	<p><b>Optional:</b></p> <ul style="list-style-type: none"> <li>• <b>profile-name</b></li> </ul>	<p>Lists all the profiles existing within the server.</p> <p>Note: It is not the list of applied profiles. For this, see the agent commands</p>
<b>feature-create</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> <li>• <b>feature-url</b></li> <li>• <b>feature-name</b></li> <li>• <b>feature-version</b></li> </ul> <p><b>Optional:</b></p> <ul style="list-style-type: none"> <li>• <b>feature-index</b>: provide a way to define the order in which features within one profile will be applied.</li> </ul>	<p>Adds a feature definition to the profile. Features are supposed to be accessible by the container where the agent is running using a local system repository (as Talend Runtime provides it ) or by an external Maven repository (for example: the Talend Nexus Artifact Repository)</p> <p>The feature name, URL and version is the standard Apache Karaf /Maven information for features.</p> <p>Profiles with features can only be applied to Talend Runtime (OSGi).</p>
<b>feature-delete</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> <li>• <b>feature-index</b></li> </ul>	Deletes a feature from the profile, using the index to specify which feature.
<b>feature-update</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> <li>• <b>feature-index</b></li> <li>• <b>new-feature-url</b></li> <li>• <b>new-feature-name</b></li> <li>• <b>new-feature-version</b></li> </ul>	Allows to change the feature which the given index to a new feature name, URL, version
<b>features-list</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> </ul> <p><b>Optional:</b></p> <ul style="list-style-type: none"> <li>• <b>profile-version</b></li> </ul>	Lists all features for a given profile
<b>resource-create</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> <li>• <b>local-file-path</b></li> </ul> <p><b>Optional:</b></p> <ul style="list-style-type: none"> <li>• <b>resource-uri</b></li> </ul>	<p>Provides the ability to upload a local file into a profile.</p> <p>The URI specifies a folder name under which this resource should be placed on the final agent container.</p> <p>A resource can be essentially any kind of file, for example: .cfg file, .txt File, Keystore file and more.</p>

Command	Parameter	Usage
		The URI has two meanings: the unique name of a resource within the profile, but at the same time, it is also the relative path + file name of the resource.
<b>resource-delete</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> <li>• <b>resource-uri</b></li> </ul>	Removes a resource from the profile with the given URI.
<b>resource-export</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> <li>• <b>resource-uri</b></li> <li>• <b>local-file-path</b></li> </ul> <b>Optional:</b> <ul style="list-style-type: none"> <li>• <b>profile-version</b></li> </ul>	Exports a profile resource into a file (released under specified version or sandbox one if no version specified)
<b>resource-update</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> <li>• <b>resource-uri</b></li> <li>• <b>new-resource-uri</b></li> </ul> <b>Optional:</b> <ul style="list-style-type: none"> <li>• <b>new-local-file-path</b></li> </ul>	Allows to update a resource by profileName / resourceUri with a new Uri and new file content. It is also possible to just update the URI (e.g. the path / file name ) but not the content. The newResourcePath is optional and only used if also the content should be updated.
<b>resources-list</b>	<ul style="list-style-type: none"> <li>• <b>profile-name</b></li> </ul> <b>Optional:</b> <ul style="list-style-type: none"> <li>• <b>profile-version</b></li> </ul>	Lists all resources of the given profile.
<b>placeholder-create</b>	<ul style="list-style-type: none"> <li>• <b>placeholders-category</b></li> <li>• <b>placeholder-name</b></li> <li>• <b>placeholder-value</b></li> </ul>	<p>Placeholders are key / value pairs grouped together with the category (for example: com.talend.esb.infra.placeholder)</p> <p>One placeholder category (group) can be applied to one profile and allows to replace values in text based configuration files at lookup time. (For example: host name , users)</p>
<b>placeholder-delete</b>	<ul style="list-style-type: none"> <li>• <b>placeholders-category</b></li> </ul> <b>Optional:</b> <ul style="list-style-type: none"> <li>• <b>placeholder-name</b></li> </ul>	Deletes either an entire placeholder category (including all its versions) or a specific placeholder via the placeholder-name parameter, within the given category.
<b>placeholder-update</b>	<ul style="list-style-type: none"> <li>• <b>placeholders-category</b></li> <li>• <b>placeholder-name</b></li> <li>• <b>placeholder-value</b></li> </ul> <b>Optional:</b> <ul style="list-style-type: none"> <li>• <b>placeholder-version</b></li> </ul>	Updates an existing placeholder with a new value.
<b>placeholders-categories-list</b>	No parameter	Lists placeholders categories.
<b>placeholders-export</b>	<ul style="list-style-type: none"> <li>• <b>local-file-path</b></li> </ul> <b>Optional:</b> <ul style="list-style-type: none"> <li>• <b>placeholders-category</b></li> <li>• <b>placeholders-version</b></li> </ul>	Exports all placeholders or the one from the specified category / version to a local file.
<b>placeholders-import</b>	<b>Optional:</b> <ul style="list-style-type: none"> <li>• <b>-o</b> to overwrite existing values</li> <li>• <b>local-file-path</b></li> </ul>	Imports placeholders from a related local export file into the provisioning service, either all, or just the one for the specified category / version.

Command	Parameter	Usage
<b>placeholders-list</b>	<ul style="list-style-type: none"> <li><b>placeholders-category</b></li> </ul> <b>Optional:</b> <ul style="list-style-type: none"> <li><b>placeholders-version</b></li> </ul>	Lists all Key / Value pairs of a placeholder category (and version)
<b>placeholders-release</b>	<ul style="list-style-type: none"> <li><b>placeholders-category</b></li> <li><b>release-version</b></li> </ul>	Releases the set of key / value pairs for the given category with the given version.
<b>export</b>	<ul style="list-style-type: none"> <li><b>local-file-path</b></li> </ul> <b>Optional:</b> <ul style="list-style-type: none"> <li><b>profile-name</b></li> <li><b>profile-version</b></li> </ul>	Exports all profiles or just the given profile (version). By this, profiles can be prepared on a test system and imported into production or any other environment.
<b>Import</b>	<b>Optional:</b> <ul style="list-style-type: none"> <li><b>-o</b> to overwrite existing profiles if exists already</li> <li><b>local-file-path</b></li> </ul>	Imports profile(s) from a given profile export file.
<b>service-help</b>	No parameter	Shows a short online help of all tprovision: commands

## 10.5.2. Provisioning Agent Commands

Commands: **tprovision-agent:**

Command	Parameter	Usage
<b>apply-profiles</b>	<ul style="list-style-type: none"> <li><b>name</b></li> <li><b>version</b></li> </ul>	With this command, a new profile can be applied, activated, and used by the local container. Depending if the profile is set to auto-update mode or not it will be updated whenever a new version is release on the server automatically, or without the auto-update mode, by this command, with an explicit version specified.
<b>list-applied-profiles</b>	<b>Optional:</b> <ul style="list-style-type: none"> <li><b>-v</b> (to include profile versions in the list)</li> </ul>	Shows all profiles which are currently applied and active in the container.
<b>update-profiles</b>	<ul style="list-style-type: none"> <li><b>name</b></li> </ul> <b>Optional:</b> <ul style="list-style-type: none"> <li><b>version</b></li> </ul>	With this command a given profile can be updated either to the current release version or to an explicit release version.
<b>help</b>	No parameter	Shows a short help for all tprovision-agent: commands.

## 10.6. Use cases

In this section, you will find use cases for the two types of profile: application profile and system profile.

The application profile use case presented to you will show you how you can deploy the same feature, in different contexts, onto one or several of your containers.

The system profile use case will show you how to change your STS settings in one, several or even all your containers at once, and even in Tomcat.

## 10.6.1. Application profile use case

In this use case, you will see how to add STS credentials to a service consumer, the **SayHelloConsumer**, while deploying it to a container via the Provisioning Service feature.

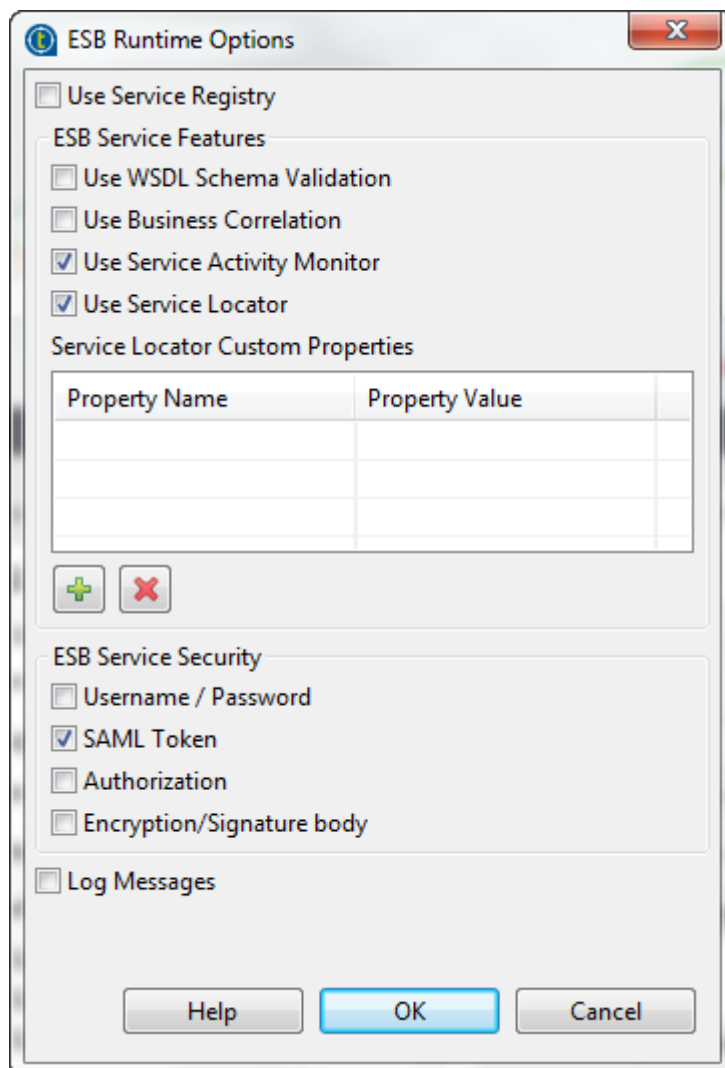
So for this use case, you need to activate the STS Infrastructure service in the container (either by using the **tesb:start-sts** or even the **tesb:start-all** that starts all the default Talend Infrastructure services). For demo purpose, you need to change the security management system from default Talend Identity Management Service to the JAAS authentication handler, which is easier to use, and does not require the installation of additional software (via the **tesb:switch-sts-jaas**). In that same container, you need to deploy the service that will be called by the consumer, the **SayHelloService**.

### 10.6.1.1. Preparing the SayHello use case for Provisioning

For both **SayHelloService** and **SayHelloConsumer**, you have to enable the STS option.

For the **SayHelloService**, in the studio:

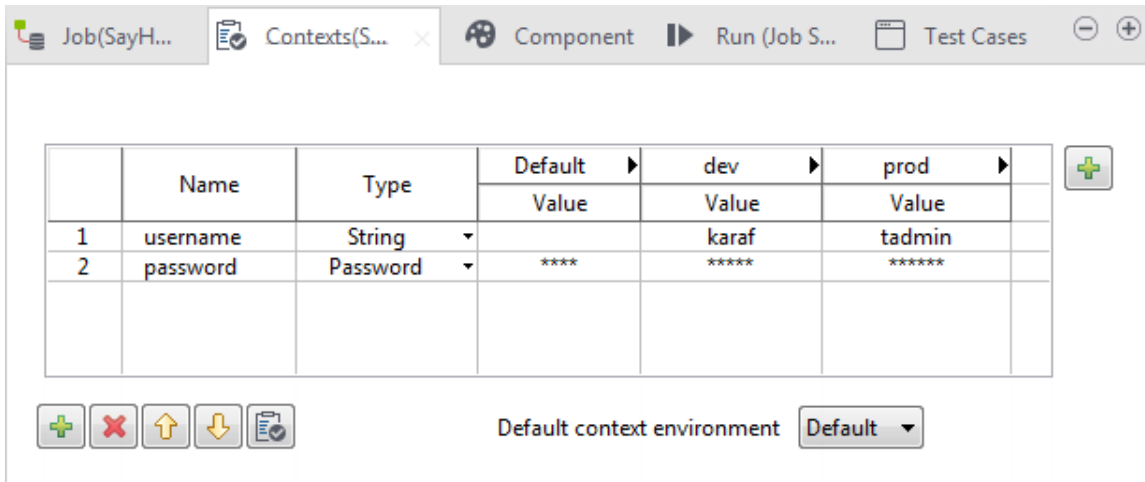
1. Right-click it in the **Repository** and click **ESB Runtime Options**. In the wizard, select the **SAML Token** check box as you are using the JAAS authentication. Select the **Use Service Activity Monitor** and **Use Service Locator** check boxes.



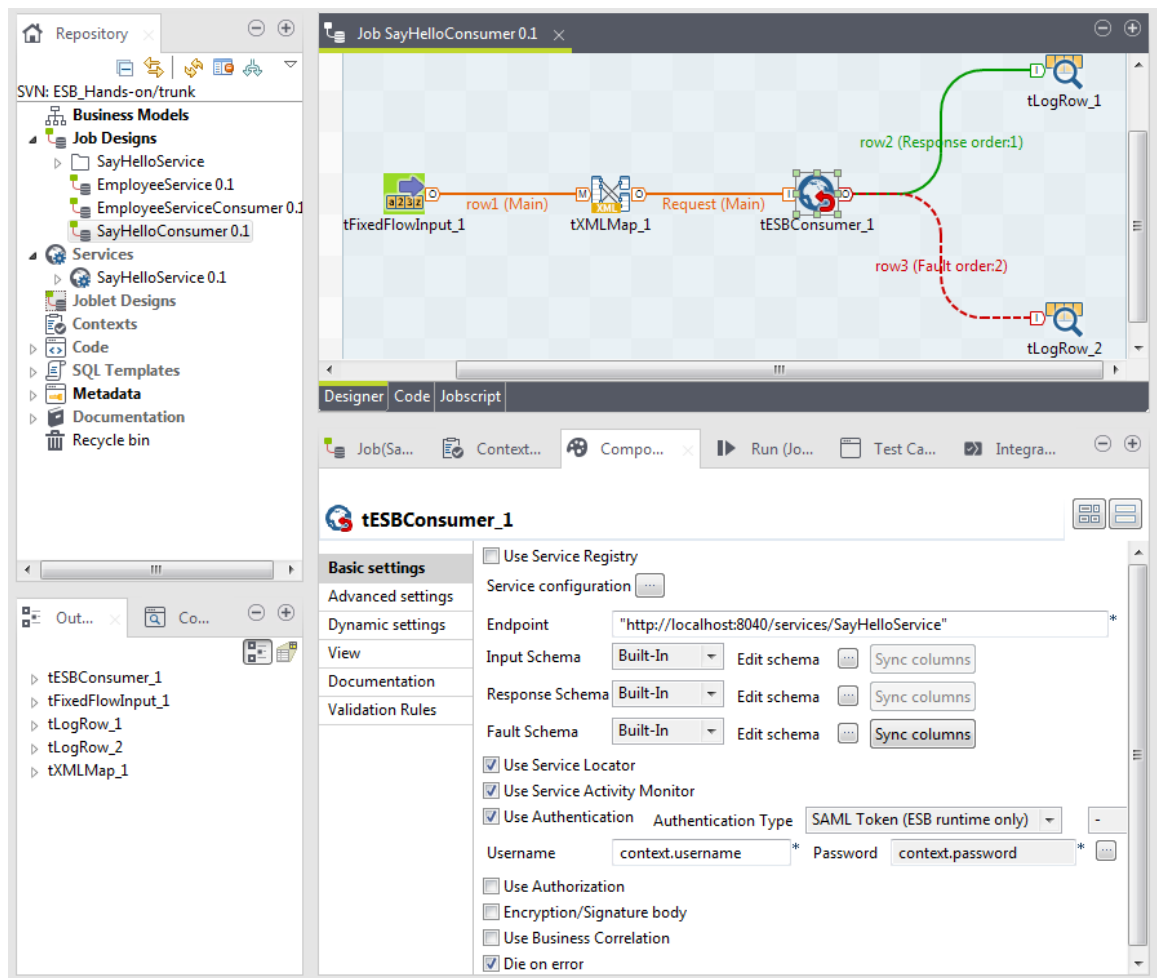
- Once STS is enabled for the **SayHelloService**, publish it in the Artifact Repository via the **Publisher** page of the **Talend Administration Center**. For more information on publishing services, see the *Talend Administration Center User Guide*.

For the **SayHelloConsumer**, in the studio, double-click the **Job Designs** to open it.

- Click the **Contexts** view at the bottom of the window to create your STS credentials as context variables depending on which environment you are going to deploy the consumer, as the STS credential might change depending on if you are in a development, testing or production environment. Here you will add STS credentials for a development environment:



- Click the green **[+]** at the right of the **Contexts** view, and in the **[Configure Contexts]** wizard, click the **New...** button to add a new context and name it *dev*. The same way, create another context named *prod*.
- Back to the **Context** view, click twice the green **[+]** at the bottom, to add two new context variables: *username* with **Type** as **String**, and *password* with the **Type Password**. Leave the **Default** variable values empty, and for the **dev** variable, put *karaf* for both username and password, and for **prod**, use *tadmin/tadmin*.
- Now, select the **tESBConsumer\_1** component in the **Designer** view, and click the **Component** view:





5. In the **Component** view, select the **Use Service Locator**, the **User Service Activity Monitoring** and finally the **Use Authentication** check boxes.
6. In the **Authentication Type** drop-down list, select **SAML Token** as for the service. A new drop-down list is displayed, select the **-** option.
7. In the **Username** field, press **Control+Space** to access all global and context variables, and select *context.username*.
8. In the **Password** field, press **Control+Space**, and select *context.password*.
9. Once STS is enabled for the **SayHelloConsumer**, publish it in the Artifact Repository via the **Publisher** page of the Talend Administration Center. For more information on publishing services, see the *Talend Administration Center User Guide*.

Deploy the **SayHelloService** and the **SayHelloConsumer** from the Talend Administration Center.

1. Once the **SayHelloService** and the **SayHelloConsumer** are published in the Artifact Repository, go to the **ESB Conductor** to deploy them to your Talend Runtime container(s).
2. In the **ESB Conductor**, click the **Add** button on the top, select **Task** in the list and fill in the **Conductor task** form that is displayed to the right:

## Conductor Task




 **Metadata**


Label:

SayHelloService

Description:

Tag:



 **Feature**

Feature:

Select Feature

Repository:


snapshots

URL:

mvn:org.example/SayHelloSer

Name:

SayHelloService-feature





Version:

0.1.0-SNAPSHOT

Type:


Service



 **Runtime Config**


Context:

Default




Server:


Talend Runtime Container



Property ID(PID):

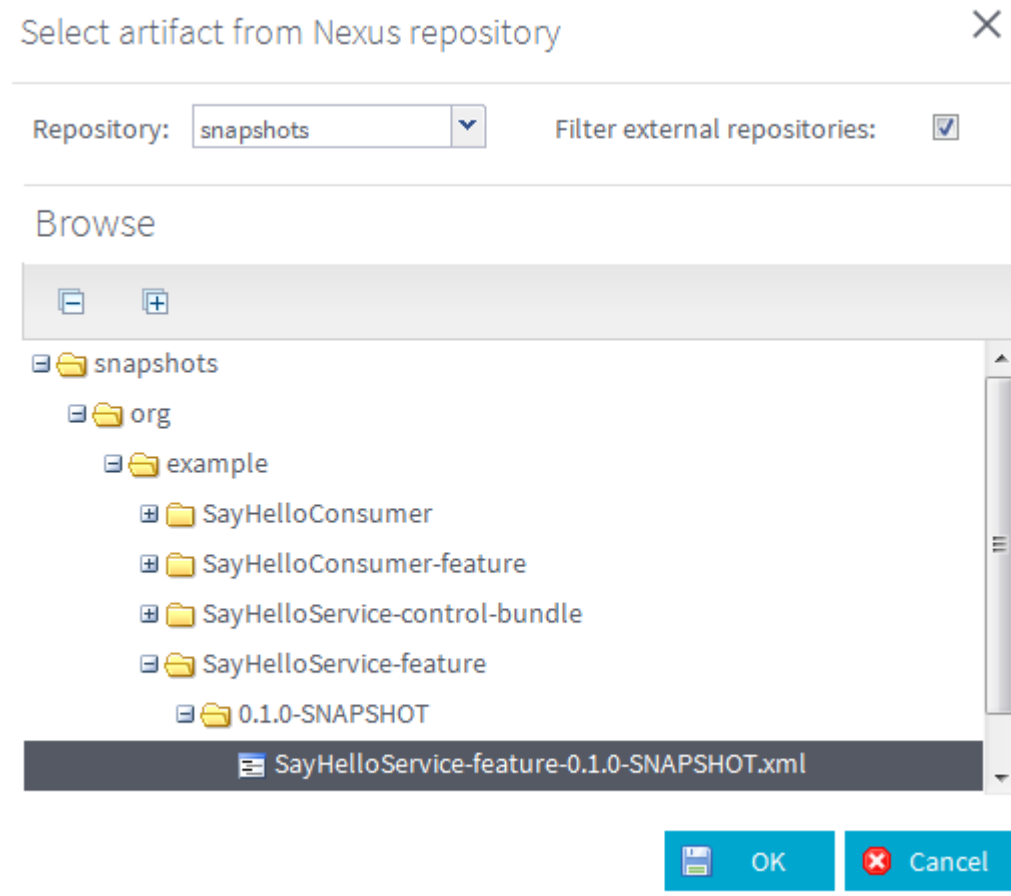
SayHelloService

 Save

 Cancel

- In the **Label** field of the **Metadata** area, name your service deployment task. Here: **SayHelloService** to deploy the SayHelloService.
- In the **Feature** area, click the **Select Feature** button to open the **[Select artifact from Nexus repository]** wizard that will help you fill in the following fields of this area.





5. In the **Repository** list, select **snapshots**, browse to the SayHello Service feature and select it. Click **OK** and all the Service feature information will be filled in in the form.
6. In the **Type** list, make sure **Service** is selected.
7. In the **Runtime Config** area, select the container on which to deploy the Service in the Server list. Click **Save** and the Service deployment task will be displayed in the list.
8. Click the **Deploy** button on the top and the Service will be deployed into the specified Talend Runtime container.
9. Repeat step 2 to 7 for the SayHelloConsumer this time, but select **Generic** in the **Type** list:

## Conductor Task ☰

**Metadata**

Label:

Description:

Tag:  ▼

**Feature**

Feature: Select Feature

Repository:

URL:

Name:  ▼

Version:

Type:  ▼

**Runtime Config**

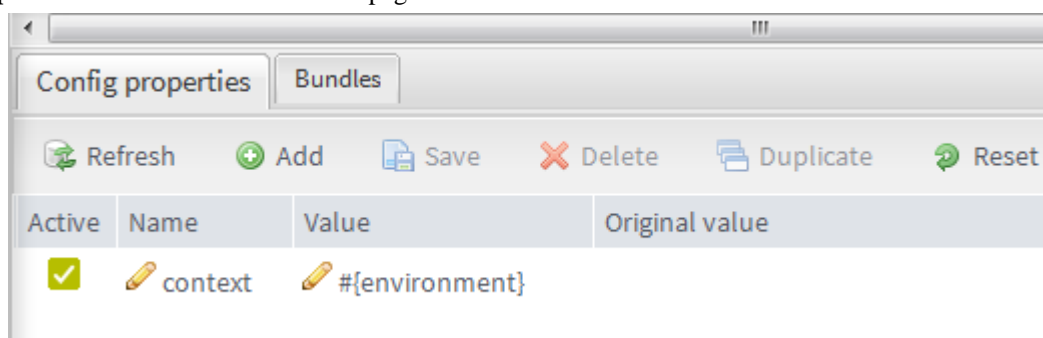
Context:  ▼

Server:  ▼

Property ID(PID):

Save
Cancel

10. Once the SayHelloConsumer deployment task listed in the **ESB Conductor**, select it and go to the **Config properties** tab in the lower half of the page:



11. Click the **Add** button to create a new property with **Name** *context* and **Value** *#{environment}*, where:
- context* corresponds to the declaration of the context parameter expected as you added execution context to the SayHelloConsumer Job.
  - #{environment}* corresponds to the declaration of the *environment* placeholder declared via the *#{}* syntax.

Click **Save**.

This way when you will deploy the consumer to the container, the configuration file that needs to contain the context parameter of the SayHelloConsumer Job will be automatically created: `esb/container/etc/SayHelloConsumer.cfg`

- Click the **Deploy** button on the top and the Service will be deployed into the specified Talend Runtime container.

### 10.6.1.2. Consuming the SayHelloService according to the environment via Provisioning

If you used the **tesb:start-all** command to start all the Talend Infrastructure services, the Provisioning Service is already started.

If not, go back to the container and start it as follows:

- `tesb:start-provision-server`

- `tesb:start-provision-agent`

Consume the SayHelloService in a development environment.

- Create a profile with the feature, resource and placeholder needed for the development environment.

```
tprovision:profile-create dev_env dev
```

- Add the feature you previously published in the Artifact Repository:

```
tprovision:feature-create dev_env mvn:org.example/SayHelloConsumer-feature/0.1.0-SNAPSHOT/xml SayHelloConsumer-feature 0.1.0-SNAPSHOT
```

- Add the resource that will contain the context used by the SayHelloConsumer:

```
tprovision:resource-create dev_env etc/SayHelloConsumer.cfg
```

- Add the placeholder providing the *dev* context to apply to the SayHelloConsumer:

```
tprovision:placeholder-create dev environment dev
```

- Release the placeholder:

```
tprovision:placeholders-release dev 1.0
```

- Release the profile:

```
tprovision:profile-release dev_env 1.0 dev 1.0
```

- Now that the new *dev\_env* profile has been released, you can apply it to the container on which you deployed the SayHelloService, it will automatically deploy the SayHelloConsumer feature in the Talend Runtime container with the corresponding context (the *environment* placeholder will be dynamically replaced by its value: *dev*, which corresponds to the *karaf* credentials defined in the Job context variables) allowing it to consume the service.

To apply the *dev\_env* profile, use the following command:

```
tprovision-agent:apply-profiles dev_env 1.0
```

Consume the SayHelloService in a production environment.

1. Create a profile with the feature, resource and placeholder needed for the production environment:

```
tprovision:profile-create prod_env prod
```

2. Add the feature you previously published in the Artifact Repository:

```
tprovision:feature-create prod_env mvn:org.example/SayHelloConsumer-feature/0.1.0/xml SayHelloConsumer-feature 0.1.0
```

3. Add the resource that will contain the context used by the SayHelloConsumer:

```
tprovision:resource-create prod_env etc/SayHelloConsumer.cfg
```

4. Add the placeholder providing the *prod* context to apply to the SayHelloConsumer:

```
tprovision:placeholder-create prod environment prod
```

5. Release the placeholder:

```
tprovision:placeholders-release prod 1.0
```

6. Release the profile:

```
tprovision:profile-release prod_env 1.0 prod 1.0
```

7. Now that the new **prod\_env** profile has been released, you can apply it to the container on which you deployed the SayHelloService, it will automatically deploy the SayHelloConsumer feature in the Talend Runtime container with the corresponding context (the *environment* placeholder will be dynamically replaced by its value: *prod*, which corresponds to the *tadmin* credentials defined in the Job context variables) allowing it to consume the service.

To apply the prod\_env profile, use the following command:

```
tprovision-agent:apply-profiles prod_env 1.0
```

The creation of profiles described above, performed directly in the Talend Runtime container can be performed as well from the **Provisioning** page of the Talend Administration Center. For more information, see the *Talend Administration Center User Guide*.

## 10.6.2. System profile use case

In this use case, you will see how to change the settings of your STS via a profile created with the Provisioning Service. As in this use case, you will only use resources and placeholders, you will be able to apply this profile to both Talend Runtime container and Apache Tomcat.

The resources in use in this use case are:

- org.talend.esb.job.client.sts.cfg
- clientKeystore.properties
- clientstore.jks

They will replace the default configuration files of your container when you will apply the profile to it.

The content of the org.talend.esb.job.client.sts.cfg is as follows:

```
#STS endpoint configuration
#sts.wsdl.location = http://localhost:8080/SecurityTokenService/UT?wsdl
```

```

sts.wsdl.location = #{STSEndpointUT}
sts.x509.wsdl.location = #{STSEndpointX509}
sts.namespace = http://docs.oasis-open.org/ws-sx/ws-trust/200512/
sts.service.name = SecurityTokenService
sts.endpoint.name = UT_Port
sts.x509.endpoint.name = X509_Port
sts.allow.renewing = false

#STS properties configuration
ws-security.sts.token.username = myclientkey
ws-security.sts.token.usecert = true
ws-security.is-bsp-compliant = false
ws-security.sts.token.properties = #{clientKSproperties}
security.encryption.username = mystskey
security.encryption.properties = #{clientKSproperties}

```

The content of the `clientKeystore.properties` is as follows:

```

org.apache.ws.security.crypto.merlin.keystore.type = jks
org.apache.ws.security.crypto.merlin.keystore.password = #{KSpwd}
org.apache.ws.security.crypto.merlin.keystore.alias = #{KSalias}
org.apache.ws.security.crypto.merlin.keystore.file = #{KSfile}

```

The values defined with the `#{ }` syntax correspond to the placeholders you will create in the profile, and they will be replaced dynamically by the values you will define in the profile.

1. Create a profile with the resources and placeholders needed for your STS settings:

```
tprovision:profile-create InfraProfile infra
```

2. Add the resource:

```
tprovision:resource-create InfraProfile c:/Temp/org.talend.esb.job.client.sts.cfg
```

3. Add the resource:

```
tprovision:resource-create InfraProfile c:/Temp/clientKeystore.properties etc/
keystores/clientKeystore.properties
```

4. Add the resource:

```
tprovision:resource-create InfraProfile c:/Temp/clientstore.jks etc/keystores/
clientstore.jks
```

5. Add the placeholder:

```
tprovision:placeholder-create infra STSEndpointUT http://localhost:8040/services/
SecurityTokenService/UT?wsdl
```

6. Add the placeholder:

```
tprovision:placeholder-create infra STSEndpointX509 http://localhost:8040/services/
SecurityTokenService/X509?wsdl
```

7. Add the placeholder:

```
tprovision:placeholder-create infra clientKSproperties file:\${tesb.home}/etc/
keystores/clientKeystore.properties
```

8. Add the placeholder:

```
tprovision:placeholder-create infra KSpwd cspass
```

9. Add the placeholder:

```
tprovision:placeholder-create infra KSalias myclientkey
```

10. Add the placeholder:

```
tprovision:placeholder-create infra KSfile ./etc/keystores/clientstore.jks
```

11. Release the placeholder:

```
tprovision:placeholders-release infra 1.0
```

12. Release the profile:

```
tprovision:profile-release InfraProfile 1.0 infra 1.0
```

13. Now that the new **InfraProfile** profile has been released, you can apply it to the container to apply the STS settings defined in the profile, it will automatically replace the configuration files by the resources you added to the profile and the placeholders defined in those resources will be replaced by their new value defined as well in the profile.

To apply this configuration profile to the container:

- Type in the following command in its console:

```
tprovision-agent:apply-profiles InfraProfile 1.0
```

The creation of the profile described above, performed directly in the Talend Runtime container can be performed as well from the **Provisioning** page of the Talend Administration Center. For more information, see the *Talend Administration Center User Guide*.

To apply this configuration profile in Tomcat:

- Go to the Provisioning agent deployed in Apache Tomcat: <http://localhost:8080/provisioning-web-agent-6.4.1/web-agent>
- In the Provisioning Web agent user interface:

**Installed Profiles:**

---

**Lookup Profile:**

Profile name:  Version:

In the **Lookup Profile** area, in the **Profile name** field, type in *InfraProfile* to look up for that profile, and in the **Version** field, type in the *1.0* version you just released. Click **Lookup**.

3. The **InfraProfile** you looked up is listed:

**Installed Profiles:**

---

**Lookup Profile:**

Profile name:  Version:

Profile name	Version	Autoupdate	Pool interval
InfraProfile	1.0	false	

Click **Apply** to apply it.

4. **InfraProfile** version **1.0** is applied in Tomcat.

Installed Profiles:

- Profile name: **InfraProfile** Version: **1.0** Auto-update: **false** Pool interval:

Lookup Profile:

Profile name:  Version:

Profile successfully applied.







## Chapter 11. ActiveMQ

Java Message Service (JMS) is a standardized Java API for sending messages between two or more applications. ActiveMQ implements the JMS 1.1 specification along with other messaging protocols.

There are two types of communication supported by JMS 1.1:

- point-to-point: direct messages are sent from a producer to a specified consumer via a JMS queue.
- publish and subscribe: communication is indirect through topics. Topics are published by producers, and consumers subscribe to specified topics.

Talend ESB embeds Apache ActiveMQ message broker to support this functionality. The job of the ActiveMQ message broker is to transport events between distributed applications, guaranteeing that they reach their intended recipients.

Beyond this documentation, see <http://activemq.apache.org> for more information.

## 11.1. Overview

- The Apache ActiveMQ broker can be run as a standalone server (see [Standalone ActiveMQ broker](#)), or inside a container (see [ActiveMQ broker inside a Talend Runtime container](#)).
- The ActiveMQ Web Console is a web based administration tool for an ActiveMQ broker (see [ActiveMQ Web Console](#)).
- ActiveMQ OSGi bundles ([ActiveMQ OSGi bundles](#)) may also be used in a Talend Runtime container to communicate with an ActiveMQ broker.
- You can also access ActiveMQ programmatically - see the section on the (Apache Camel) ActiveMQ component in the *Talend ESB Mediation Developer Guide*.

### 11.1.1. Download and install

ActiveMQ ships with Talend ESB; the relevant files are in the `<TalendRuntimePath>/activemq` directory, and include binary distributions for all supported platforms.

## 11.2. Standalone ActiveMQ broker

You can run the Apache ActiveMQ broker server as follows:

1. In a command console:

```
cd <TalendRuntimePath>/activemq/bin
```

2. Then enter:

- `activemq console` (Linux\*)
- `activemq start` (Windows)

The Apache ActiveMQ broker should now be running.

\*Note the Linux console option runs the broker in the foreground; the default is to run it in the background.

You can view this using the local Web Console at `http://localhost:8161/admin/`. To increase reliability, you may wish to run the Web Console in a separate container as preconfigured in Talend ESB, see [ActiveMQ Web Console](#).

### 11.2.1. Configuration

There are a number of configuration options, and these are listed by entering `activemq -h`.

You can configure the ActiveMQ broker by using either a configuration file or via broker URI. The default location for configuration files is in `activemq/conf`.

The syntax is `Main start [start-options] [uri]`

**Table 11.1. [start-options] syntax**

Option	Description	Example
-D<name>=<value>	Define a system property	activemq -Dactivemq.home=<TalendRuntimePath>/activemq (default if using Talend ESB)
--version	Display the version information	activemq --version
-h,-?,-help	Display the start broker help information	activemq -h

Note in the table below, the transport URI specifies the transport and ports to connect to the broker, for example TCP to connect to a remote ActiveMQ using a TCP socket, or VM which allows clients to connect to a broker in a container within the same VM. (Having multiple connectors may improve reliability and load balancing.) For the full list of options, see <http://activemq.apache.org>.

**Table 11.2. [uri] parameter syntax**

Example	Type	Description
xbean:file:activemq.xml	XBean based	Loads the xbean configuration file from the current working directory
activemq xbean:activemq.xml	XBean based	Loads the xbean configuration file from the classpath
activemq broker:(tcp://localhost:61616, tcp://localhost:5000)?useJmx=true	URI based	Configures the broker with 2 transport connectors and jmx enabled.
activemq broker:(tcp://localhost:61616, network:tcp://localhost:5000)?persistent=false	URI based	Configures the broker with 1 transport connector, and 1 network connector and persistence disabled

Note, the broker URI information can also be added to the configuration file instead of being specified on the command line.

## 11.3. ActiveMQ OSGi bundles

All ActiveMQ modules are packaged as OSGi bundles and can be used in any OSGi container, such as the Talend Runtime container.

1. By default, the ActiveMQ Karaf features are already added to the Talend Runtime container, but if not present, they can be added using:

```
feature:repo-add mvn:org.apache.activemq/activemq-karaf//xml/features
```

2. Enter the following command to display the ActiveMQ bundles in the container console:

```
feature:list | grep activemq
```

```
karaf@trun> feature:list | grep activemq
```

```
[uninstalled] [] activemq-broker-noweb activemq- Full ActiveMQ broker with
default configuration
[uninstalled] [] activemq-broker activemq- Full ActiveMQ broker with
default configuration and web console
[uninstalled] [] activemq-camel activemq-
[uninstalled] [] activemq-web-console activemq-
[uninstalled] [] activemq-blueprint activemq-
[uninstalled] [] activemq-client activemq-core- ActiveMQ client libraries
[uninstalled] [] activemq activemq-core- ActiveMQ broker libraries
```

3. To install and start the ActiveMQ features, execute the following command:

```
feature:install activemq
```

Once ActiveMQ installed and started in the container, you will be able to use it via the following commands:  
`activemq:[command]`

[command]	Description
browse	Display selected messages in a specified destination
bstat	Displays useful broker statistics
dstat	Displays a tabular summary of statistics for the queues on the broker
list	Lists all available brokers in the specified JMX context
purge	Delete selected destination's messages that matches the message selector
query	Display selected broker component's attributes and statistics

And to obtain detailed help on a given command, just execute: `activemq:[command] --help`

## 11.4. ActiveMQ broker inside a Talend Runtime container

An ActiveMQ broker may also be run in an OSGi container such as Talend Runtime container.

### 11.4.1. Broker creation

By default, no broker is created in the Talend Runtime container. The following commands can be used to start a broker within the Talend Runtime container: `karaf@trun> feature:install activemq-broker`

It creates a broker with a sensible default configuration, but you can edit the mentioned file to modify the broker's configuration.

### 11.4.2. Broker querying

Several commands are available to query the broker. To address local brokers, you'll need to use the `--jmxlocal` parameter. The following command displays available brokers:

```
karaf@trun> activemq:list --jmxlocal
BrokerName = mybroker
```

For more detailed information, run:

```
karaf@trun> activemq:query --jmxlocal
```

It will display informations about the connectors, list of queues, and so on. You can also browse or purge queues using the `activemq:browse` and `activemq:purge` commands.

## 11.5. ActiveMQ Web Console

The ActiveMQ Web Console is a web based administration tool for working with ActiveMQ, which can be configured to communicate with a standalone ActiveMQ broker or one running in a container. Web Console is included in the ActiveMQ distribution.

## 11.5.1. Configuring ActiveMQ Web Console

When an ActiveMQ broker is running, an ActiveMQ Web Console is automatically created in the same VM or container. Similarly, starting an ActiveMQ Web Console with no configuration specified will create a broker embedded in the same VM or container. However, to increase reliability, you may wish to run the Web Console in a separate container from the broker.

In the Talend Runtime, the ActiveMQ Web Console is pre-configured to connect to a broker running within another Talend Runtime via tcp. So by default, it does not create its own embedded broker.

The pre-configured properties are enabled when installing the Talend Runtime and are in the `<container>/etc/system.properties` file:

```
webconsole.type=properties
webconsole.jms.url=tcp://localhost:61616
webconsole.jmx.url=service:jmx:rmi:///jndi/rmi://localhost:1099/karaf-trun
webconsole.jmx.user=tesb
webconsole.jmx.password=tesb
```

Where:

- `webconsole.jms.url` is the URL of the broker
- `webconsole.jmx.url` is the JMX URL of the Talend Runtime.



If any configuration changes are made, the container will need to be restarted for them to take effect.

## 11.5.2. Install the Web Console to a container

In order to install the Web Console to a container, enter:

```
karaf@trun> feature:install activemq-web-console
```

This will install and start the Web Console normally accessible at `http://localhost:8040/activemqweb` but dependent on your configuration. See [Configuring ActiveMQ Web Console](#) for the configuration details.

- To connect to a standalone broker from a Web Console in a container, the configuration details in `<container>/etc/system.properties` will need to be updated. The default, local Web Console for a standalone broker is at `http://localhost:8161/admin/`.
- If the default Talend configuration is commented out or deleted, a broker will start in the local container and connect to it (an error will be shown if a broker is already running).

## 11.5.3. Additional configuration for authentication

In order to use the ActiveMQ Web Console with a broker configured with authentication, it is necessary to edit the `container/etc/system.properties` file and configure the username and password for a JMS connection:

```
webconsole.jms.user=system
webconsole.jms.password=manager
```

## 11.6. Examples

The examples are located in `<TalendRuntimePath>/activemq/example`. For more information on each example, please see the corresponding *readme* file.





## Chapter 12. Artifact Repository

Most Java Developers already use Nexus Open Source or Nexus Pro, therefore it will be the default Artifact Repository to use with the Talend ESB. If you are using Nexus Open Source or Nexus Pro, just follow the standard Nexus Java developer documentation for more information on how to upload artifacts to the Repository and how to manage them with the Nexus Web User Interface. However, the use of Apache Archiva as Artifact Repository is also possible.

Talend Administration Center is a web-based application for administering all aspects of associated software, from collaborative work and the related code repository management, up to the remote deployment of production data services and routes. Talend Administration Center uses Artifact Repository to store and to provide the deployment of artifacts for the Talend Runtime container, and their user interfaces are linked for ease of use.

Talend Administration Center is available **only for Talend subscription products**. For information on the Talend Administration Center, see the *Talend Installation Guide* and the *Talend Administration Center User Guide*.

To change the default Artifact Repository to use from Nexus to Archiva, change the default repository parameters in the `org.ops4j.pax.url.mvn.cfg` configuration file of the Talend Runtime container to point to Archiva.

For more information on the Artifact Repository, see the *Talend Installation Guide*.

This chapter discusses both Nexus and Archiva configuration in Talend ESB for both the Talend Runtime provided with Talend Open Studio for ESB, and Talend ESB.



Please note that Maven 3.0.3+ is required for the functionality described in this document.

## 12.1. Nexus Artifact Repository

Nexus Artifact Repository is based on Sonatype Nexus, which acts as a sort of shared server of Maven artifacts repositories. It is possible to deploy artifacts in the Talend Runtime provided with Talend Open Studio for ESB via a Maven Repository Manager. Nexus is the recommended Artifact Repository to be used with Talend ESB, and it is the default one to be used with the subscription version of Talend ESB to support the deployment of artifacts to the distributed Talend Runtime container, using a number of pre-configured Talend repositories, which are in addition to the default Nexus ones. This default version of Nexus: *Artifact-Repository-Nexus-VA.B.C.D.E*, is provided as a zip with the Talend Administration Center download.

For more information on Sonatype Nexus, see its documentation on <http://www.sonatype.org/nexus>.

This section focuses on using Nexus, and configuring Maven to access and deploy to Nexus repositories. However, note that the use of Apache Archiva is also possible. For more information about the use of Archiva, see [Archiva Artifact Repository \(deprecated\)](#).

### 12.1.1. Downloading and installing Nexus

**If using the Talend Runtime provided with Talend Open Studio for ESB:**

1. Download Nexus from <http://www.sonatype.org/nexus> and extract it.
2. In the Nexus directory, run:
  - `./bin/nexus console` (Linux)
  - `.\bin\nexus.bat console` (Windows)

Nexus will now be running on `http://localhost:8081/nexus/index.html`.

**If using Talend ESB** (subscription version of Talend ESB):

See the *Talend Installation Guide* for details on how to install the Nexus Artifact Repository.

The Artifact Repository will be running on: `http://localhost:8081/nexus/index.html`

With:

User	admin
Password	Talend123

### 12.1.2. Deploying in Nexus repository

1. Make sure Nexus is installed properly (according to the installation instructions related to the version you use: Nexus Open Source or Nexus Pro).

In case you installed it via the Talend Installer, you will find it as the **Talend Artifact** Service in your environment.

2. In your Maven `settings.xml` file, add the section:

```
<server>
```



```
<id>nexus</id>
<username>deployment</username>
<password>deployment123</password>
</server>
```

3. In the Maven project file (pom.xml), add your repository configuration in the <distributionManagement> section:

```
<distributionManagement>

  <!-- use the following if you're not using a snapshot version. -->
  <repository>
    <id>nexus</id>
    <name>RepositoryProxy</name>
    <url>
      http://localhost:8081/nexus/content/repositories/releases
    </url>
  </repository>

  <!-- use the following if you ARE using a snapshot version. -->
  <snapshotRepository>
    <id>nexus</id>
    <name>RepositoryProxy</name>
    <url>
      http://localhost:8081/nexus/content/repositories/snapshots
    </url>
  </snapshotRepository>
</distributionManagement>
```

http://localhost:8081/nexus/content/repositories/releases is the right URL to use when Nexus is installed as the default Artifact Repository via the Talend Installer or as standalone. However, if Nexus was installed as a Web Application in Tomcat (WAR File deployment), the URL is http://localhost:8080/nexus-webapp/content/repositories/releases.

4. In the file <TalendRuntimePath>/container/etc/org.ops4j.pax.url.mvn.cfg, check the URL of the Nexus repository in the parameter org.ops4j.pax.url.mvn.repositories and edit it if necessary.

```
org.ops4j.pax.url.mvn.repositories= \
  http://localhost:8081/nexus/content/repositories/releases@id=tesb.re
lease,\
  http://localhost:8081/nexus/content/repositories/snapshots@snapshots
@id=tesb.snapshot, \
```

http://localhost:8081/nexus/content/repositories/releases is the right URL to use when Nexus is installed as the default Artifact Repository via the Talend Installer or as standalone. However, if Nexus was installed as a Web Application in Tomcat (WAR File deployment), the URL is http://localhost:8080/nexus-webapp/content/repositories/releases.

## 12.2. Archiva Artifact Repository (deprecated)

Archiva Artifact Repository is based on Apache Archiva, and provides a standard-based repository. It can be used with Talend ESB to support the deployment of artifacts to the Talend Runtime container, using a number of configured repositories in addition to the default Archiva ones.

For more information on Apache Archiva, see <http://archiva.apache.org/>.

This section focuses on using Apache Archiva, and configuring Maven to access and deploy to Archiva repositories, but please keep in mind that the recommended artifact repository to use, and the default one to use with the suscription version of Talend ESB, is Nexus. For more insformation about the use of Nexus, see [Nexus Artifact Repository](#).

## 12.2.1. Downloading and installing Archiva



If using Linux, check the JDK version in the `conf\wrapper.conf` file in the Archiva installation to make sure the correct JDK is being referenced (see [Prerequisites to using Talend ESB products](#)); otherwise the default JDK on the local machine will be used. If needed, update this line by inserting the correct JDK path:

```
# Java Application
wrapper.java.command=/pathToCorrectJDK/java
```

1. Download Apache Archiva from <http://archiva.apache.org>, and extract it.
2. In the Archiva directory, run:
  - `./bin/archiva console` (Linux)
  - `.\bin\archiva.bat console` (Windows))

Archiva will now be running on `http://localhost:8080/archiva/`, with:

User	tadmin
Password	tadmin

## 12.2.2. Browsing repositories

### 12.2.2.1. Permissions

The user can browse only those repositories where the user is an observer or a manager. If the user does not have permission to access any repository, a message indicating that the user will need to be granted access from the system administrator access" will be displayed.

## 12.2.2.2. Artifact Info

Items in the repositories are hyperlinked allowing easy access to viewing more information. By clicking on the Group Id or Artifact Id the repository browser will be shown. The Artifact Info page is divided into six views:

1. **Info:** Basic information about the artifact is displayed here. These are the `groupId`, `artifactId`, `version` and `packaging`. A dependency pom snippet is also available, which a user can simply copy and paste in a POM file to declare the artifact as a dependency of the project.
2. **Dependencies:** The dependencies of the artifact will be listed here. The user can navigate easily to a specific dependency by clicking on the `groupId`, `artifactId`, or `version` link. The scope of the dependency is also shown.
3. **Dependency Tree:** The dependencies of the artifact are displayed in a tree-like view, which can also be navigated.
4. **Used By:** Lists all the artifacts in the repository which use this artifact.
5. **Mailing Lists:** The project mailing lists available in the artifact's pom are displayed here.
6. **Download:** Clicking on this link will download the artifact to your local machine.

## 12.2.2.3. Downloading Artifacts

Artifacts can be downloaded from the artifact info page. All files, except for the `metadata.xml` files, that are associated with the artifact are available in the download box.

The size of the files in bytes are displayed at the right section of the download box. Note: Upon downloading the artifact, you will be asked to enter your username and password for the repository where the artifact will be downloaded from. Only users with Global Repository Manager, Repository Manager, or Repository Observer roles for that repository can download the artifact.

## 12.2.2.4. Identifying an Artifact

Archiva indexes all of the artifacts that it discovers during the repository scanning process, storing information about their contents. This includes the checksum of the artifact, which can help to uniquely identify it within the repository.

You can search for an artifact using this checksum, please see <http://archiva.apache.org> for more details.

## 12.2.3. Configuring Maven to use an Archiva repository

To get your local Maven installation to use an Archiva proxy, you will need to add the repositories you require to your `settings.xml`. This file is usually found in `$user.dir/.m2/settings.xml` (see <http://maven.apache.org/settings.html> for more details).

How you configure the settings depends on how you would like to utilise the repository. You can add the Archiva repository as an additional repository to others already declared by the project.

### 12.2.3.1. Using Archiva as an additional repository

You will need to add one entry for each repository that is setup in Archiva. If your repository contains plugins, remember to also include a `<pluginRepository>` setting.

1. Create a new profile to setup your repositories:

```
<settings>
...
  <profiles>
    <profile>
      <id>Repository Proxy</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <!-- ***** -->
      <!-- repositories for jar artifacts -->
      <!-- ***** -->
      <repositories>
        <repository>
          ...
        </repository>
        ...
      </repositories>
      <!-- ***** -->
      <!-- repositories for maven plugins -->
      <!-- ***** -->
      <pluginRepositories>
        <pluginRepository>
          ...
        </pluginRepository>
        ...
      </pluginRepositories>
    </profile>
    ...
  </profiles>
  ...
</settings>
```

2. Add your repository configuration to the profile.

You can copy the repository configuration from the POM Snippet on the Archiva Administration Page for a normal repository. It should look much like:

```
<repository>
  <id>repository-1</id>
  <url>
    http://repo.mycompany.com:8080/archiva/repository/internal/
  </url>
  <releases>
    <enabled>true</enabled>
  </releases>
  <snapshots>
    <enabled>false</enabled>
  </snapshots>
</repository>
```

3. Add the necessary security configuration

This is only necessary if the guest account does not have read access to the given repository.

```
<settings>
...
  <servers>
    <server>
```

```

        <id>repository-1</id>
        <username>{archiva-user}</username>
        <password>{archiva-pwd}</password>
      </server>
      ...
    </servers>
    ...
  </settings>

```

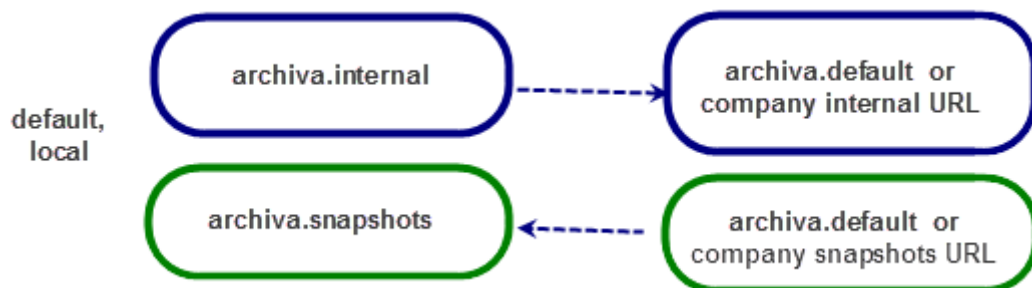
## 12.2.4. Deploying to a Repository

Now that we have configured Maven to use Archiva, we are ready to deploy to it. There are different ways on how you can deploy artifacts in an Archiva repository.

- Configuring Maven to deploy to an Archiva repository which is covered in this section.
- Deploying via the Web UI Form - please see <http://archiva.apache.org> for more details.

### 12.2.4.1. Configuring Maven to deploy to an Artifact Repository

**Figure 12.1. Default Archiva Repositories**



1. Create a user in Archiva to use for deployment (or use 'guest' if you wish to deploy without a username and password - however, 'guest' is not available with Talend repositories).
2. The deployment user needs the Role 'Repository Manager' for each repository it is desired to deploy to.
3. Define the server for deployment inside your 'settings.xml', use the newly created user for authentication:

```

<settings>
  ...
  <servers>
    <server>
      <id>archiva.internal</id>
      <username>{archiva-deployment-user}</username>
      <password>{archiva-deployment-pwd}</password>
    </server>
    <server>
      <id>archiva.snapshots</id>
      <username>{archiva-deployment-user}</username>
      <password>{archiva-deployment-pwd}</password>
    </server>
    ...
  </servers>
  ...
</settings>

```

## 12.2.4.2. Deploying to Archiva using HTTP

Configure the `distributionManagement` part of your `pom.xml` (customizing the URLs as needed).



The ID of the repository in `distributionManagement` must match the ID of the server element in `settings.xml`.

```
<project>
...
<distributionManagement>
  <repository>
    <id>archiva.internal</id>
    <name>Internal Release Repository</name>
    <url>
      http://reposerver.mycompany.com:8080/archiva/repository/internal/
    </url>
  </repository>
  <snapshotRepository>
    <id>archiva.snapshots</id>
    <name>Internal Snapshot Repository</name>
    <url>
      http://reposerver.mycompany.com:8080/archiva/repository/snapshots/
    </url>
  </snapshotRepository>
</distributionManagement>
...
</project>
```

## 12.2.4.3. Deploying to Archiva using WebDAV

In some cases you may wish to use WebDAV (Web-based Distributed Authoring and Versioning) instead of HTTP for deployment, perhaps for greater ease of collaboration. For WebDAV follow the same process as for HTTP, with this additional step:

Add `dav:` to the front of the deployment URLs:

```
<project>
...
<distributionManagement>
  <repository>
    <id>archiva.internal</id>
    <name>Internal Release Repository</name>
    <url>dav:http://reposerver.mycompany.com:8080/archiva/repository/i
internal/</url>
  </repository>
  <snapshotRepository>
    <id>archiva.snapshots</id>
    <name>Internal Snapshot Repository</name>
    <url>dav:http://reposerver.mycompany.com:8080/archiva/repository/s
napshots/</url>
  </snapshotRepository>
</distributionManagement>
...
</project>
```



## Chapter 13. Auxiliary Storage

Auxiliary storage is created as a lightweight persistent storage for Request\Callback context or other small objects.

The Auxiliary storage service is an OSGi service which is a part of Talend ESB distribution, for both Community and subscription versions.

After starting the Talend Runtime container, to start the Auxiliary Storage service, enter the following commands at the console prompt:

**tesb:start-aux-store** to start the Auxiliary Storage Service,

**tesb:stop-aux-store** to stop the Auxiliary Storage Service.

## 13.1. Implementation details and configuration

The service can use two different tools as a persistence layer. One is Apache Avro -based file storage. This storage is a default. Another, a more scalable persistence layer based on Apache Jackrabbit, supports Derby, MySQL, Oracle, Postgres and MS SQL Server as a backend. The configuration file of the auxiliary storage service is *etc/org.talend.esb.auxiliary.server.cfg*. This file contains settings for setting up persistence layer as well as path to the persistent repository. Here is an example of this file:

### **org.talend.esb.auxiliary.server.cfg configuration file**

```
# Repository type FILEStore,JCRStore
callback.store = FILEStore

# Repository home directory
callback.store.home = ${tesb.home}/esbrepo/callbackrepo
```

The security settings of the RESTful service, that provides access to the Auxiliary storage, are configured in a separate configuration file: *etc/org.talend.esb.auxiliary.storage.service.cfg*. By default, the authentication is deactivated.

### **org.talend.esb.auxiliary.storage.service.cfg configuration file**

```
# Authentication method BASIC,SAML,NO
auxiliary.storage.service.authentication = NO

security.signature.properties = file:${tesb.home}/etc/keystores/
serviceKeystore.properties
security.signature.username = myservicekey
security.signature.password = skpass
```

The Jackrabbit storage configuration file is *etc/org.talend.esb.auxiliary.repo.xml*.

For more information about extending Jackrabbit for different backends, see [Backend configuration](#).





## Appendix A. Backend configuration

Several Talend ESB features use the Apache Jackrabbit repository as Java Content Repository (JCR), for example: the Service Registry, the XACML, the Auxiliary Storage, and the Provisioning Service, as a backend application. For the Service Registry, it is one of the main components of its Server part used to store registry data. For the XACML, it is used as XACML Registry to deploy, retrieve, and delete XACML policies. For the Auxiliary Storage, the repository is used to store Request/Callback context.

Jackrabbit supports two types of storage: file and database. By default, the file type is used, but you can change it to the database type. When the database type of storage is used, by default, an embedded Apache Derby DB is configured. This appendix describes how to configure Apache Jackrabbit with another database, for example: Postgres and Oracle.

## A.1. Configuring database-based storage in Apache Jackrabbit

The content repository used for the backend of the Service Registry, XACML, Auxiliary Storage, and Provisioning Service features can be configured to use a database as backend separately for versioning and the repository itself. By default, an embedded Apache Derby DB is configured.

The configuration for the Jackrabbit backend is located in:

- `container/system/org/talend/esb/registry/registry-server/<version>/registry-server-<version>-org.talend.esb.registry.repo.xml` for the Service Registry
- `container/system/org/talend/esb/authorization/xacml/registry/server/<version>server-<version>-org.talend.esb.authorization.repo.xml` for XACML
- `container/system/org/talend/esb/auxiliary/storage/auxiliary-storage-persistence-jcr/<version>/auxiliary-storage-persistence-jcr-<version>-org.talend.esb.auxiliary.repo.xml` for Auxiliary Storage
- `container/system/org/talend/esb/provisioning/provisioning-server/<version>/provisioning-server-<version>-org.talend.esb.provisioning.repo.xml` for Provisioning Service

A default configuration is created during the first use of each of the four features, if there is no configuration in place. By default, it uses Apache Derby and the file system as persistent store. For more information about the Jackrabbit PersistenceManager, see <http://wiki.apache.org/jackrabbit/PersistenceManagerFAQ>.

## A.2. Changing the default Apache Derby DB to Postgres

Below is explain how to change the default Apache Derby DB to use Postgres. Other databases can be configured the same way.

1. Make sure the Postgres JDBC Driver (or any other driver that corresponds to the database you want to use) has been deployed to the Talend Runtime container before starting the configuration of Postgres. If the jdbc driver is installed later, the Service Registry, XACML Authorization Repository, Auxiliary Storage, or Provisioning Service server bundle would need to be refreshed.

The Postgres Driver can be downloaded at <http://jdbc.postgresql.org/download.html>, and the BND tool, needed to bundle the driver jar, can be found at <https://bndtools.ci.cloudbees.com/job/bnd.master/lastBuild/artifact/dist/bundles/biz.aQute.bnd/>.

2. Apply the BND to create a driver bundle:

```
java -jar bnd.jar wrap postgresql-9.2-1003.jdbc4.jar
```

```
mv postgresql-9.2-1003.jdbc4.bar postgresql-9.2-1003.jdbc4.jar
```

3. And in the container console, type in the following command to deploy the JDBC driver bundle into the container:

```
install file:///PATH_TO_DRIVER/postgresql-9.2-1003.jdbc4.jar
```

4. Use Postgres pgAdmin III to create a new database named:

- *jcrRegistry* for the Service Registry

- *jcrXacml* for the XACML Authorization Repository
  - *jcrAuxiliary* for Auxiliary Storage
  - *jcrProvisioning* for Provisioning Service
5. Assign the appropriate access rights to a role corresponding to the user/password used for the configuration.
  6. If you choose a different name for the database, replace all occurrences of *jcrRegistry*, *jcrXacml*, *jcrAuxiliary*, or *jcrProvisioning* in the following configuration examples accordingly.
  7. Please choose different database names for each feature (Service Registry, XACML Authorization Repository, Auxiliary Storage, and Provisioning Service) in case they use Postgres, to avoid clashes.
  8. In the configuration file, substitute the `PersistenceManager` for both the workspace and the versioning as shown in the example configurations below.

- In the **Workspace** configuration, replace the default `DerbyPersistenceManager` by `PostgreSQLPersistenceManager`:

```
<Workspace name="$ {wsp.name}">
  <!-- persistence manager of the workspace: -->
  <PersistenceManager
    class="org.apache.jackrabbit.core.persistence.bundle.PostgreSQLPersistenceManager">
    <param name="driver" value="org.postgresql.Driver"/>
    <param name="url" value="jdbc:postgresql://localhost:5432/jcrRegistry"/>
    <param name="schema" value="postgresql"/>
    <param name="user" value="postgres"/>
    <param name="password" value="secret"/>
    <param name="schemaObjectPrefix" value="jcr_${wsp.name} _"/>
    <param name="externalBLOBs" value="false"/>
  </PersistenceManager>
```

- In the **Versioning** configuration, replace the default `DerbyPersistenceManager` by `PostgreSQLPersistenceManager`:

```
<Versioning rootPath="$ {rep.home}/version">
  <!--
  Configures the persistence manager to be used for persisting version state.
  Please note that the current versioning implementation is based on
  a 'normal' persistence manager, but this could change in future
  implementations.
  -->
  <PersistenceManager
    class="org.apache.jackrabbit.core.persistence.bundle.PostgreSQLPersistenceManager">
    <param name="driver" value="org.postgresql.Driver"/>
    <param name="url" value="jdbc:postgresql://localhost:5432/jcrRegistry"/>
    <param name="schema" value="postgresql"/>
    <param name="user" value="postgres"/>
    <param name="password" value="secret"/>
    <param name="schemaObjectPrefix" value="version_"/>
    <param name="externalBLOBs" value="false"/>
  </PersistenceManager>
```

9. If you also want to store large binary objects inside the database, adapt the `DataStore` definition as follows. By default, a file system based solution is used. For potential drawbacks of this decision, see <http://wiki.apache.org/jackrabbit/DataStore>.

In the **DataStore** configuration, replace the default `FileDataStore` by `DbDataStore`:

```
<!-- data store configuration -->
<DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
```

```

<param name="url" value="jdbc:postgresql://localhost:5432/jcrRegistry"/>
<param name="user" value="postgres"/>
<param name="password" value="secret"/>
<param name="databaseType" value="postgresql"/>
<param name="driver" value="org.postgresql.Driver"/>
<param name="minRecordLength" value="1024"/>
<param name="maxConnections" value="3"/>
<param name="copyWhenReading" value="true"/>
<param name="tablePrefix" value="" />
<param name="schemaObjectPrefix" value="" />
</DataStore>

```

## A.3. Configuring the Apache Jackrabbit storage to use Oracle

This section explains how to configure the Apache Jackrabbit storage to use Oracle.

Make sure your Oracle database driver has been deployed to the Talend Runtime container before starting the configuration as shown in the previous section.

The following procedure shows how to change the configuration file for the Service Registry. For the XACML, the Auxiliary Storage, and the Provisioning Service, change the value of the `schemaObjectPrefix`. For example, for the XACML, in the **FileSystem** configuration for the repository, set the `schemaObjectPrefix` to "AX\_FS\_REPO".

1. Substitute the `FileSystem` and `DataStore` for the repository as shown in the example configurations below.

Replace the default `LocalFileSystem` and `FileDataStore` by `OracleFileSystem` and `DbDataStore` respectively:

```

<Repository>
  <!-- virtual file system where the repository stores global state
        (e.g. registered namespaces, custom node types, etc.) -->
  <FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
    <param name="url" value="jdbc:oracle:thin:@HOST:1521:xe" />
    <param name="user" value="USER" />
    <param name="password" value="secret" />
    <param name="schema" value="oracle" />
    <param name="schemaObjectPrefix" value="SR_FS_REPO_" />
  </FileSystem>
  <!-- data store configuration -->
  <DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
    <param name="url" value="jdbc:oracle:thin:@HOST:1521:xe" />
    <param name="driver" value="oracle.jdbc.OracleDriver" />
    <param name="user" value="USER" />
    <param name="password" value="secret" />
    <param name="databaseType" value="oracle" />
    <param name="minRecordLength" value="1024" />
    <param name="maxConnections" value="3" />
    <param name="copyWhenReading" value="true" />
    <param name="tablePrefix" value="" />
    <param name="schemaObjectPrefix" value="SR_DS_REPO_" />
  </DataStore>

```

2. Substitute the `FileSystem` and `PersistenceManager` for the workspace as shown in the example configurations below.

Replace the default `LocalFileSystem` and `DerbyPersistenceManager` by `OracleFileSystem` and `OraclePersistenceManager` respectively:

```

<Workspace name="$ {wsp.name}">

```

```

<!-- virtual file system of the workspace:
      class: FQN of class implementing the FileSystem interface -->
<FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
  <param name="url" value="jdbc:oracle:thin:@HOST:1521:xe" />
  <param name="user" value="USER" />
  <param name="password" value="secret" />
  <param name="schema" value="oracle" />
  <param name="schemaObjectPrefix" value="SR_FS_WS_" />
</FileSystem>
<!-- persistence manager of the workspace:
      class: FQN of class implementing the PersistenceManager interface -->
<PersistenceManager
class="org.apache.jackrabbit.core.persistence.pool.OraclePersistenceManager">
  <param name="driver" value="oracle.jdbc.OracleDriver" />
  <param name="url" value="jdbc:oracle:thin:@HOST:1521:xe" />
  <param name="user" value="USER" />
  <param name="password" value="secret" />
  <param name="databaseType" value="oracle" />
  <param name="schemaObjectPrefix" value="SR_${wsp.name}_" />
  <param name="externalBLOBs" value="false" />
</PersistenceManager>

```

3. Substitute the `FileSystem` and `PersistenceManager` for the versioning as shown in the example configurations below.

Replace the default `LocalFileSystem` and `DerbyPersistenceManager` by `OracleFileSystem` and `OraclePersistenceManager` respectively:

```

<Versioning rootPath="$ {rep.home}/version">
  <!-- Configures the filesystem to use for versioning for the respective
        persistence manager -->
  <FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
    <param name="url" value="jdbc:oracle:thin:@HOST:1521:xe" />
    <param name="user" value="USER" />
    <param name="password" value="secret" />
    <param name="schema" value="oracle" />
    <param name="schemaObjectPrefix" value="SR_FS_VERSION_" />
  </FileSystem>
  <!--
    Configures the persistence manager to be used for persisting version state.
    Please note that the current versioning implementation is based on
    a 'normal' persistence manager, but this could change in future
    implementations.
  -->
  <PersistenceManager
class="org.apache.jackrabbit.core.persistence.pool.OraclePersistenceManager">
    <param name="driver" value="oracle.jdbc.OracleDriver" />
    <param name="url" value="jdbc:oracle:thin:@HOST:1521:xe" />
    <param name="user" value="USER" />
    <param name="password" value="secret" />
    <param name="databaseType" value="oracle" />
    <param name="schemaObjectPrefix" value="SR_VERSION_" />
    <param name="externalBLOBs" value="false" />
  </PersistenceManager>

```

