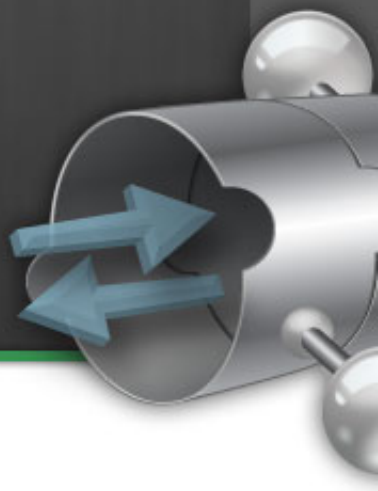
talend*
*open integration solutions



Talend ESB Infrastructure Services

Configuration Guide

5.1

Talend ESB Infrastructure Services: Configuration Guide

Publication date 3 May 2012

Copyleft

Notices

Table of Contents

List of Figures

List of Examples

Chapter 1. Introduction

This guide covers runtime installation and configuration information for Talend ESB services. Topics covered include installation and configuration of the Service Locator, Service Activity Monitoring and the Security Token Service.

1.1. Prerequisites to using Talend ESB

Compatible software versions

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

For a complete list of compatible software and software versions:

- If you are using Talend Enterprise ESB Studio or Talend Enterprise ESB, see **Talend Enterprise ESB Installation Guide**.
- If you are using Talend Open Studio for ESB or Talend ESB Standard Edition, see the link to the Installation Guide on the Talend ESB download page (`http://www.talend.com/download.php`).

1.2. Installation directory

We use the term `<Talend.runtime.dir>` for the directory where Talend Runtime is installed. This is typically the full path of either `Runtime_ESBSE` or `Talend-ESB-V5.1.x`, depending on the version of the software that is being used. Please substitute appropriately.

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

1.3. Other manuals

For more details on the Security Token Service, please also see the **Talend ESB STS User Guide**.

Chapter 2. Service Locator Installation

This chapter describes the steps to install and run the Service Locator. The Service Locator is a service that provides service consumers with a mechanism to discover service endpoints at run time. The Service Locator consists of two parts: The endpoint repository and the Service Locator feature.

Since creating a distributed, fault-tolerant endpoint repository is a non-trivial task, the Service Locator 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, visit http://zookeeper.apache.org/.

Only one instance of the Service Locator

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

2.1. Download

To get a server distribution, download a recent stable release from one of the Apache Download Mirrors. After this unpack it and cd to the root.

2.2. Standalone Operation

Setting up the Service Locator server in standalone mode is straightforward. The server is contained in a single JAR file, so installation consists of creating a configuration. Once you've downloaded a stable Service Locator server release, unpack it and navigate to its root directory. To start the Service Locator you need a configuration file. Here is a sample, create it in `conf/zoo.cfg`:

Example 2.1. Service Locator standalone configuration

```
tickTime=2000
dataDir=/var/locator
clientPort=2181
```

This file can be named as you wish, but for the sake of this discussion we call it `conf/zoo.cfg`. Change the value of dataDir to specify an existing (empty to start with) directory. Here is a description for each of the fields:

Field name	Description
`tickTime`	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 `tickTime`
`dataDir`	the location to store the in-memory database snapshots and, unless specified otherwise, the transaction log of updates to the database
`clientPort`	the port to listen for client connections

2.3. Start using command line

Now that you have created the configuration file, you can start Service Locator server. The bin directory contain scripts that allow easy access (classpath in particular) to the Service Locator server and command line client:

```
bin/zkServer.sh start
bin/zkServer.sh start configFilename (if configFilename != "zoo.cfg")
```

Files ending in `.sh` are Linux compatible. Files ending in `.cmd` are Windows compatible. 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. The steps outlined in this section 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.

2.4. 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:

Example 2.2. Replicated Service Locator configuration

```
tickTime=2000
dataDir=/var/locator
clientPort=2181
initLimit=5
syncLimit=2
server.1=locator_host1:2888:3888
server.2=locator_host2:2888:3888
server.3=locator_host3:2888:3888
```

The new configuration entry, `initLimit` limits the time the Service Locator servers in quorum have to connect to a leader. The configuration entry `syncLimit` limits how far out of date a server can be from a leader. For both of these timeouts the unit of time is specified using `tickTime`.

In this example, the timeout for `initLimit` is 5 ticks at 2000 milleseconds a tick, or 10 seconds. 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 has the contains the server number, in ASCII. Finally, 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.

2.5. Running Apache Zookeeper as an OSGI bundle

Another way to run Apache Zookeeper server is to install its OSGI bundle into the Karaf container. The configuration of Apache Zookeeper server is similar to the one used in standalone mode. The path to configuration file `\container\etc\org.apache.cxf.dosgi.discovery.zookeeper.server.cfg`

1. Start tesb container
2. Execute console command **`features:install tesb-zookeeper-server`**
3. Execute console command **list**

   You should see similar output

```
 ID State Blueprint Spring Level Name
      [ 168] [Active ] [ ] [ ] [ 60] ZooKeeper
      server control bundle (1.2)
```

To ensure that feature installed successfully, you can run examples related to Service Locator server.

2.6. 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's a good idea to back up the Service Locator data directory periodically. Although the Service Locator is highly reliable because a persistent copy is replicated on each server, recovering from backups may be necessary if a catastrophic failure or user error occurs.

The Service Locator server does not remove the snapshots and log files, so they will accumulate over time. You will need to cleanup this directory occasionally, based on your backup schedules and processes. To automate the cleanup, a `zkCleanup.sh` script is provided in the bin directory. Modify this script as necessary for your situation. In general, you 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.

2.7. Enabling Service Locator usage in CXF

You need the client component of the Service Locator (locator-<5.1.0>.jar) to enable your CXF service or consumer to use the Service Locator.

To use the Locator client in CXF you need to add the `locator-<5.1.0>.jar` into your classpath or war file. Also add it to the OSGi container if it uses one. To learn more about Locator client configuration for both provider or consumer please see the Service Locator Configuration Manual.

2.8. Service Locator SOAP Service

The Service Locator SOAP Service component provides a way to access the Service Locator operations (such as **Register an endpoint**, **Unregister an endpoint**, **Lookup endpoints for given service**, etc.) via the SOAP interface.

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

1. Type **`features:install tesb-locator-soap-service`** in the tesb container to enable the Service Locator service component;
2. Type **`features:install tesb-zookeeper-server`** in the tesb container to enable the Service Locator server (zookeeper server) component;
3. Type **list** in the tesb container. You should see the output:

```
ID State Blueprint Spring Level Name
[ 189] [Active ] [ ] [ ] [ 60] Locator
Service :: Common (5.1.0)
[ 190] [Active ] [ ] [ ] [ 60] Locator
Service :: SOAP Service (5.1.0)
[ 191] [Active ] [ ] [ ] [ 60] ZooKeeper
server control bundle (1.2)
```

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

Also you can configure the ZooKeeper server in the Talend Runtime container by editing the following configuration file:

`container/etc/org.apache.cxf.dosgi.discovery.zookeeper.server.cfg`

```
 # 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 acknowledgement
          syncLimit=5
          # the directory where the snapshot is stored.
          dataDir=./zookeeper/data
          # the port at which the clients will connect
```

```
clientPort=2181
#Number of client connection (default = 10; unlimited = 0)
maxClientCnxns = 0
```

This configuration is the same as the Service Locator configuration, described in Section 2.2, “Standalone Operation”.

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`

Currently the Service Locator service provides the following operations:

- **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-proxy/war/`:

An example of simple locator service configuration is in `/examples/talend/tesb/locator-service/soap-proxy/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-proxy/war/src/main/java/demo/service/ContextListener.java`:

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

try {
        client.registerEndpoint(new QName(
                "http://talend.org/esb/examples/", "GreeterService"),
                serviceHost, BindingType.SOAP_11, TransportType.HTTP, null);
    } catch (InterruptedExceptionFault 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="unregisterEnpointRequest">
 <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 **Unregister an endpoint** for a specific service provided in project `/examples/talend/tesb/locator-service/soap-proxy/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);
```

- **Lookup all endpoints for a given service:** 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 fullfil the required properties. If none exists return 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"/>
   <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 fullfills the required properties. If several endpoints match, select one randomly. If none exists, return business fault.

  The Lookup 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-proxy/client/`:

Example of simple locator service configuration you can see in `/examples/talend/tesb/locator-service/soap-proxy/client/src/main/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-proxy/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);
```

2.9. Service Locator REST Service

Service Locator REST Service component provides a way to access the Service Locator operations in REST manner.

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

1. Type **`features:install tesb-locator-rest-service`** in the tesb container to enable the REST Locator Service component;

2. Type **`features:install tesb-zookeeper-server`** in the tesb container to enable the Service Locator server (zookeeper server) component;

3. Type **list** in the tesb container. You should see the output:

```
ID State Blueprint Spring Level Name
[ 190] [Active ] [ ] [ ] [ 60] Locator
Service :: Common (5.1.0)
[ 191] [Active ] [ ] [ ] [ 60] Locator
Service :: REST Service (5.1.0)
[ 192] [Active ] [ ] [ ] [ 60] ZooKeeper
server control bundle (1.2)
```

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

Service Locator server (Zookeeper server) configuration is the same as described in Section 2.8, “Service Locator SOAP Service”.

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 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`

Currently the Service Locator REST Service has these operations:

- 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 locator-rest-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:RegisterEndpointRequestType>
```

- **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** 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
```

- **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 fullfil 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 mediaType="application/json"
```

```
            element="ns:EndpointReferenceList" />
      </response>
    </method>
  </resource>
```

Example of request url with GET method:

```
locator/endpoints/{namespaceURI}localPart/
p=key1,value1;p=key2,value2;p=key3,value3
```

- **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 fullfills 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
```

GUI interface in Talend Enterprise ESB

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

Chapter 3. Service Locator configuration

Like any standard CXF feature, the Service Locator Feature is configured separately for the service provider side and service consumer side. The provider side Service Locator Feature extension registers and deregisters service endpoints in the endpoint repository when the provider becomes available or unavailable. The consumer side Service Locator Feature extension transparently retrieves service endpoint addresses from the endpoint repository when a service call to a provider is to be made.

The chapter describes in detail the Service Locator Feature Spring configuration.

3.1. Service Locator Provider configuration

To enable Locator feature import locator beans in Spring configuration file `<import resource="classpath:tesb/locator/beans.xml" />` for servlet container and `<import resource="classpath:tesb/locator/beans-osgi.xml" />` for OSGI container.

To add the Locator feature to a CXF service provider, use `<jaxws:features>` including the bean `org.talend.esb.servicelocator.cxf.LocatorFeature`.

Example 3.1. Service Locator Feature configuration for endpoint

```
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jaxws="http://cxf.apache.org/jaxws"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://cxf.apache.org/jaxws
http://cxf.apache.org/schemas/jaxws.xsd">
<import resource="classpath:META-INF/cxf/cxf.xml" />
<import resource="classpath:META-INF/tesb/locator/beans-osgi.xml" />
<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"/>
 </jaxws:features>
</jaxws:endpoint>
</beans>
```

In the example above you can see that locator client was added through configuration exactly the same way as a standard CXF feature using `<jaxws:features>`.

3.2. Service Locator Consumer configuration

To enable Locator feature, import locator beans in Spring configuration file `<import resource="classpath:tesb/locator/beans.xml" />` for servlet container and `<import resource="classpath:tesb/locator/beans-osgi.xml" />` for OSGI container.

To add the Locator feature to a CXF service consumer, use `<jaxws:client>` including the bean `org.talend.esb.servicelocator.cxf.LocatorFeature`.

Example 3.2. Service Locator Feature configuration for client

```
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jaxws="http://cxf.apache.org/jaxws"
xmlns:util="http://www.springframework.org/schema/util"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.0.xsd ">
<import resource="classpath:META-INF/cxf/cxf.xml" />
<import resource="classpath:/META-INF/tesb/locator/beans.xml" />
<jaxws:client id="greeterService" address="locator://GreeterService"
serviceClass="demo.common.Greeter">
 <jaxws:features>
  <bean
   class="org.talend.esb.servicelocator.cxf.LocatorFeature">
  </bean>
 </jaxws:features>
</jaxws:client>
</beans>
```

In the example above you can see that Locator client was added through configuration in exactly the same way as a standard CXF feature using `<jaxws:features>`. Another important point is to configure the JAX-WS client address. We must use the locator protocol for client: `address="locator://service_name"`.

3.3. Additional Metadata

Sometimes you need a more fine grained control of endpoints for a specific service a client gets when retrieving the endpoints. For this purpose you can define additional metadata for an endpoint, such as the country for which the endpoint is only valid or the bandwith 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 3.3. 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="bandwith" 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 `bandwith` with value `Class A`.

Example 3.4. 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.

3.4. Service Locator endpoint selection strategy configuration

Currently three endpoint selection strategies are supported: "defaultSelectionStrategy", "randomSelectionStrategy" and "evenDistributionSelectionStrategy". "defaultSelectionStrategy" keeps the endpoint as long as there is no failover, the other two strategies distribute the load at different endpoints: "randomSelectionStrategy" selects randomly from the available endpoints for each call, "evenDistributionSelectionStrategy" performs a client side round robin strategy. In case of a failover, all strategies are equivalent - a random alternative endpoint is selected. If multiple clients use "evenDistributionSelectionStrategy", it could happen that all clients choose subsequently the same endpoints since the locator instances for each client operate independently. "randomSelectionStrategy" avoids this problem.

Selection strategy globally on container level is configured in properties file as described below in the Section 3.5, “Properties file” section of the document by setting the "locator.strategy" property. If that property is not added in the configuration file, the "defaultSelectionStrategy" is chosen as default endpoint selection strategy.

Endpoint selection strategy can also be configured for each consumer by adding additional property in consumer configuration. For the consumer selection strategy setting, add "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>
```

3.5. Properties file

On Talend Runtime container, if you want to configure some properties of the locator feature, please edit:

`<Talend.runtime.dir>/container/etc/org.talend.esb.locator.cfg`

On Servlet Container, if you want to configure some properties, please edit the `locator.properties` in your classpath.

You can specify following properties in your configuration file:

Property name	Description
`locator.endpoints`	Specify the endpoints of all the instances belonging to the Service Locator ensemble the Service Locator client might be talking to. The Service Locator client will pick an endpoint one by one (the order is non-deterministic) to connect to the Service Locator until a connection is established. If the property is not set, the default localhost endpoint `"localhost:2181`
`endpoint.http.prefix`	necessary when running in a container where the endpoint is only relative to the container. e.g. http://localhost:8040/services, By default prefix is an empty. string.
`endpoint.https.prefix`	necessary when running in a container where the endpoint is only relative to the container and secured, for example: "https://localhost:9001/services". By default, `prefix` is an empty string.
`locator.strategy`	Three endpoint selection strategies are supported: "defaultSelectionStrategy", "randomSelectionStrategy" and "evenDistributionSelectionStrategy".
`locator.reloadAdr essesCount`	This parameter is not only relevant for `locator.strategy=evenDistributionSelectionStrategy` and `locator.strategy=randomSelectionStrategy`. These strategies cache the list of endpoints returned by the locator for a fixed number of service calls determined by parameter "locator.reloadAdressesCount". After these calls, the list of available addresses is refreshed. Set this parameter to a high value to reduce the number of locator calls in case your services are reliable and a failover occurs seldom.

Property name	Description
`connection.timeout`	specify the time the Service Locator client waits for a connection to get established. Must be greater than zero, by default 5000 ms.
`session.timeout`	specify the time out of the session established with the server. The session is kept alive by requests sent by the client. If the session is idle for a period of time that would timeout the session, the client will send a ping request to keep the session alive. Must be greater than zero and less than 60000, by default 5000 ms.

Locator feature properties config sample:

```
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
```

3.6. Service Locator for RESTful services

The Service Locator feature can be used for both SOAP and RESTful Web Services.

The Service Locator configuration for web services using the REST architectural style is similiar to the SOAP services configuration as described in previous sections.

To add the Locator feature to a RESTful service provider, use `<jaxrs:features>` including the bean `org.talend.esb.servicelocator.cxf.LocatorFeature`.

Example 3.5. Service Locator Feature RESTful service provider configuration

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

 <import resource="classpath:META-INF/cxf/cxf.xml" />
 <import resource="classpath:META-INF/tesb/locator/beans-osgi.xml" />


 <bean id="orderService" class="demo.service.OrderServiceImpl">
 </bean>

 <jaxrs:server id="orderRESTService" address="/rest">
  <jaxrs:features>
   <bean id="orderServiceLocator"
   class="org.talend.esb.servicelocator.cxf.LocatorFeature"/>
  </jaxrs:features>
  <jaxrs:serviceBeans>
   <ref bean="orderService" />
  </jaxrs:serviceBeans>
 </jaxrs:server>
</beans>
```

To add the Locator feature to a CXF service consumer, use `<jaxrs:client>` including the bean `org.talend.esb.servicelocator.cxf.LocatorFeature`.

Example 3.6. Service Locator RESTful service consumer configuration

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

<import resource="classpath:META-INF/cxf/cxf.xml"/>
<import resource="classpath:META-INF/tesb/locator/beans.xml" />

  <jaxrs:client id="restClient"
        address="locator://some_usefull_information"
        serviceClass="demo.common.OrderService"
        xmlns:serviceNamespace="http://service.demo/"
        serviceName="serviceNamespace:OrderServiceImpl"
        inheritHeaders="true">
        <jaxrs:headers>
            <entry key="Accept" value="application/xml"/>
        </jaxrs:headers>
 <jaxrs:features>
  <bean class="org.talend.esb.servicelocator.cxf.LocatorFeature">
   <property name="selectionStrategy"
   value="evenDistributionSelectionStrategy"/>
  </bean>
 </jaxrs:features>
 </jaxrs:client>
   </beans>
```

As you can see, in the example above `<jaxrs:client>` was configured by setting the `serviceName` attribute. We need this service name to discover the endpoint from the Locator server. Please note the `serviceName` attribute specifies a service `QName`, here `xmlns:serviceNamespace="http://service.demo/" serviceName="serviceNamespace:OrderServiceImpl"`

The `locator` protocol in the `address` attribute is used to enable the Locator feature.

Chapter 4. Service Activity Monitoring (SAM)

4.1. Introduction

The Service Activity Monitoring (SAM) component allows for logging and monitoring service calls made with the Apache CXF framework. For example, Service Activity Monitoring could be used for collecting usage statistics and fault monitoring. This component 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:

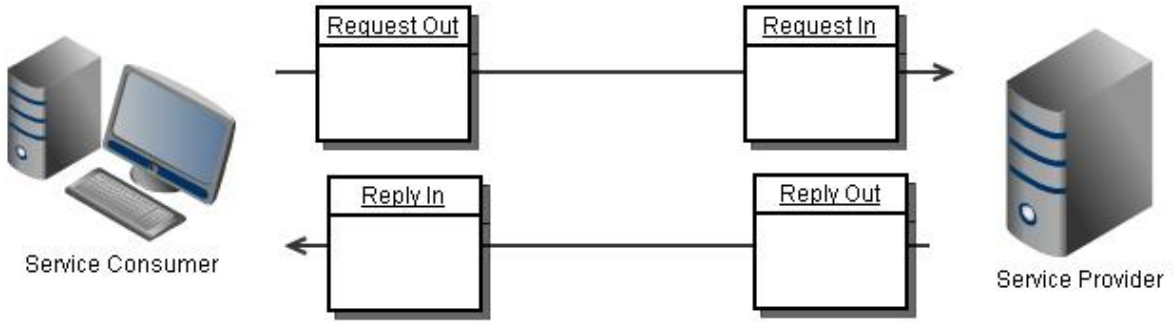
1. The Agent creates events out of requests and replies from both the service consumer and provider side.
2. The events are first collected locally and then sent to the Service Activity Monitoring Server periodically (to not disturb the normal message flow).
3. When the server receives events from the Agent, it optionally uses filters and/or handlers on those events and stores them into 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 (tesb:start-sam).
- Alternatively, the Service Activity Monitoring Server can be deployed as a WAR in a servlet container and needs access to a database.
- 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.

4.1.1. Messages, Events and Flow IDs



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 service call, on 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 the Section 4.2, “Architecture”.

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.

Note

The value of collector.lifecycleEvent property must be set to true if you want to generate/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

4.1.2. Structure of this chapter

In Section 4.2, “Architecture”, we overview the architecture of how a Service Activity Monitoring Agent and Server interact.

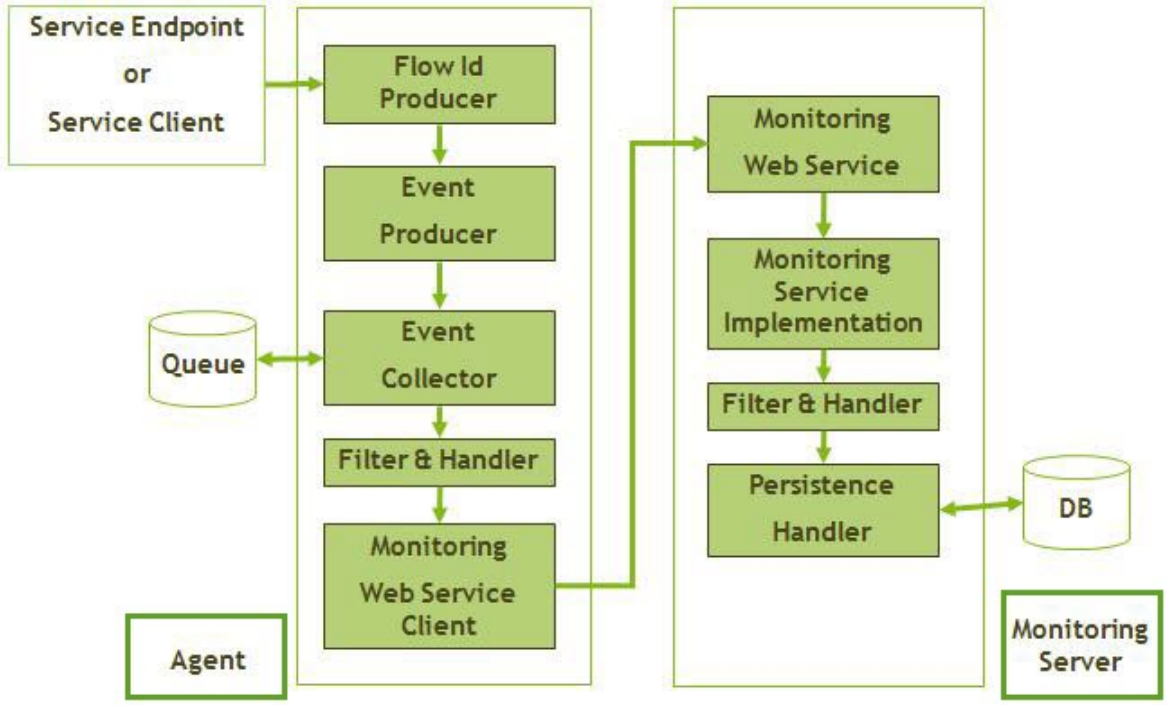
In Section 4.3, “Installation”, we look at installing a Service Activity Monitoring Agent and Server into a customer application.

In Section 4.4, “Configuration”, we look at configuration files and values for Service Activity Monitoring Agents and Servers.

In Section 4.5, “Running and Testing”, we look at running and testing the system.

4.2. Architecture

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 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 Section 4.6, “Event Structure”.

4.3. Installation

The Service Activity Monitoring installation includes Agent side installation and Server side installation. 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 Servlet container or OSGi Container.

Multiple instances of Service Activity Monitoring Server

You can have multiple instances of the Service Activity Monitoring Server running on a machine at the one time.

If you want to use the same Service Activity Monitoring Server from multiple containers, please update the `service.url` property in file `<Talend.runtime.dir>/container/etc/org.talend.esb.sam.agent.cfg` in each container. (See Section 4.4.1, “Agent Configuration”).

4.3.1. Agent Installation in a Servlet container

Installing an Agent in a Servlet container (for example, Apache Tomcat, 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: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.

4.3.2. Agent Installation in an OSGi Container

Installing the Agent in an OSGi Container (for example, Talend Runtime container):

- Start the Talend Runtime container and type in the following commands to install the Agent bundle:

  ```
  features:addurl mvn:org.talend.esb/features/5.1.0/xml
  ```

```
features:install tesb-sam-agent
```

Then, the Agent will be installed into Talend Runtime container.

4.3.3. DataSource Installation

DataSource installation is a prerequisite to Service Activity Monitoring installation. There are several out-of-box DataSource features which can be installed into Talend OSGi container or when using J2EE/Tomcat. Here are the instructions to install the JNDI DataSource:

4.3.3.1. Installing the DataSource in an OSGi container

Type in the following command on the Talend Runtime container console:

```
features:install tesb-datasource-<Database>
```

The corresponding DataSource will be installed into the container and a configuration file named `org.talend.esb.datasource.<Database>.cfg` will be created in the `<Talend.runtime.dir>/container/etc` folder.

For example, to install the Derby DataSource:

1. Execute the following command:

   ```
   features:install tesb-datasource-derby
   ```

2. On the Talend Runtime container console, execute the **`list`** command, you will find the installed bundles and configuration of Derby driver:

   ```
   [225] [Active] [       ] [ ] [60] Apache Derby 10.8 (10.8.1000002.1095077)
   [226] [Active] [Created] [ ] [60] Service Activity Monitoring :: Datasource-
   derby (5.1.0)
   ```

   The `org.talend.esb.datasource.derby.cfg` configuration file has been created into the `<Talend.runtime.dir>/container/etc` folder. In this configuration file, you can change the Database settings dynamically. For example, the default properties of `org.talend.esb.datasource.derby.cfg` are:

   ```
   datasource.server=localhost
   datasource.port=1527
   datasource.database=db
   datasource.createdatabase=create
   datasource.user=test
   datasource.password=test
   ```

Here is a table with the DataSource information for other databases which work with the Talend Runtime container:

DataSource Name	Database	Database, Driver Version	Feature	ConfigFile
ds-derby	Derby	10.8, 10.8.1.2	tesb-datasource-derby	org.talend.esb.datasource.derby.cfg
ds-h2	H2 Engine	1.3, 1.3.165	tesb-datasource-h2	org.talend.esb.datasource.h2.cfg

DataSource Name	Database	Database, Driver Version	Feature	ConfigFile
ds-mysql	MySQL	5.1, 5.1.18	tesb-datasource-mysql	org.talend.esb.datasource.mysql.cfg
ds-oracle	Oracle	11.2.0, 11.2.0.2.0	tesb-datasource-oracle	org.talend.esb.datasource.oracle.cfg
ds-db2	IBM DB2	9.7, 9.7	tesb-datasource-db2	org.talend.esb.datasource.db2.cfg
ds-sqlserver	SQL Server	2008R2, 3.0	tesb-datasource-sqlserver	org.talend.esb.datasource.sqlserver.cfg

Driver versions

Other versions of the drivers may work, but have not been tested. See Section 1.1, “Prerequisites to using Talend ESB” for general information on software prerequisites.

Oracle, DB2 and SQLServer drivers in Maven

As Oracle, DB2 and SQLServer do not publish the driver in the Maven repository, users have to explicitly install the driver into local Maven repository before installing the DataSource feature.

Here is the installation instructions for each of these:

- Oracle: mvn install:install-file -Dfile="C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6.jar" -DgroupId=ojdbc -DartifactId=ojdbc -Dversion=11.2.0.2.0 -Dpackaging=jar
- DB2: mvn install:install-file -Dfile="C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc.jar" -DgroupId=com.ibm.db2.jdbc -DartifactId=db2jcc -Dversion=9.7 -Dpackaging=jar
- SQLServer: mvn install:install-file -Dfile="" -DgroupId=com.microsoft.sqlserver -DartifactId=sqljdbc4 -Dversion=3.0 -Dpackaging=jar

4.3.3.2. 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 the Resource entry 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;databa
seName=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"/>
```

4.3.4. Service Activity Monitoring Server Installation

The Service Activity Monitoring Server can be installed into a Servlet container or an OSGi Container. It supports Apache Derby, MySQL, Oracle, SQL Server, IBM DB2 and H2 Database Engine to store Events data.

4.3.4.1. Database installation and initialization

This section describes database initialization.

1. Make sure your chosen database is installed properly and is accessible.
2. Login with a user which has CREATE permissions and run the "init SQL" script for the corresponding database (see table below).

   The script files for the corresponding databases are described in the following table. You can find the SQL scripts in the `<Talend.runtime.dir>/add-ons/sam/db` directory

SQL script filename	Database
create.sql	Apache Derby
create_mysql.sql	MySQL
create_oracle.sql	Oracle
create_sqlserver.sql	SQL Server
create_h2.sql	H2 Database Engine
create_db2.sql	IBM DB2

You will then find the EVENTS and EVENTS_CUSTOMINFO table have been created in your database.

Warning

If the value of `db.recreate` property in `logserver.properties` is set to true, the "init SQL" script will be executed automatically when starting the Service Activity Monitoring Server.

However, this is NOT recommended for any database except Apache Derby running in embedded mode.

For Oracle database, this causes expected exceptions during starting the Service Activity Monitoring Service the first time, which can be ignored. *For production databases it causes deleting all data from database after restarting Service Activity Monitoring server*.

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.

4.3.4.2. Install the Service Activity Monitoring Server into a Servlet container

The Service Activity Monitoring Server can be deployed into any Servlet container as a WAR. For example, to deploy into Tomcat:

`copy <Talend.runtime.dir>\add-ons\sam\sam-server-war.war $TOMCAT_HOME\webapps` (Windows)

`cp <Talend.runtime.dir>/add-ons/sam/sam-server-war.war $TOMCAT_HOME/webapps` (Linux)

And to start Apache Tomcat:

`$TOMCAT_HOME\bin\startup.bat` (Windows)

`./$TOMCAT_HOME/bin/startup.sh` (Linux)

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 configurated in the J2EE/Tomcat container. You will find how to configure the database properties in Section 4.4, “Configuration”.

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

```
cd <Talend.runtime.dir>/examples/talend/tesb/sam/sam-server-jetty
```

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

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

4.3.4.3. Install the Service Activity Monitoring Server into the OSGi Container

Note

Be sure the DataSource feature for your prefered Database has been installed in the container before installing the Service Activity Monitoring Server.

For your 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
```

These are a shortcut. Here we give the expanded version of these, for using with Talend Runtime container or another OSGi container:

Install the Service Activity Monitoring Server, type in these commands on the console:

```
features:addurl mvn:org.talend.esb/features/5.1.0/xml
```

```
features: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

4.3.5. Example Installation

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/.

4.4. Configuration

4.4.1. Agent Configuration

The main configuration files for Agents are agent.properties and filter, handler configuration files. The agent.properties can be created by user and put it into where classpath included. Filter and handlers are based on Spring bean configuration and you can add them into your application's context (for example, beans.xml).

If the Agent has been installed into OSGi Container, the configuration file is located in the <Talend.runtime.dir>/container/etc/org.talend.esb.sam.agent.cfg in the container.

Properties description:

Property	Default	Description
collector.scheduler.interval		Interval(in milliseconds) of Agent built-in scheduler. Agent will make one or several calls to the Service Activity Monitoring Server sending Events from local queue every interval milliseconds. How many calls 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 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. The purpose is to avoid send large size of soap message body to the sam-server in one call.
collector.lifecycleEvent	false	An on-off switch used for Agent to decide if it is to collect and send the lifecycle events to the Service Activity Monitoring Server. If true, the Service Activity Monitoring Server must have been started before the Talend Runtime container is started, otherwise, the Connection Exceptions will be thrown.
log.messageContent	true	An on-off switch used for Agent to decide if the SOAP message content from the Provider/Consumer should be stored into Event and sent to Service Activity Monitoring Server.
log.maxContentLength	-1	The value of this parameter is used to set maximum SOAP content length per Event. -1 is unlimited.
log.enforceMessageIDTransfer	true	if enforceMessageIDTransfer=true, SAM will add WS-Addressing funtionality implicitly and enforce MessageID transfer between the Events; if enforceMessageIDTransfer=false (Default value), the MessageID will be null in the Events if the user doesn't enable the WSAddressingFeature or Policy with Addressing explicitly.
service.url		The URL of Service Activity Monitoring Server which the Agent will 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 you can add pre-defined or your own filters and handlers to the Agent. Put the filter or handler beans into any Spring configuration file. Some example bean definitions can be found below:

Example 1: filters out all messages that contain "contractor" in the body:

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

Example 2: the Passwordhandler replaces <Password> tags with <Replaced>:

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

4.4.2. DataSource Configuration

In case of OSGi container, you can configure the database information for every DataSource installed. Currently, there are six built-in DataSource for frequently used Database (Derby, H2, MySQL, Oracle, DB2 and SQLServer). Each configuration file has its own properties, here is an example of the Derby configuration file: `<Talend.runtime.dir>/container/etc/org.talend.esb.datasource.derby.cfg`.

```
datasource.server=localhost
datasource.port=1527
datasource.database=db
datasource.createdatabase=create
datasource.user=test
datasource.password=test
```

Please also see Section 4.3.3.1, “Installing the DataSource in an OSGi container” for information on DataSource installation.

4.4.3. Service Activity Monitoring Server Configuration

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

If the Service Activity Monitoring Server has been installed into OSGi Container, the configuration file is located in `<Talend.runtime.dir>/container/etc/org.talend.esb.sam.server.cfg`.

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/ Tomcat, it should be like java:comp/env/<Resource name>, for OSGi container, it should be like ds-<Database>.
db.dialect	derbyDialect	Which DB used to store/query Event data (with different ID Incrementer, Query, and so on)
db.recreate	true	Whether re-create tables in database
db.createsql	create.sql	SQL file name which used to create tables in database

`logserver.properties` example (for Derby):

```
monitoringServiceUrl=/MonitoringServiceSOAP

db.datasource=ds-derby (for Tomcat, value should be: java:comp/env/jdbc/datasource)
db.dialect=derbyDialect
db.recreate=true
db.createsql=create.sql
```

logserver.properties example (for H2 Database Engine):

```
monitoringServiceUrl=/MonitoringServiceSOAP

db.datasource=ds-h2 (for Tomcat, value should be: java:comp/env/jdbc/datasource)
db.dialect=h2Dialect
db.recreate=false
db.createsql=create_h2.sql
```

logserver.properties example (for Mysql):

```
monitoringServiceUrl=/MonitoringServiceSOAP

db.datasource=ds-mysql (for Tomcat, value should be like: java:comp/env/jdbc/datasource
db.dialect=mysqlDialect
db.recreate=false
db.createsql=create_mysql.sql
```

logserver.properties example (for Oracle):

```
monitoringServiceUrl=/MonitoringServiceSOAP
```

```
db.datasource=ds-oracle (for Tomcat, value should be like: java:comp/env/jdbc/datasourc
db.dialect=oracleDialect
db.recreate=false
db.createsql=create_oracle.sql
```

logserver.properties example (for IBM DB2):

```
monitoringServiceUrl=/MonitoringServiceSOAP

db.datasource=ds-db2 (for Tomcat, value should be like: java:comp/env/jdbc/datasource )
db.dialect=DB2Dialect
db.recreate=true
db.createsql=create_db2.sql
```

logserver.properties example (for SQL Server):

```
monitoringServiceUrl=/MonitoringServiceSOAP

db.datasource=ds-sqlserver (for Tomcat, value should be like: java:comp/env/jdbc/dataso
db.dialect=sqlServerDialect
db.recreate=true
db.createsql=create_sqlserver.sql
```

For filter, handler configuration, please refer to Section 4.4.1, “Agent Configuration”.

4.5. Running and Testing

4.5.1. Pre-requisites

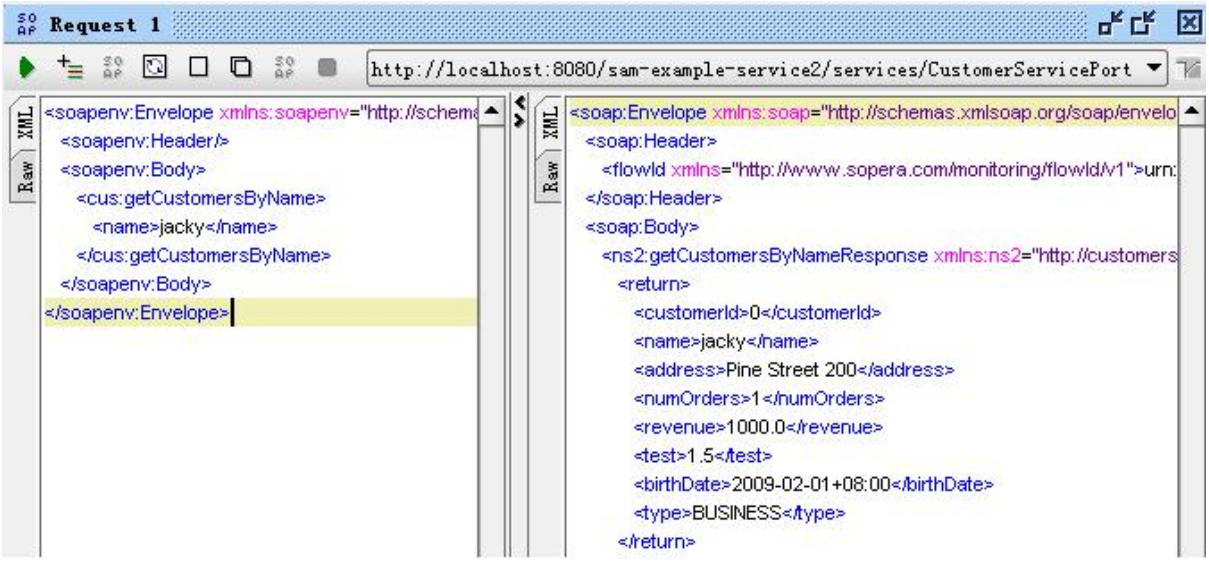
This section shows you how to run the examples (sam-example-service, sam-example-service2) with the SAM Agent supported. First, please check the following:

- 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 your container.
- the configuration files (`agent.properties`, `logserver.properties`, and so on) are configured correctly.

4.5.2. General Test

Start SoapUI tool, send the SOAP message below to sam-example-service2 endpoint, for example like this: `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>
```



4.5.3. 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 need 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-usernametoken-profile-1.0#PasswordDigest">
      IR55epSSTb7sg3Z3+HKNb9MqAWg=</wsse:Password>
   </wsse:UsernameToken>

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

The value of <Password> Element will 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/value 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.

☐	80102	80101	Application name	Dummy App
☐	80103	80101	Stage	Dev

3. Finally, add filter configuration on the Service Activity Monitoring Server side

Modify server.xml on the Service Activity Monitoring Server. 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>
  ......
```

4. The information should now be stored in the database.

4.5.4. Monitoring events from database

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

☐	80068	<soapenv:Envelope x...	317B	(NULL)	OK	2011-03-14 13:59:18	REQ_IN	(NULL)	7240@DaiXilai	DaiXilai	192.168.0.54
☐	80069	<soap:Envelope xmln...	362B	(NULL)	OK	2011-03-14 13:59:22	REQ_OUT	(NULL)	7240@DaiXilai	DaiXilai	192.168.0.54
☐	80070	<soap:Envelope xmln...	798B	(NULL)	OK	2011-03-14 13:59:22	RESP_IN	(NULL)	7240@DaiXilai	DaiXilai	192.168.0.54
☐	80071	<soap:Envelope xmln...	798B	(NULL)	OK	2011-03-14 13:59:22	RESP_OUT	(NULL)	7240@DaiXilai	DaiXilai	192.168.0.54
☐	80072	<soap:Envelope xmln...	362B	(NULL)	OK	2011-03-14 13:59:22	REQ_IN	(NULL)	7240@DaiXilai	DaiXilai	192.168.0.54
☐	80073	<soap:Envelope xmln...	798B	(NULL)	OK	2011-03-14 13:59:22	RESP_OUT	(NULL)	7240@DaiXilai	DaiXilai	192.168.0.54
☐	80074	<soapenv:Envelope x...	293B	(NULL)	OK	2011-03-15 14:38:11	REQ_IN	(NULL)	5056@DaiXilai	DaiXilai	192.168.0.54
☐	80077	<soap:Envelope xmln...	802B	(NULL)	OK	2011-03-15 14:38:11	RESP_OUT	(NULL)	5056@DaiXilai	DaiXilai	192.168.0.54

GUI interface in Talend Enterprise ESB

If you have Talend Enterprise ESB, there is GUI functionality provided by the Talend Administration Center, for viewing the Service Activity Monitoring information. Please see **Talend Enterprise ESB Installation Guide** and **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.

4.6. Event 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 message content which come from Service Provider/Service Consumer. *Note: It will be null for all Lifecycle Events.*
EI_TIMESTAMP	datetime	Timestamp which the Event 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 be used currently
ORIG_PROCESS_ID	varchar(255)	Process id is the OS process id
ORIG_HOSTNAME	varchar(128)	The name of Host which the SAM agent running

Field	Type	Description
ORIG_IP	varchar(64)	The IP address which the SAM agent running
MI_PORT_TYPE	varchar(255)	Service port type which enabled the SAM agent. *Note: It will be null for SERVER_START/SERVER_STOP Events.*
MI_OPERATION_NAME	varchar(255)	Service operation name which enabled the SAM agent. *Note: It will be null for all Lifecycle Events.*
MI_MESSAGE_ID	varchar(255)	the MessageID which is generated/transfered using the CXF Addressing feature. According to the common definition of the MessageId in the WS-Addressing Spec, REQ_OUT and REQ_In are the same message, they should have the same MessageId; RESP_OUT and RESP_IN are the same message, they should have the same MessageId. Note: 1. The MessageID will be null for all Lifecycle Events. 2. If log.enforceMessageIDTransfer=false and doesn't enable the WSAddressingFeature or Policy with Addressing explicitly, it also will be null. 3. It will be null for REQ_IN/RESP_OUT if WS-Addressing feature only enabled on the provider side not enabled on the consumer side.
MI_FLOW_ID	varchar(64)	Unique id (UUID) for the message flow. All events with the same id belong together. *Note: It will be null for all Lifecycle Events.*
MI_TRANSPORT_TYPE	varchar(255)	Transport type of event. *Note: It will be null for all Lifecycle Events.*
ORIG_PRINCIPAL	varchar(255)	Principal info in the message header. *Note: It will be null for all Lifecycle Events.*
CONTENT_CUT	tinyint(1)	Flag, if the event content has been cut from the Agent. *Note: It will be null for all Lifecycle Events.*

4.7. 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)	custom property's key, for example, Application name
CUST_VALUE	varchar(255)	custom property's value, for example, Dummy App

Chapter 5. 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.

Note

We use the term `<Talend.runtime.dir>` for the directory where Talend Runtime is installed. This is typically the full path of either `Runtime_ESBSE` or `Talend-ESB-V5.1.x`, depending on the version of the software that is being used. Please substitute appropriately.

5.1. Deploying the STS into the Talend Runtime container

Warning

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 (STS) in the Talend Runtime, we need to deploy it into a Talend Runtime container:

1. Replace the STS' sample keystore/truststore called `stsstore.jks` located in the <Talend.runtime.dir>/container/etc/keystores folder with your own keystore. See Section 5.3, “Security Token Service (STS) Configuration” for more information.

2. cd <Talend.runtime.dir>/container/bin directory, enter **trun** to start Talend Runtime, a Talend Runtime container (Karaf) console window will open.

3. In the console, type **features:install tesb-sts** to install the Security Token Service component.

4. Type **list | grep STS** in the console. You should see the 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 (5.1.0)
```

The above shows that Security Token Service (STS) component is enabled in the Talend Runtime container. The Fragment Bundle `204: Talend :: ESB :: STS :: CONFIG (5.1.0)` provides the custom configuration about the Security Token Service (STS), which will be described in Section 5.3, “Security Token Service (STS) Configuration”.

5.2. Deploying the STS into a Servlet Container (Tomcat)

Warning

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 (STS) using a servlet container (here we are using Tomcat as an example) follow the below steps:

1. Extract the `<Talend.runtime.dir>/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'll see several Security Token Services available, such as Username Token service (UT), X.509 Token service, etc.

4. Enter URL: `http://{tomcat host}:port/SecurityTokenService/UT?wsdl`, the displayed WSDL file will describe the details about the Security Token Service.

5.3. Security Token Service (STS) 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, etc.

In Talend Runtime container, the configuration of Security Token Service (STS) can be defined in the file:

```
<Talend.runtime.dir>/etc/org.talend.esb.sts.server.cfg
```

```
    stsServiceUrl=/SecurityTokenService/UT
    jaasContext=karaf
    signatureProperties=file:${tesb.home}/etc/keystores/stsKeystore.properties
    signatureUsername=mystskey
    bspCompliant=false
```

By default STS is configured to use JAAS interface to verify the user credentials and perform authentication. As shown above, STS uses *`karaf`* JAAS Context which is the default context configured for Talend Runtime container and uses *`PropertiesLoginModule`* of Karaf. This login module uses users.properties file located in `/etc/users.properties` which contains a list of users and their passwords, hence the users which are needed to be authenticated via the STS should be listed here. A different login module can be configured for the STS 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.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=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.

5.4. 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: `<Talend.runtime.dir>/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
ws-security.sts.token.username = myclientkey
ws-security.sts.token.usecert = true
ws-security.is-bsp-compliant = false
ws-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 `<Talend.runtime.dir>/etc/org.talend.esb.job.token.policy`.
- For SAML tokens, 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 `<Talend.runtime.dir>/etc/org.talend.esb.job.saml.policy`.

When using Username tokens, 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 `users.properties` file located in `/etc/users.properties` which contains a list of users and their passwords. 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 `<Talend.runtime.dir>/etc/org.talend.esb.job.service.cfg`.

```
ws-security.signature.properties = \
   file:${tesb.home}/etc/keystores/serviceKeystore.properties
ws-security.signature.username = myservicekey
ws-security.signature.password = skpass
```

5.5. 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.

5.5.1. Using OpenSSL to create certificates

First, create the keys.

Note

Replace "<PW-Sk>", "<PW-Sk>","<PW-Cs>" and "<PW-Ck>" in the example below with your own passwords.

5.5.1.1. 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
```

5.5.1.2. 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
```

5.5.2. 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:

Note

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 6. Apache Archiva and the Talend Artifact Repository

This chapter discusses configuring Apache Archiva in Talend ESB. Most of this chapter refers to Talend ESB Standard Edition. For completeness, it also includes a brief overview of Talend Artifact Repository, which is repository management software, based on Apache Archiva, and preconfigured to be used directly within the Talend Enterprise ESB.

Rent-a-car example

In the **Talend ESB Getting Started User Guide**, there is an example of using Archiva and the Talend Artifact Repository to upload and deploy services in the Rent-a-Car demo.

This chapter mainly focuses on aspects of Archiva that are typically used in Talend ESB development.

6.1. Overview

Apache Archiva is extensible repository management software.

The Talend Artifact Repository is based on Apache Archiva, and provides a standard-based repository. It is used within the Talend Enterprise 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 Archiva ones.

It is available in the Talend Enterprise ESB as a zip file as part of the Talend Administration Center download.

The 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.

The Talend Administration Center uses the Talend Artifact Repository to store and to provide the deployment artifacts for the Talend Runtime container, and their user interfaces are linked for ease of use.

6.2. More information

For more information on Apache Archiva, see `http://archiva.apache.org/`.

Talend Administration Center is available **only for Talend Enterprise ESB**.

For information on the Talend Artifact Repository, see **Talend Enterprise ESB Installation Guide**, and also the fully-worked example in **Talend ESB Infrastructure Services Configuration Guide**.

For information on the Talend Administration Center, see **Talend Enterprise ESB Installation Guide** and **Talend Administration Center User Guide**.

The remainder of this chapter focuses on using Apache Archiva, and configuring Maven to access and deploy to Archiva repositories.

Maven version

Please note that Maven 3.0.3 is required for the functionality described in this document.

6.3. Downloading and installing Archiva

6.3.1. Talend ESB Standard Edition

If you are using Talend ESB Standard Edition, then download Apache Archiva from `http://archiva.apache.org` and extract it. In the Archiva directory, run:

```
./bin/archiva console (Linux)
.\bin\archiva.bat console (Windows)
```

Archiva is now running on `http://localhost:8080/archiva/`.

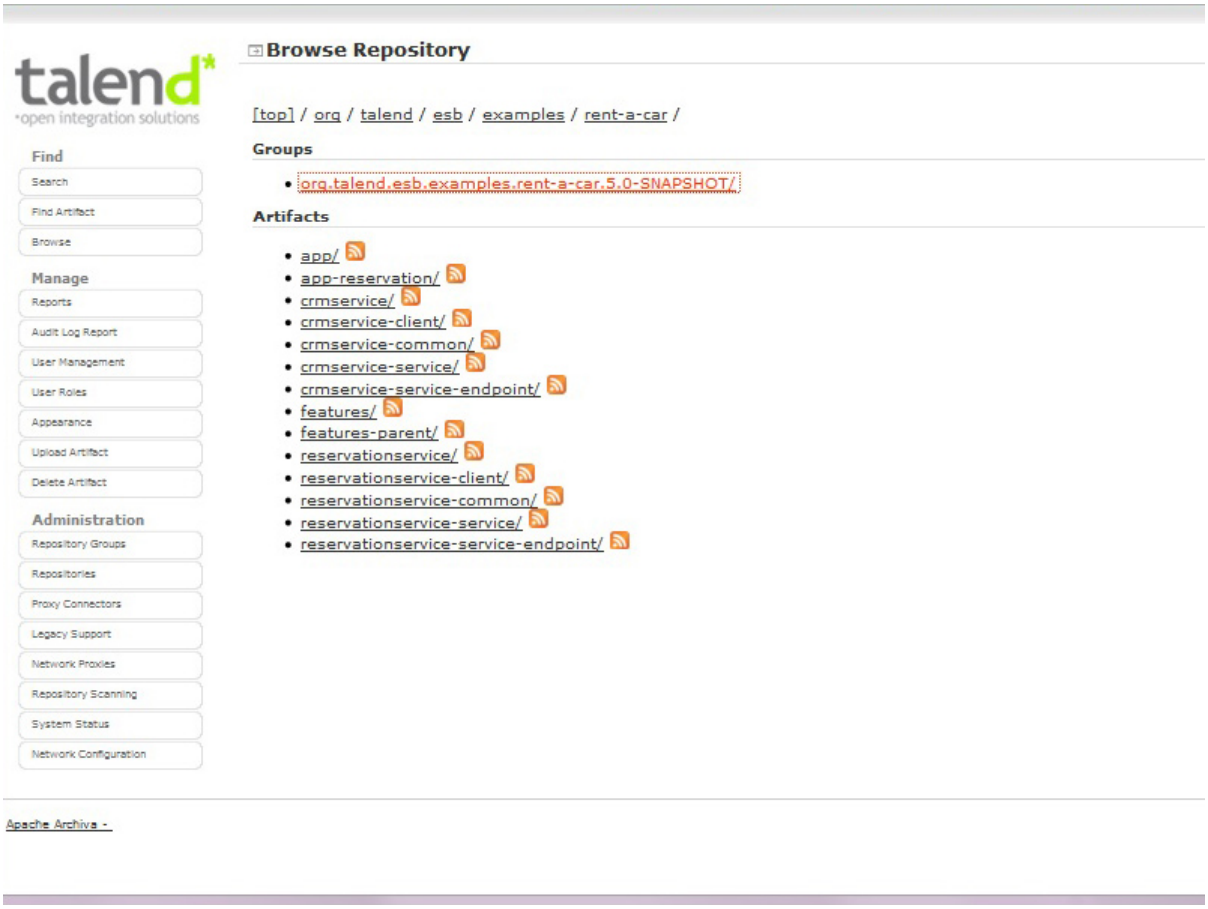
6.3.2. Talend Enterprise ESB

If you are using Talend Enterprise ESB, then see **Talend Enterprise ESB Installation Guide** for details on how to install the Talend Artifact Repository (based on Achiva).

The Talend Artifact Repository is now running on `http://localhost:8082/archiva/` with **`user: tadmin pwd: tadmin`**.

6.4. Browsing repositories

Figure 6.1. A repository with some Talend artifacts already deployed



6.4.1. Permissions

The user can only browse those repositories where the user is an observer or a manager. If the user does not have permission to access any repository, a message saying "You have access to no repositories. Ask your system administrator for access" will be displayed.

6.4.2. Artifact Info

Items in the repositories are hyperlinked allowing you easy access to view more information. By clicking on the Group Id or Artifact Id you will be taken to the repository browser. 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 just 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 easily navigate 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.

6.4.3. 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.

6.4.4. 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.

6.5. Configuring Maven to use an Archiva repository

To get your local Maven installation to use an Archiva proxy you 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.

6.5.1. 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>
```

Example

An example of this is given in the Archiva section in the **Talend ESB Getting Started User Guide**.

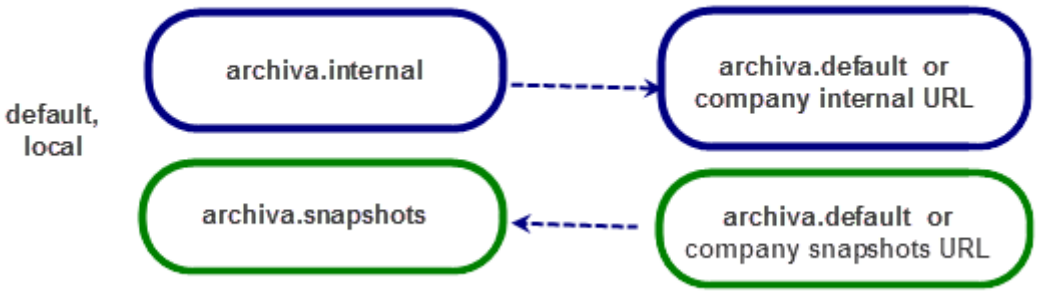
6.6. Deploying to a Repository

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.

6.6.1. Configuring Maven to deploy to an Archiva repository

Figure 6.2. 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 respositories).
2. The deployment user needs the Role 'Repository Manager' for each repository that you want 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>
```

6.6.2. Deploying to Archiva using HTTP

Configure the distributionManagement part of your `pom.xml` (customising the URLs as needed).

Matching ids

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/snapshot
s/</url>
      </snapshotRepository>
   </distributionManagement>
   ...
</project>
```

6.6.3. Deploying to Archiva using WebDAV

In some cases, you may want to use WebDAV (Web-based Distributed Authoring and Versioning) to deploy instead of HTTP. If you find this is necessary, to facilitate greater ease of collaboration, then 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
nternal/</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>
```