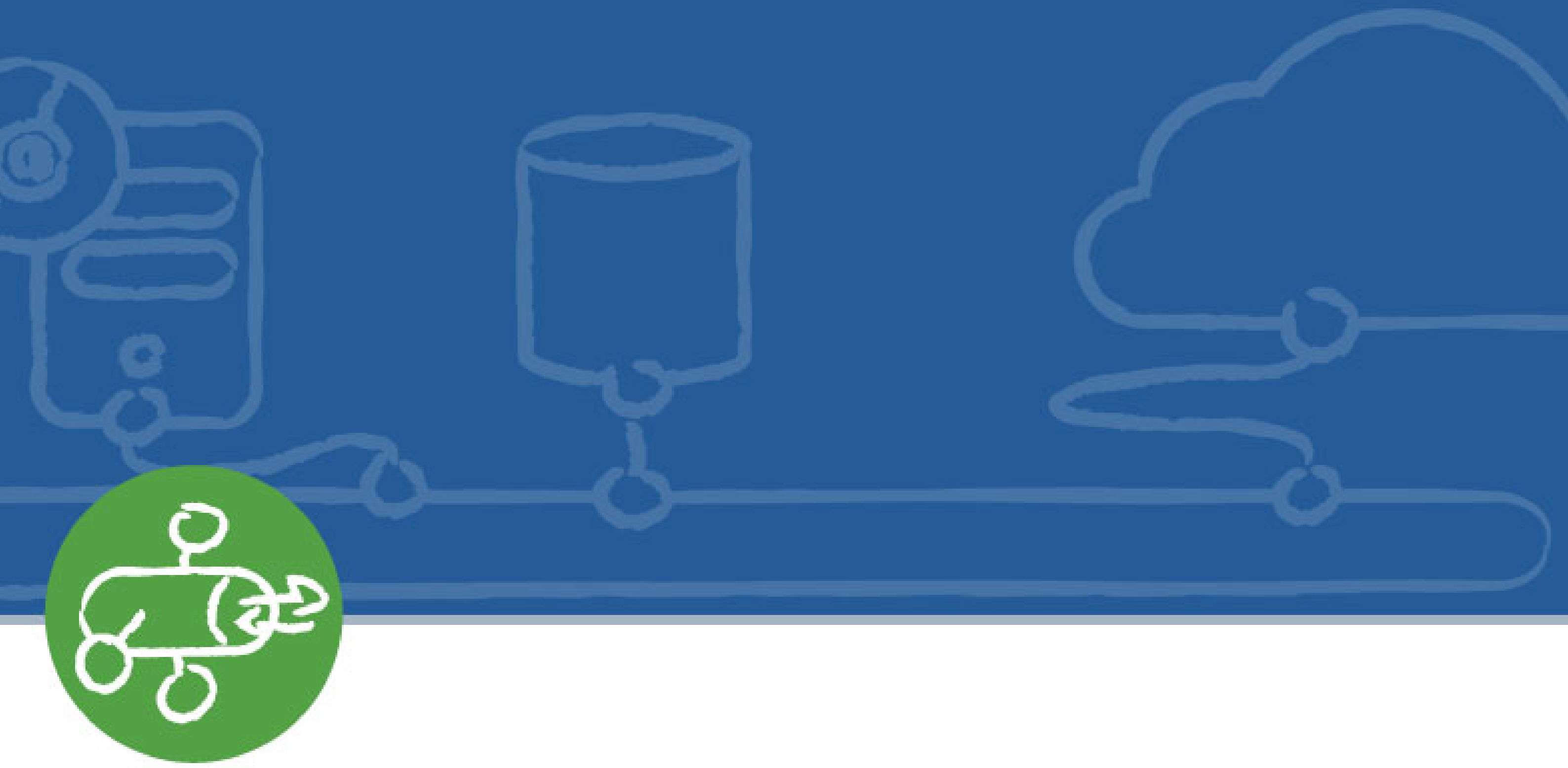
talend*
*open integration solutions



Talend ESB

Getting Started Guide

5.2.1

Publication date 12 November 2012

Copyleft

Notices

Table of Contents

List of Figures

Chapter 1. Talend ESB Products and Architecture

The Enterprise Service Bus (ESB) has always been the cornerstone of every vendor's Service Oriented Architecture (SOA) strategy. Talend ESB is a considerable improvement on previous ESBs in that it has:

- relatively small footprint
- uses proven open source technologies
- easy integration of existing applications and infrastructure

This chapter gives a high level overview of Talend ESB, its components and features, including an OSGi container, support for Web Services, and Studio.

It also describes the packages that are available, and details of the architecture of each of them.

1.1. Overview

Talend ESB is a lightweight, robust, modular solution for integrating new or existing applications. Leveraging the key integration projects from the Apache Software Foundation (ASF) (for example, there are 200,000 downloads per month for Apache CXF), Talend ESB is designed for both creating secure Web Services and also building enterprise class integration solutions.

Talend participates in the development of ESB components through the Apache community. Talend is actively working with a large number of developers in the Apache community and has made major contributions to many Apache projects. This includes critical fixes and enhancements for our customers, but also continuous delivery of new features and releases. These are glued together to provide a seamless experience for end users, developers and administrators.

Talend ESB delivers:

- Standards based web services stack

- A high performance message broker
- Enterprise-class security
- Flexible deployment options
- Developer tooling for Eclipse
- Centralized management and monitoring
- Integration and mediation user interface
- Service Enablement - support for REST, SOAP web services
- Mediation and Routing

1.2. Advanced Features

The following advanced features are available in Talend ESB:

- Service Locator - this provides automatic and transparent failover and load balancing between service Consumers and Providers through a newly developed extension that allows for dynamic endpoint registration and lookup via Apache Zookeeper. See Section 6.1, “*Technical overview of the Service Locator*”, Chapter 6, *Enabling the Service Locator and Service Activity Monitoring for the demo* and the *Talend ESB Infrastructure Services Configuration Guide* for more details.
- Service Activity Monitoring (SAM) facilitates the capture of analysis of service activity, including service response times, traffic patterns, auditing and more, by capturing events and storing information - see Section 6.2, “*Technical overview of Service Activity Monitoring*”, Chapter 6, *Enabling the Service Locator and Service Activity Monitoring for the demo* and the *Talend ESB Infrastructure Services Configuration Guide*.
- Hyperic HQ Plugins, providing visibility of Talend ESB CXF and Camel endpoints - see the *Talend ESB System Management Integration User Guide*.
- Security Token Service Framework - supports Security Assertion Markup Language 2.0 (SAML 2.0) to federate security credentials. This allows clients and services to securely and transparently authenticate during connections without the need for custom coding. See Chapter 7, *Enabling WS-Security for the demo*, the *Talend ESB Infrastructure Services Configuration Guide* and the *Talend ESB STS User Guide*.

In the following sections, we describe the underlying technology and overviews of the elements.

1.3. Web Services Support

Talend ESB helps you to create new Web services or to service-enable your existing applications and interfaces for use with the Web. Developers use a declarative, policy-centric approach to enable different qualities of service through configuration, rather than code. Talend ESB leverages the features of Apache CXF for developing and deploying Web Services and REST applications.

Apache CXF provides a lightweight, modular architecture that is supported by the popular Spring Framework. So it works with your application, regardless of the platform on which it is running. It can be run as:

- a stand-alone Java application
- as part of a servlet engine (such as Tomcat)
- as an OSGi bundle on an OSGi container (such as Karaf)
- or within a JavaEE server

Apache CXF supports all important Web services standards and fully complies to the Java API for XML Web Services (JAX-WS) specification. JAX-WS defines annotations that allow you to tell a tool like CXF how your standalone Java application should be represented in a Web services context. Two types of Web services development with CXF are possible:

1. Contract-first development:

   WSDLs define what operations and types a Web service provides. This is often referred to as a Web services contract, and in order to communicate with a Web service, you must satisfy the contract. Contract-first development means that you start out by writing a WSDL file (either by hand or with the help of tooling), and then generating stub Java class implementations from the WSDL file by using a tool like CXF.

2. Code-first development:

   The other way to develop Web services is by starting out with a Java class and then letting the Web service framework handle the job of generating a WSDL contract for you. This is the easiest mode of development, but it also means that the tool (CXF in this case) is in control of what the contract will be. When you want to fine-tune your WSDL file, it may be better to go for contract-first approach.

Apache has been certified and tested to work with the broadest set of vendors' Web services implementations. Users benefit from this interoperability testing, which reduces the overall cost and complexity for application integration.

Talend ESB supports the creation of SOAP and REST Web services and offers the best WS-* functionality in the market, including support for WS- Addressing, WS-Reliable Messaging, and WS-Security over both HTTP and JMS transports.

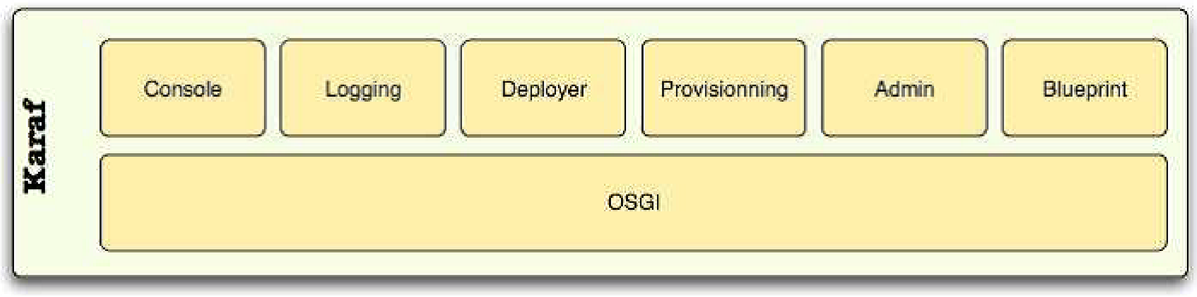
In addition, the Web services stack in Talend ESB distribution goes well beyond Apache CXF, with support for:

- OSGi containers along with illustrative examples and documentation
- Development tools, like Maven plug-ins
- WSDL document creation
- Apache Spring configuration generation
- Management and monitoring of services

1.4. Standard OSGi Runtime

The standard runtime in Talend ESB is an OSGi container. The OSGi implementation shipped with Talend ESB is Apache Karaf using Eclipse Equinox as OSGi Runtime, providing a lightweight container into which various components and applications can be deployed.

Figure 1.1. Overview of Karaf components



Karaf supports the following features:

- Hot deployment: Karaf monitors jar files inside the [home]/deploy directory. So if a jar is copied into this directory, it is automatically installed inside the runtime; subsequently this can be updated or deleted, and Karaf will act correspondingly.
- Dynamic configuration: Services are usually configured through a standard OSGi service, using property files, which are monitored; changes are propogated to the service.
- Logging: using a centralized logging back end supported by Log4J.
- Managing instances: Karaf provides simple console commands for managing multiple instances.
- See Chapter 4, *A real-world Rent-a-Car demo example* for an example of using Karaf. For further information, please see *Talend ESB Container Administration Guide* and http://karaf.apache.org/.

1.5. Messaging

Enterprise messaging software has been used for integration projects more than 30 years now. Not only is messaging a style of communication between applications, it's also a widely adopted style of integration between applications. Messaging fulfills the need for both notification and asynchronous, loosely coupled interoperation between applications.

Talend ESB embeds Apache ActiveMQ message broker to support a number of different messaging options. Although ActiveMQ is written in Java and implements the JMS specification, APIs for many languages other than Java are provided, including C/C++, .NET, Perl, PHP, etc. In addition to that many other features are added on top to provide high availability, performance, scalability, reliability, and security for enterprise messaging.

The job of the message broker is to transport events between distributed applications, guaranteeing that they reach their intended recipients. The broker therefore must be highly available, performant, and scalable for this goal.

1.6. Talend ESB products

The ESB features and functionalities provided by Talend are available in several different products:

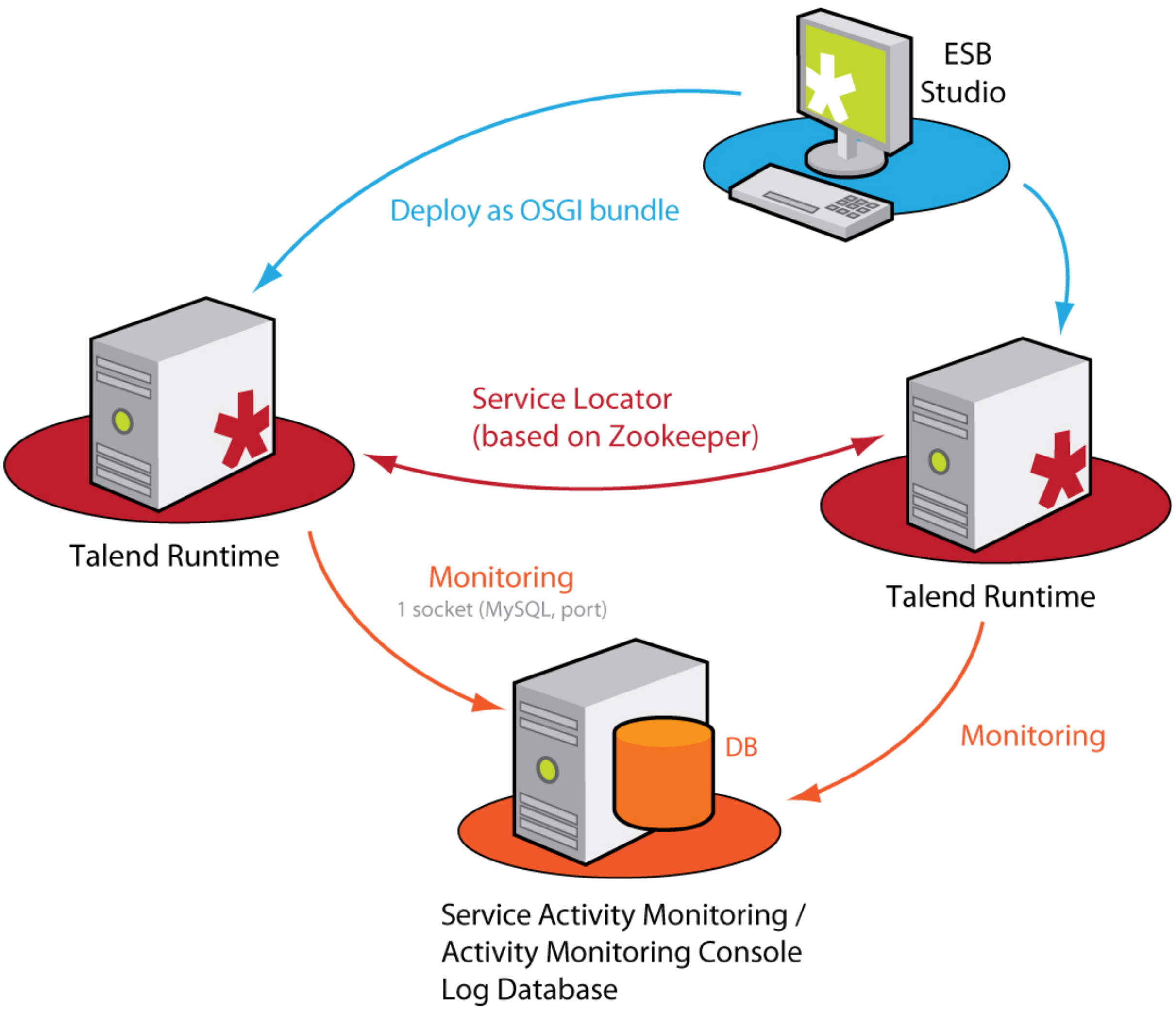
- Talend ESB Standard Edition provides a standards-based connectivity layer, Talend Runtime, used to integrate distributed systems across functional, enterprise, and geographic boundaries. Capabilities include messaging, Web services, intelligent routing, and data transformation. Its modular, pluggable architecture allows it to be easily expanded to suit most enterprise requirements, and it is available under the open source Apache license. Talend ESB is made of an OSGI container, based on Apache Karaf, into which various components and applications can be deployed, and additional ESB infrastructure services: Service Locator, Service Activity Monitoring (SAM) and Security Token Service.
- Talend Open Studio for ESB provides all of the Talend ESB Standard Edition in addition to the graphical development interface (under GPL) to develop, build, test and publish Java Web services, REST applications, data services and messaging routes.
- In addition to Talend Open Studio for ESB, Talend offers subscription products: ESB and Data Services Studio, adding advanced deployment and team collaboration facilities, as well as management functions. Team members can store and share their Data Services, Mediation Routes and metadata in a shared subversion based repository. This promotes reusability of objects and code, and facilitate the design of development best practices. Users, roles, permissions and privileges are managed centrally through the Web-based Talend Administration Center.

  Talend Administration Center also provides a graphical interface for Service Locator and Service Activity Monitoring (SAM) functionalities and gives the ability to remotely deploy the data services and mediation routes developed from the Talend Studio.

Note that while the Service Locator and Service Activity Monitoring are in both Talend Open Studio for ESB and subscription products, the Web-based user interfaces to these are part of the Web-based Talend Administration Center, and are thus only in the subscription ones. Thos product are licensed under the Talend License, and requires a license file to be installed.

We now look at each of these in more detail:

Figure 1.2. Talend Open Studio for ESB operating principles


ESB Studio
Deploy as OSGI bundle
Service Locator (based on Zookeeper)
Talend Runtime
Talend Runtime
Monitoring
1 socket (MySQL, port)
Monitoring
DB
Service Activity Monitoring / Activity Monitoring Console Log Database

Three different types of functional block are defined:

- The blue block represents a Studio API where you can carry out data integration or data service processes, mediation routes and services. For more information, see their corresponding chapters in the *Talend Open Studio for ESB User Guide*.
- The red blocks represent one or more Talend Runtimes (execution container) deployed inside your information system. Talend Runtime enables you to deploy and execute the Jobs, Routes, and Services created in the Studio. For more information on how to deploy items in Talend Runtime, see the *Talend Open Studio for ESB User Guide* and for more information about Talend Runtime itself, see the *Talend ESB Container Administration Guide*.

  If you have several Talend Runtimes on which to deploy the Services and Routes, you will be able to load balance their execution according to your needs. All instances of Talend Runtime will communicate between each other via the Service Locator to identify the one more likely to deploy and execute them.

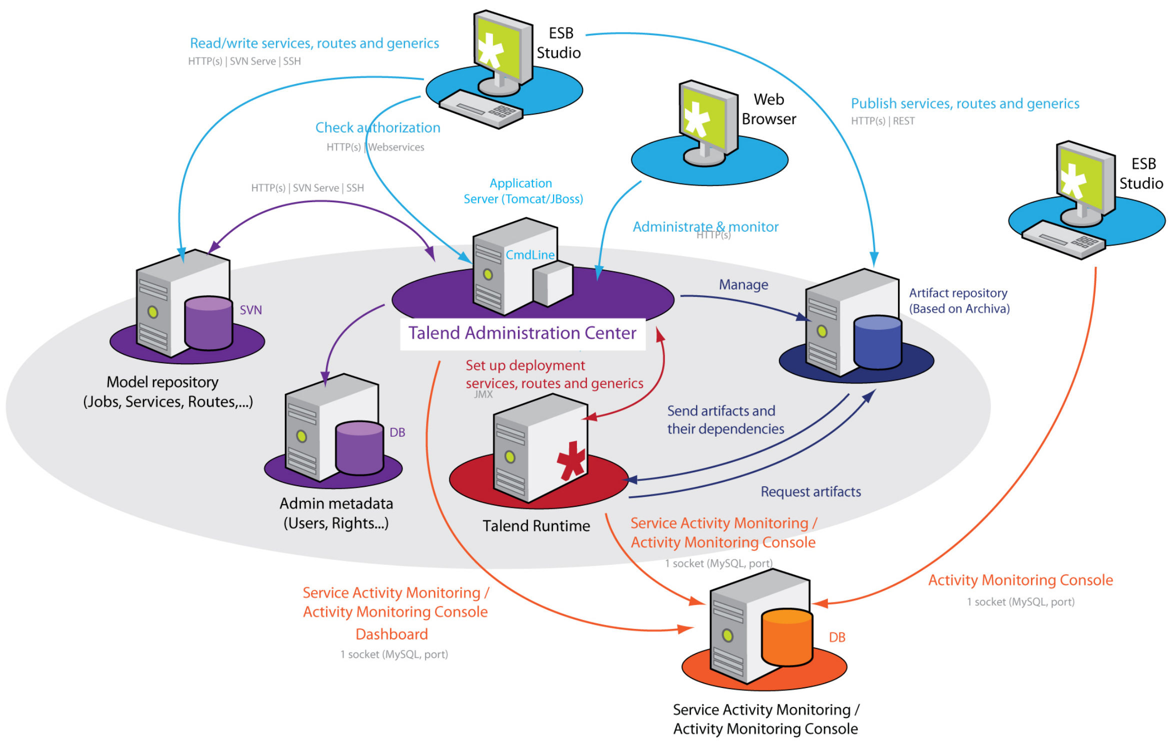
- The orange block represents a monitoring database gathering log information of the execution of your data processes and service activity.

  Data processes log information can be captured with the use of the **tFlowMeterCatcher**, **tStatCatcher**, **tLogCatcher** components. For more information, see the *Talend Open Studio Components Reference Guide*. And to automate the functionalities of the **tFlowMeterCatcher**, **tStatCatcher**, **tLogCatcher** components without using them, you can use the Stats & Logs tab. For more information regarding Stats & Logs, see the *Talend Open Studio for ESB User Guide*.

  The Service Activity Monitoring allows the end-users to monitor service calls. It provides monitoring and consolidated event information that the end-user can use to understand the underlying requests and replies that compose the event, monitor faults that may be unexpectly generated and support the system management decisions. For more information on the Service Activity Monitoring, see its corresponding chapter in *Talend ESB Infrastructure Services Configuration Guide*.

For more information on the installation of all these applications, see the Chapter 2, *Downloading and installing Talend ESB software* of the present guide or see the link to the Installation procedure on the Talend ESB download page http://www.talend.com/download/esb.

Figure 1.3. Talend Enterprise and Platform operating principles



ESB Studio
Read/write services, routes and generics
HTTP(s) | SVN Serve | SSH
Check authorization
HTTP(s) | Webservices
Web Browser
Publish services, routes and generics
HTTP(s) | REST
ESB Studio
HTTP(s) | SVN Serve | SSH
Application Server (Tomcat/JBoss)
Administrate & monitor
HTTP(s)
CmdLine
SVN
Model repository (Jobs, Services, Routes,...)
Talend Administration Center
Manage
Artifact repository (Based on Archiva)
Set up deployment services, routes and generics
JMX
DB
Admin metadata (Users, Rights...)
Send artifacts and their dependencies
Talend Runtime
Request artifacts
Service Activity Monitoring / Activity Monitoring Console
1 socket (MySQL, port)
Activity Monitoring Console
1 socket (MySQL, port)
Service Activity Monitoring / Activity Monitoring Console Dashboard
1 socket (MySQL, port)
DB
Service Activity Monitoring / Activity Monitoring Console

Five different types of functional block are defined:

- The light blue block includes one or more Studio APIs and Web browsers that could be on the same or on different machines.

  From the Studio API, you can carry out data integration or data service processes, mediation routes and services, and publish them on the Artifact Repository. Talend Studio allows the user (such as a project manager, a developer, or an administrator) to work on any project for which he has authorization to build Web, REST and data services, and mediation routes. For more information, see their corresponding chapters in the *Talend Studio User Guide*.

  From the Web browser, end-users connect to the remotely based Talend Administration Center through a secured HTTP protocol.

- The violet block includes a web-based Talend Administration Center (application server) with two shared repositories: one based on an SVN server and one based on a database server.

  The Talend Administration Center enables you to set up the execution of the tasks that handle routes or services execution into the Talend Runtime. Through the Talend Administration Center, you can access and manage Routes or Services created from the Studio and published into the Artifact Repository, and set and monitor their deployment and execution in the Talend Runtime. For more information, see the *Talend Administration Center User Guide*

- The dark blue block represents the Artifact Repository that stores all the Routes and Services that are published from the Studio and are ready to be deployed in Talend Runtime.

- The red block represents one or more Talend Runtimes (execution container) deployed inside your information system. The Talend Runtime deploys and executes the routes and services retrieved from the Artifact Repository according to the set up defined in the Talend Administration Center via the web application. For more information on how to manage deployment, see the *Talend Administration Center User Guide* and for more information about Talend Runtime itself, see the *Talend ESB Infrastructure Services Configuration Guide*.

  If you have several Talend Runtimes in which to deploy the Service and Route artifacts, you will be able to load balance their execution according to your needs. All instances of Talend Runtime will communicate between each other via the Service Locator to identify the one more likely to deploy and execute the artifact(s) set to deployment in Talend Administration Center. The Talend Runtime elected for the deployment will request for the artifact(s) to deploy and execute from the Artifact Repository and the Artifact Repository will thus send the artifact(s) requested along with all the dependencies needed for its/their execution to the Talend Runtime, Talend Runtime that will deploy and execute them.

- The orange block represents the Dashboard: the Activity Monitoring Console and the Service Activity Monitoring.

  The Activity Monitoring Console allows end-users to monitor the execution of technical processes. It provides detailed monitoring capabilities that can be used to consolidate collected log information, understand the underlying data flows interaction, prevent faults that could be unexpectedly generated and support the system management decisions.

  The Service Activity Monitoring allows the end-users to monitor service calls. It provides monitoring and consolidated event information that the end-user can use to understand the underlying requests and replies that compose the event, monitor faults that may be unexpectly generated and support the system management decisions.

For more information on the installation of all these applications, see the *Talend Installation Guide*.

1.7. Studio

Talend Open Studio for ESB and its subcription equivalents provide a graphical development tool with:

- an Integration perspective
- a Mediation perspective
- a Java perspective (ESB and Data Services Studios including m2eclipse Plugin)
- a soapUI perspective (ESB and Data Services Studios only)

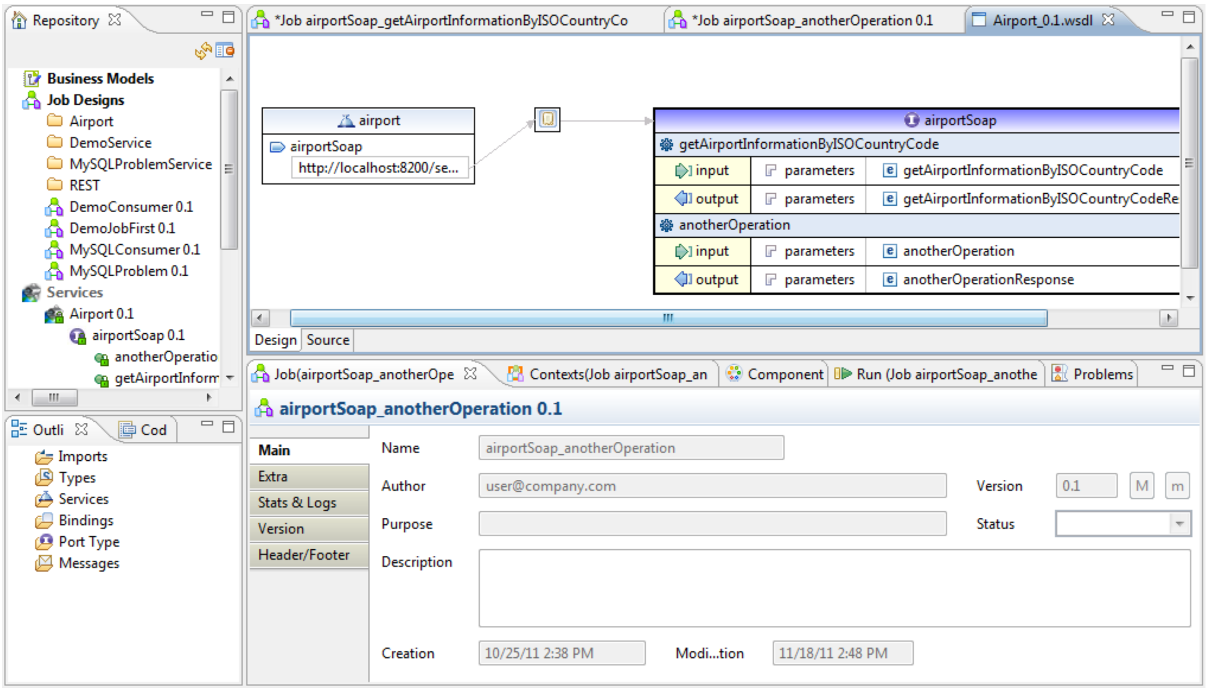
These are discussed in more detail in the rest of this section. We will use the term "Studio" for both versions unless there is a specific difference involved.

1.7.1. Integration perspective

The Integration perspective is a graphical tool within the Studio which allows you to use the extensive list of data adapters and components to build ESB data services and export them for standalone or for deployment in a local Talend Runtime container. The ESB or Data Services Studio also enables to publish services in a provisioning repository in Talend Artifact Repository for remote deployment from Talend Administration Center. The Integration Perspective provides data services development through an easy-to-use graphical development environment. It enables rapid deployment and reduces maintenance costs with prebuilt connectors to all source and target systems, with support for all types of data services, data migration and data synchronization operations.

The core of the Integration perspective comprises four major applications (Business Modeler, Job Designer, Service Designer, and Metadata Manager) within a single graphical development environment based on Eclipse.

Figure 1.4. Talend ESB Integration perspective with a Service design

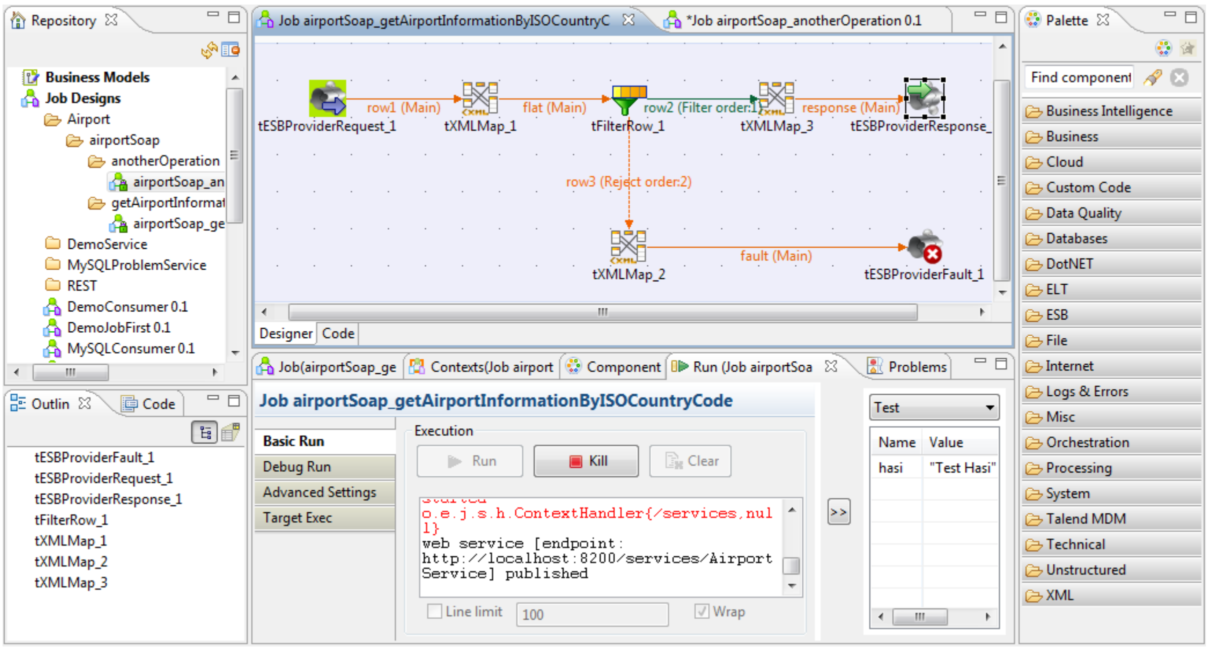


airportSoap_anotherOperation 0.1

Name	airportSoap_anotherOperation
Author	user@company.com
Version	0.1
Purpose	
Status	
Description	
Creation	10/25/11 2:38 PM
Modi...tion	11/18/11 2:48 PM

A Service in the 'Services' node is a Web-Service defined by a WSDL. The WSDL can be just imported, created from scratch in the tooling using the embedded graphical WSDL editor or an existing WSDL can be imported and then edited within the studio. In this case the Service is based on this WSDL information and each service operation can then be implemented in the Job Design node.

Figure 1.5. Talend ESB Integration perspective with a Job design



```
started
o.e.j.s.h.ContextHandler{/services,nul
l}
web service [endpoint:
http://localhost:8200/services/Airport
Service] published
```

Name	Value
hasi	"Test Hasi"

A data service Job is a graphical design, of one or more components connected together, that allows you to set up and run data service operations. Jobs address all of the different sources and targets that you need for data integration processes and combine it with Web services.

Additionally, in the ESB or Data Services Studio, you can use the shared repository feature to work in larger teams, and share resources. It has the facility of team collaboration - team members can store and share their business models, integration and service jobs, integration services and metadata in an industry-standard source manager (SVN). This promotes reusability of objects and code, as well as facilitating the design of development best practices. There are also more extended productivity features like Wizards (for example, for SAP) and deployment options (Publish to Talend Artifact Repository) available in those Studios.

More information on this can be found in *Talend Open Studio for ESB User Guide*, *Talend Data Services Studio User Guide* or *Talend ESB Studio User Guide*, depending on your product or subscription; see Section 1.8, “*List of resources*” for a complete list of manuals.

1.7.2. Mediation perspective

First, let's discuss Apache Camel, then we'll look at the Mediation perspective which is a graphical interface to this functionality.

Apache Camel

The Mediation functionality of Talend ESB is based on the popular Apache Camel project. The core of the Camel framework is a routing engine. It allows you to define routing rules, that accept and send messages through Components. There are no restrictions on the message format - for example, Java objects, XML, JSON, plain text and so on, can be used. These routing rules are based on the book “Enterprise Integration Patterns” from Gregor Hohpe (et al).

Thus, Apache Camel is a framework allowing developers to assemble Endpoints / Processors into workflows (Routes) to achieve higher level functionality. It facilitates application integration by leveraging Enterprise Integration Patterns (EIPs) to essentially assemble scalable services, and make message-based system integration simpler to design and implement. Camel is an open source Java framework that lets you create EIPs to implement routing and mediation rules using a Java-based Domain Specific Language (DSL). This means you get completion of routing rules that can easily and rapidly be updated if requirements change.

EIP and hence Camel are focused on asynchronous messaging patterns because asynchronous patterns are necessary for scalability. But asynchronous messaging is more complex. Camel wraps that complexity in a common framework which is widely understood by integration developers.

Because of the high-level abstractions in Camel (for example, Message, Exchange, Component, Endpoint), you can use the same APIs to interact with various systems. For example, every route implements Component and Endpoint APIs and works with the same basic Message and Exchange structure. Each component is implemented to work with specific protocols and data types. So you can wire together very different protocols and system very easily while using the same patterns and routing semantics.

Camel mediates both formats and transports between different endpoints. Typically, this allows Services to communicate with each other, via Camel mediation, even if using different message formats, and written using different languages. It also makes adding functionality like logging and tracking easy. Camel is also very lightweight and can be embedded or deployed anywhere you like, such as in a standalone Java application, web application, Java EE application or an OSGi bundle.

You can use the Java Domain Specific Language (DSL) to specify a route. For example:

```
From(“file:directory”).to(“jms:queuename”)
```

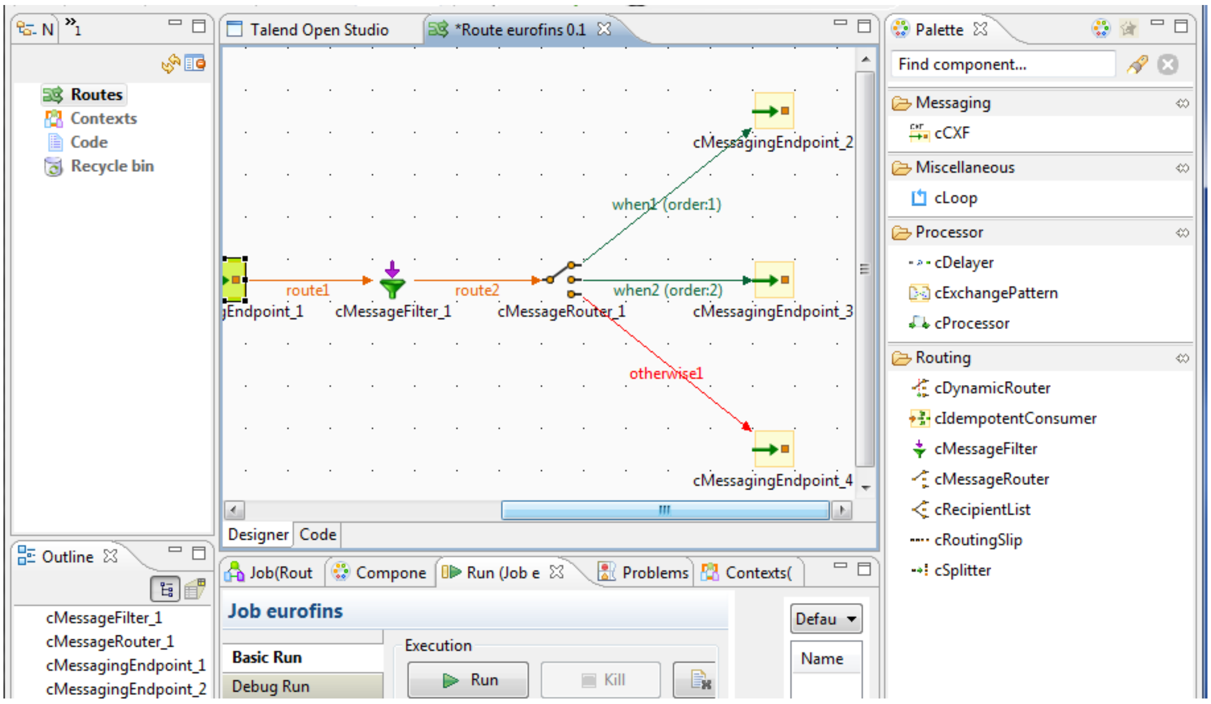
This simple statement polls files from a directory and sends the content of each file a JMS message to a JMS queue. This can be done because Camel provides a uniform interface based on the Exchange object.

Talend ESB Mediation perspective

On top of Apache Camel, and integrated with the Studio, the Route Builder (Mediation perspective) is a GUI that allows a developer to build these Routes in a visual way.

You can run the Routes in a standalone mode or export them as OSGi Bundles which can be easily deployed within the Talend Runtime container. Additionally in Talend Studio you can use the shared repository feature to work in larger teams.

Figure 1.6. Talend ESB Mediation perspective



Out of the box, Talend ESB supports over 80 protocols and data types, through its extensive library of components. These components enable a Route to connect over transports, use APIs, and understand data formats. You can also create your own components very easily; the component API is very concise. Talend ESB has a modular architecture, which allows any component to be loaded into Camel, regardless of whether the component ships with Camel, is from a third party, or is your own custom creation. The creation of your own components (for example, for processing, connectivity, routing, and so on) is simple and well documented.

1.8. List of resources

This table contains a summary of further resources to look at:

1.8.1. Apache-related component resources

Description	**Related Apache component**	**Related manual(s)**
Services enablement using CXF	CXF: http://cxf.apache.org/	
Mediation and integration components	Camel: http://camel.apache.org/	
Talend ESB message broker	ActiveMQ http://activemq.apache.org/	*Talend ESB Getting Started Guide*
Security Token Service	WSS4J (WS-Security) http://ws.apache.org/wss4j	*Talend ESB Getting Started Guide*, *Talend ESB Infrastructure Services Configuration Guide*, *Talend ESB STS User Guide*
Service Locator	ZooKeeper http://zookeeper.apache.org/	*Talend ESB Getting Started Guide*, *Talend ESB Infrastructure Services Configuration Guide*
Talend Runtime container	Karaf (OSGi) http://karaf.apache.org/	*Talend ESB Getting Started Guide*, *Talend ESB Container Administration Guide*, *Talend ESB Development Guide*

1.8.2. List of Talend ESB related manuals

1.8.2.1. Talend ESB standard manuals

These manuals are for the parts of Talend ESB not related to Studio.

- *Talend ESB Getting Started Guide* - an overview of Talend ESB and how to run the demo software
- *Talend ESB Development Guide* - recommended practices in developing software using Talend ESB
- *Talend ESB Container Administration Guide* - information on Karaf tooling commands and administration
- *Talend ESB System Management Integration User Guide* - using the Hyperic HQ plugins
- *Talend ESB Infrastructure Services Configuration Guide* - configuring the Service Locator, Service Activity Monitoring and Security Token Service
- *Talend ESB STS User Guide* - detailed description of configuring and using Security Token Service

- *Talend ESB Mediation Development Guide* - describes Camel mediation components
- *Talend ESB Service Developer Guide* - describes service enablement using CXF

1.8.2.2. Studio manuals

- *Talend Open Studio for ESB User Guide*, *Talend Data Services Studio User Guide* or *Talend ESB Studio User Guide,* depending on your product or subscription.
- *Talend Components Reference Guide* or *Talend Open Studio Components Reference Guide* for the individual components listed in the user interface of Studio.
- *Talend ESB Mediation Components Reference Guide* or *Talend Open Studio for ESB Mediation Components Reference Guide*, for the components in the Mediation perspective of Studio.
- In addition, Talend Enterprise editions have an addition manual *Talend Administration Center User Guide* for the Enterprise Talend Administration Center component.

Chapter 2. Downloading and installing Talend ESB software

This chapter describes how to download and install Talend ESB software. By the end of this chapter, you should be able to run the Studio and Talend Runtime software and also the ready to run related examples.

2.1. Prerequisites to using Talend ESB

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 ESB Studio, Talend Data Services Studio or Talend ESB, see the corresponding *Talend Installation Guide*.
- If you are using Talend Open Studio for ESB or Talend ESB Standard Edition, see the link to the Installation procedure on the Talend ESB download page (http://www.talend.com/download/esb).
- Some of the ESB components use Apache software components (for example, Apache CXF, Apache Camel). For details on the exact software versions involved, see the section on Apache software in the product release notes.

2.2. Downloading, extracting the software

This section describes downloading and extracting Talend ESB software.

- The Community Edition of Talend software is freely available on the website http://www.talend.com/download.php.

- Customers who subscribe to the Enterprise Edition of Talend software are sent an email with download instructions and a license file.

Download the relevant file(s) and when the download is complete, extract the archive files on your hard drive.

If you initially get errors about directory names being too long when extracting the archive file, please extract again, perhaps using software such as 7-Zip, to replace any missing directories. Otherwise, you may later get the error "JET initialisation Time Out" when running a job.

When extracted to a directory of your choice, these are the parts involved:

For **Talend Open Studio for ESB**, both Studio and Talend Runtime are bundled together.

- There is one compressed file, of the format: `TOS_ESB-rYYYYY-V5.2.x.zip`. When you extract it you get two folders: `Studio` and `Runtime_ESBSE`.
  - Studio is at the root level of the `Studio` directory.
  - The `Runtime_ESBSE` directory contains Talend Runtime and examples.

For the **ESB** or **Data Services Studio**, there are a number of compressed files, as each of these is often used in different combinations with other packages:

- A file of the format `Talend-Studio-rYYYYY-V5.2.x.zip`: Studio is at the root level of the extracted directory.
- `Talend-ESB-V5.2.x.zip`: the extracted directory contains Talend Runtime and examples.
- `Talend-Runtime-V5.2.x`: this is a standalone package for just the Talend Runtime. It does not contain examples, and corresponds to the `container` subdirectory of `Talend-ESB-V5.2.x`. Note: For the purposes of most of this document, which discusses the examples in detail, it is assumed that this standalone package is not being used.

We use the term `<TalendRuntimePath>` for the directory where Talend Runtime is installed. This is the full path of either `Runtime_ESBSE` or `Talend-ESB-V5.2.x`, depending on the version of the software that is being used. Please substitute appropriately.

Directory paths mentioned in the Rent-a-Car demo chapters will be relative to this; examples are in the `<TalendRuntimePath>/examples/talend` directory.

2.3. Installing Studio

This section describes installing Talend Open Studio for ESB and also covers the ESB and Data Services Studio, as the installation process is similar. The installation process is based on Eclipse RCP. In the section we will refer to Talend Open Studio for ESB, ESB and Data Services Studio as 'Studio' unless there is a specific difference.

In this section we use names with the prefix *Talend-** for the files, but these may have a different prefix depending on the edition, for example *TOS_ESB-** in Talend Open Studio for ESB.

If you are only interested in the Talend Runtime, you can skip this section and continue to Section 2.4, *“Installing Talend Runtime - quick start”*

2.3.1. Starting Studio

1. Download the Talend software as described in Section 2.1, *“Prerequisites to using Talend ESB”* and Section 2.2, *“Downloading, extracting the software”*

2. Next is optional configuration: If you want to tune the memory allocation for your JVM, you only need to edit the .ini file corresponding to your executable file. For example:

   - For Studio on Windows, edit the file: `Talend-win32-x86.ini`.
   - For Studio on Linux, edit the file: `Talend-linux-gtk-x86.ini`..

   The default values are:

   ```
   -vmargs -Xms40m -Xmx500m -XX:MaxPermSize=128m
   ```

3. Then launch Studio:

   - On Windows, double-click the appropriate executable file to launch Studio (for example `Talend-win32-x86.exe` for a 32 bit system).
   - On Linux, add execution rights on the desired “Talend-*” binary before launching it.

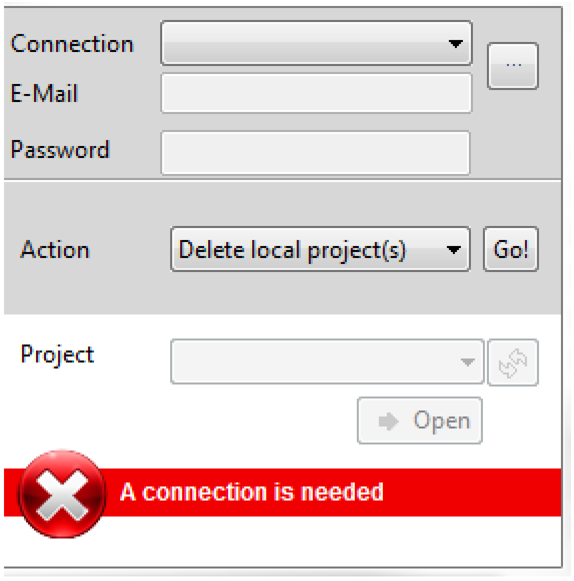
   ```
   $ chmod +x Talend-linux-gtk-x86
   $ ./Talend-linux-gtk-x86
   ```

4. In the case of ESB or Data Services Studio, you will be asked to load a license file at this point. Browse to where your license file is located, and click ok to load it.

5. You are then asked to accept the Talend license agreement. If you agree, please click ok.

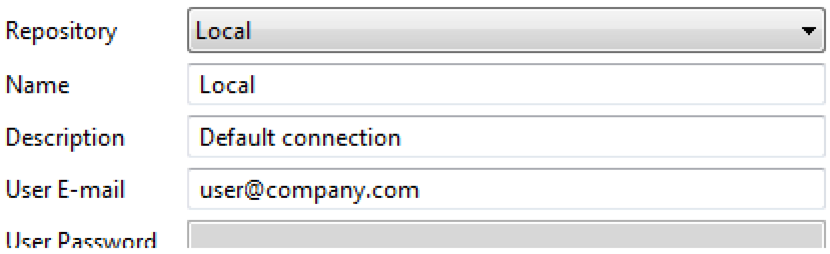
2.3.2. Login screen and registration

1. You need to create a connection and register your details before logging to Studio. The first screen is a login screen which indicates that a connection is needed.



A connection is needed

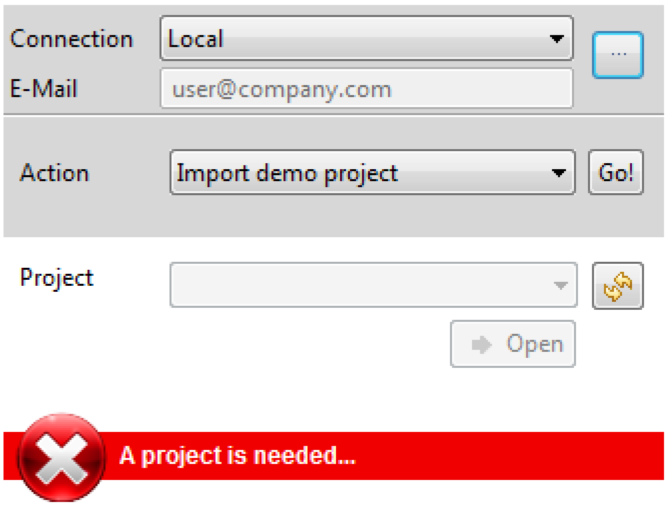
   Click the […] button next to the connection field.

2. 

Repository: Local
Name: Local
Description: Default connection
User E-mail: user@company.com
User Password

In the Email field, type in a valid email address. Also specify the path of your workspace, or accept the default. Note that here we assume that you are using a local environment. In ESB or Data Services Studio, you can collaborate with other users with a shared environment, and this is described in the *Talend ESB Studio User Guide* or *Talend Data Services Studio User Guide*, depending on your product or subscription.

3. Click OK to validate and go back to the login window. In the login window, the connection information you just gave displays in the Connection area.



Connection: Local
E-Mail: user@company.com
Action: Import demo project
Project:
A project is needed...

You are now ready to start using Studio. As first time user, you need to set up a new project - as an example, we set up a very simple "SayHello" example in Chapter 3, *A simple SayHello example for Studio* . You can also import the Demo project which contains job samples (these are discussed in *Talend Open Studio for ESB User Guide*).

2.4. Installing Talend Runtime - quick start

This section is a quick start overview of installing the Talend Runtime of both Talend ESB Standard Edition and Talend ESB. In the section we will refer to both of these as 'Talend Runtime' unless there is a specific difference. The section additionally describes starting the Talend Runtime infrastructure services.

We strongly encourage you to read Section 2.1, “*Prerequisites to using Talend ESB*” before starting this section, and then download the Talend Runtime as described in Section 2.2, “*Downloading, extracting the software*”.

This package includes:

- Talend Runtime container

and the infrastructure services:

- Service Locator
- Service Activity Monitoring
- Security Token Service

and also:

- the message broker Apache ActiveMQ

2.4.1. Talend Runtime container - quick start

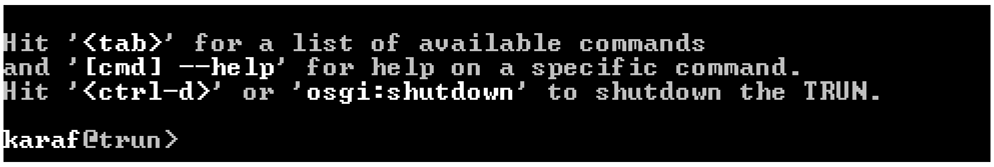
Talend Runtime container is a ready to be used OSGi container allowing you to deploy all your features.

To run Talend Runtime container:

1. Go to subdirectory `<TalendRuntimePath>\container\bin` of Talend Runtime installation directory.
2. Run the `trun.bat` or `trun.sh` file.

When the container starts up, you will see a short introduction (similar to the one below) followed by the Talend Runtime container console command prompt:

Figure 2.1. Talend Runtime container startup



```
Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown the TRUN.

karaf@trun>
```

For further information, see Section 2.6, *"Starting a Talend Runtime container"*.

2.4.2. Start-all

Instead of these three individual start commands described below, you can also use: **`tesb:start-all`** in the container console window, which starts the Service Locator, Service Activity Monitoring server and the Security Token Service.

2.4.3. Service Locator

Service Locator is a ready-to-be-used failover and load balancer tool (based on Apache Zookeeper) allowing you to dynamically register your endpoints.

To run Service Locator inside the Talend Runtime, just type in the container console window:

```
tesb:start-locator
```

To run Service Locator independently:

1. Go to subdirectory `<TalendRuntimePath>\zookeeper\bin` of Talend Runtime installation directory.
2. On Linux: Run the **`zkServer.sh start`** command.

   On Windows: Run the **`zkServer.cmd start`** command.

Under Linux, ensure execution rights for the locator startup scripts:

```
chmod a+x zookeeper/bin/*.sh
```

For further information and examples, see Chapter 6, *Enabling the Service Locator and Service Activity Monitoring for the demo*.

2.4.4. Service Activity Monitoring

Service Activity Monitoring is a monitoring tool facilitating the capture of analysis of service activity.

The Service Activity Monitoring Server can be installed into the Servlet Container (such as Apache Tomcat) or Talend Runtime container. It supports Apache Derby, MySQL, Oracle, SQL Server, IBM DB2 and H2 Database Engine to store Events data.

To install the Service Activity Monitoring server within the Talend Runtime container please type:

`tesb:start-sam`

in the console. This will also automatically start an Apache Derby database. Note: the Service Activity Monitoring within the Talend Runtime can only be used with Apache Derby - if you would like use one of the other supported databases, use the Tomcat deployment of the Service Activity Monitoring server.

For further information and examples, see Chapter 6, *Enabling the Service Locator and Service Activity Monitoring for the demo*.

2.4.5. Security Token Service

Security Token Service is a framework allowing clients and services to securely and transparently authenticate during connections.

To install the Security Token Service Server within the Talend Runtime container please type:

`tesb:start-sts`

in the console.

For further information and examples, see Chapter 7, *Enabling WS-Security for the demo*.

Note: if you are using Talend ESB, then a GUI interface for these services is available via the Talend Administration Center. In addition to the information in this guide, please see *Talend Enterprise ESB Installation Guide* and *Talend Administration Center User Guide* for more details.

2.4.6. Apache ActiveMQ

Apache ActiveMQ implements the Java Message Service (JMS 1.1). To start the standalone Apache ActiveMQ broker, in a normal command window, **cd** to `<TalendRuntimePath>/activemq/bin`. Then enter:

`activemq console` (Linux*)
`activemq` (Windows)

The Apache ActiveMQ broker should now be running.

*Note the "console" option in Linux runs the broker in the foreground; the default is to run it in the background.

2.5. Software prerequisites for the Demos

In addition to the prerequisites in Section 2.1, *“Prerequisites to using Talend ESB”*, there is one extra piece of software needed to run the demos:.

Apache Maven 3.0.3 is used to build the "export the service" part of SayHello example (see Section 3.5, *“Exporting the service and running it in an Talend Runtime container”*) and all of the Rent-a-car demos. So, Apache Maven should be downloaded from http://maven.apache.org/ and installed, and the **mvn** executable should be in your `PATH`.

When running **mvn** initially, HTTP access to the Internet is required, in order to download the required artifacts (and subsequently to get the most recent versions). The local Maven repository is expected to be created in its default location, that is, the standard Maven configuration should not be modified.

2.6. Starting a Talend Runtime container

Talend Runtime containers are OSGi containers which are used by the "Export Job" part of the Studio SayHello example, and also by the Rent-a-Car demos in Talend ESB.

An OSGi container is included in the installation in the subdirectory `container` in the Talend ESB installation.

When you start up the Talend Runtime container, you should allow more than 20 seconds for it to initialize and configure itself properly before entering any command at the prompt. Even if the command prompt is visible, the job commands may not be available until after this initialization time.

In addition, the container should not be shut down during this phase. Otherwise, when it is started up again, it may not activate all components properly. See also Re-initializing the container later in this section.

Start the Talend Runtime container running:

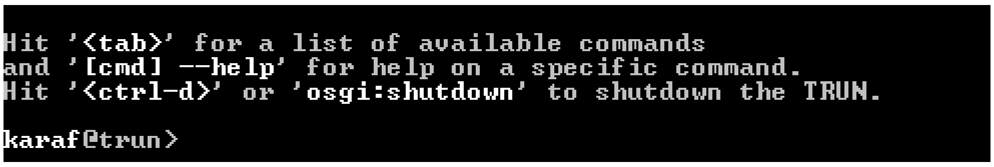
Under Linux, go to directory `<TalendRuntimePath>/container` and execute the following command:

```
./bin/trun
```

Under Windows, go to directory `<TalendRuntimePath>\container` and execute the following command:

```
.\bin\trun.bat
```

When the container starts up, you will see a short introduction (similar to the one below) followed by the OSGi console command prompt:


```
Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown the TRUN.

karaf@trun>
```

The OSGi commands explained in the following sections will be entered at this prompt.

In order to check that the container has started up and activated all components, enter the command:

```
list
```

You should get a listing like the following fragment (note that the details of the list can vary, depending on what is installed):

```
karaf@trun> list
START LEVEL 100 , List Threshold: 50
```

```
   ID   State      Blueprint    Spring    Level Name
[  49] [Active ] [           ] [       ] [ 60] Activation 1.1 (1.1)
[  50] [Active ] [           ] [       ] [ 60] geronimo-servlet_2.5_spec (1.1
.2)
[  51] [Active ] [           ] [       ] [ 60] JavaMail API (compat) (1.4.4)
[  52] [Active ] [           ] [       ] [ 60] geronimo-jta_1.1_spec (1.1.1)
[  53] [Active ] [           ] [       ] [ 60] Jetty :: Utilities (7.4.5.v201
10725)
[  54] [Active ] [           ] [       ] [ 60] Jetty :: IO Utility (7.4.5.v20
110725)
....
```

Make sure all the listed components show state "Active" before you initiate any further action. If any component remains in state "Resolved", after waiting for 30-60 seconds, you can manually activate it entering "**`start <ID>`**" where "<ID>" refers to the number in the first column, for example, "**`start 100`**" activates the component with ID "100".

Re-initializing the container

An OSGi container preserves state between runs. This is stored in `data` subdirectory in the container - it is created on the first run. If you run into problems, and want to make a fresh start, just shutdown the container (using **`<ctrl-d>`** or **`osgi:shutdown`** on the OSGi command prompt), delete the relevant container's `data` subdirectory and restart the container.

You can find further information about the OSGi container used by Talend ESB (Talend Runtime container) and how to get started it at *Talend ESB Container Administration Guide* and http://karaf.apache.org/.

2.7. Creating an alternate container

One OSGi container is included in the installation in the subdirectory `container` in the Talend ESB installation. However, to run the demos, we recommend having two OSGi containers, and this section describes how to create the second one on the same machine.

The second container matches a common real world situation of client and server running on different containers. This second container is also needed in the SayHelloRoute part of the SayHello example Section 3.6, *“SayHelloRoute example”*.

A second instance is a copy of the Talend Runtime container (or Karaf) that you can launch separately and deploy applications into.

Please make sure the first container is stopped (using the shutdown command or CTRL-D), and that its `data` directory has been deleted, before copying it (see below). Otherwise the data in the first container (which contains absolute paths relating to the first container) will cause problems in the alternate container.

Creating the alternate container can be achieved in the following way:

1. In a Linux terminal or Windows command line window, change directory to `<TalendRuntimePath>`.

2. Create a copy of directory `container` with all its content by executing the most applicable command:

   `cp -r container alternate-container` (Linux)

   `robocopy /e container alternate-container` (Windows)

   `xcopy /e container alternate-container` (older Windows with no robocopy)

   Now, you have a second OSGi container in directory `alternate-container`.

3. In order to avoid conflicts between the two container instances, the configuration of the alternate-container needs to be adjusted. We do this by running the alternate-container with default settings, and then running a script at the prompt containing Karaf commands to update and save the new settings - the container thus updates its own settings.

   So make sure the default (first) container is not running at the same time, or there will be a port conflict.

   Start the alternate OSGi container:

   Under Linux, go to directory `<TalendRuntimePath>/alternate-container` and execute the following command:

   ```
   ./bin/trun
   ```

   Under Windows, go to directory `<TalendRuntimePath>\alternate-container` and execute the following command:

   ```
   .\bin\trun.bat
   ```

4. Wait until the initialisation phase has finished, then execute the Karaf configuration adaption script. Input this command at the alternate-container console:

   ```
   source scripts/configureC1.sh
   ```

   The settings are now updated and saved locally. The next time you run the alternate container, the new settings will be used, and both container and alternate-container can be running at the same time.

5. It is not required for the following demos, but if you would like to create a third or fourth container, copy the original container directory to a third location (see above). Again, start running the third container and use this call at the Karaf prompt to set another unique group of settings:

   ```
   source scripts/configureC2.sh
   ```

   Similarly, a fourth container should use the call:

   ```
   source scripts/configureC3.sh
   ```

   You can reconfigure the default values in any container using:

   ```
   source scripts/configureC0.sh
   ```

If you get the "Port already in use" exception when starting an alternate container, recheck that there is not already a container running using the default parameters. If you still get the error, it may also be that the port is in use by an unrelated process, so update the configuration files in the `alternate-container/etc` folder by hand accordingly.

Chapter 3. A simple SayHello example for Studio

This chapter will help you to get up and running with Talend Open Studio for ESB, Talend Data Services Studio or Talend ESB Studio, depending on your product or subscription, which we will call 'Studio' in the following text. We will build a simple SayHello example, using pre-defined templates.

If you are only interested in the Talend Runtime, you can skip this chapter and continue to Chapter 4, *A real-world Rent-a-Car demo example*

3.1. Overview

Here are the steps involved in the SayHello example:

1. First, we will install Studio (Section 2.3, "*Installing Studio*").
2. Then, we will build a simple "SayHello" data service, in which a consumer sends a number of names to a service, which then prints "Hello!" to each of them in turn.
3. Finally, we will build a simple "SayHello" route

This chapter gives enough information to create and run the demo. For a comprehensive look at the Studio UI, please see *Talend Open Studio for ESB User Guide*, *Talend Data Services Studio User Guide* or *Talend ESB Studio User Guide*, depending on your product or subscription.

For more details on specific components mentioned in this demo, please see *Talend Components Reference Guide* or *Talend Open Studio Components Reference Guide*.

3.2. Building a simple "SayHello" data service

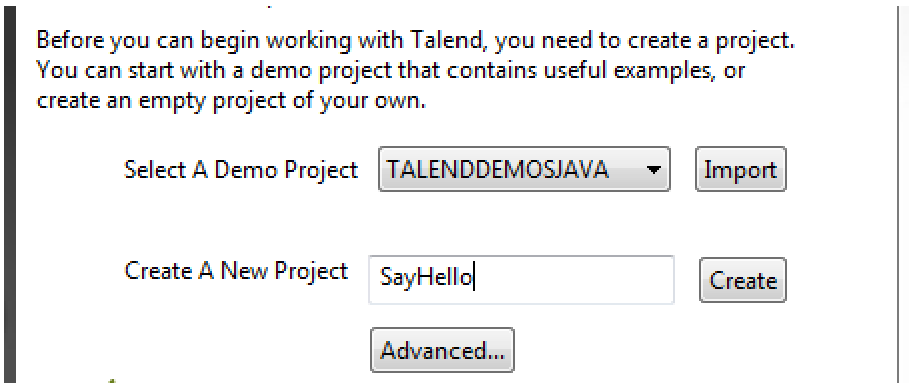
There are a number of parts to creating this, implemented by dragging and dropping existing functionality.

1. Create a "SayHello" provider
2. Create a "SayHello" consumer
3. Export the service to a Talend Runtime container, and run the consumer again

3.3. Creating a "SayHello" provider

This chapter provides you a step by step instruction to build the "SayHello" example on your own. However, if you would just like to see other Data Services and Routes right away, or if you would like to continue after you completed the SayHello examples, we would recommend that you import the ESBDEMO Project which you can select from the **Select a Demo Project** combo box.

1. In the Studio login screen, select **Create a new local project** and click **Create**.

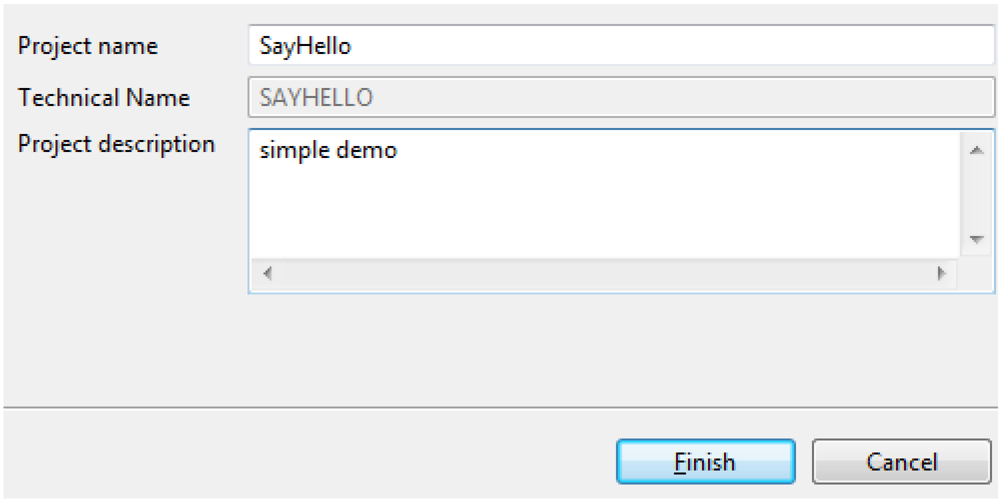


Before you can begin working with Talend, you need to create a project.
You can start with a demo project that contains useful examples, or
create an empty project of your own.

Select A Demo Project: TALENDDEMOSJAVA

Create A New Project: SayHello

A screen opens for the project details. Give the **Project name** **`SayHello`** and **Project description** **`simple demo`**.



Project name	SayHello
Technical Name	SAYHELLO
Project description	simple demo

Click **Finish**. This action returns us to login screen. Then select the project from the project drop down menu, and click **Open**.

2. If you haven't registered previously, then please enter your details to create an account to access TalendForge (tutorials, support, and so on), or else click **Skip** to skip this step for the moment. If you have registered and logged in previously, this screen does not appear.

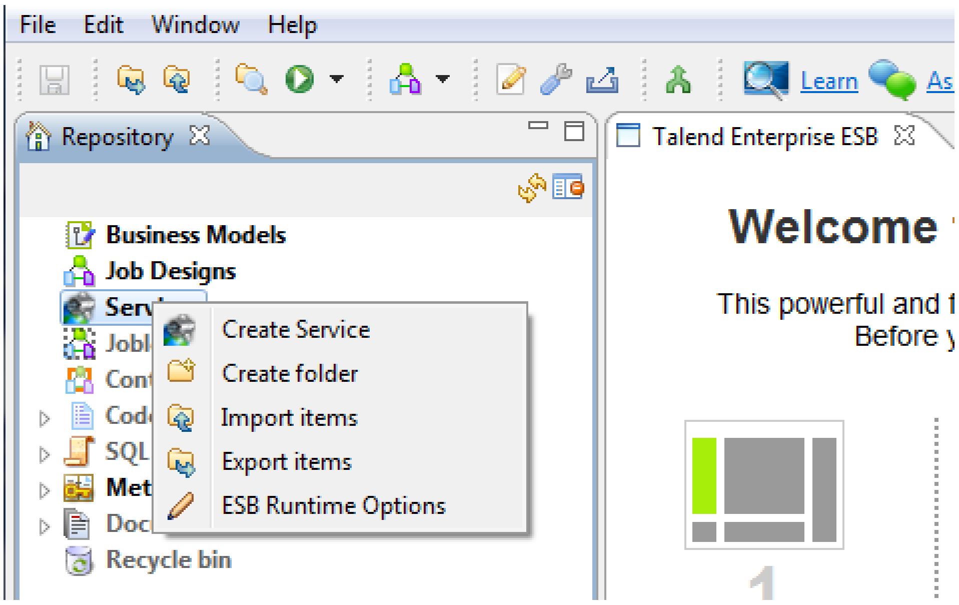
3. Now, the components are loaded, which takes a few seconds. This is followed by a message about "Generation Engine Initialization in progress" (you can optionally run this in the background, and tick the box "always run this in the background" for future use).

4. The Welcome screen in Studio is now displayed. For the purposes of this demo, click "X" to the right of Welcome, to exit to the main Studio screen.

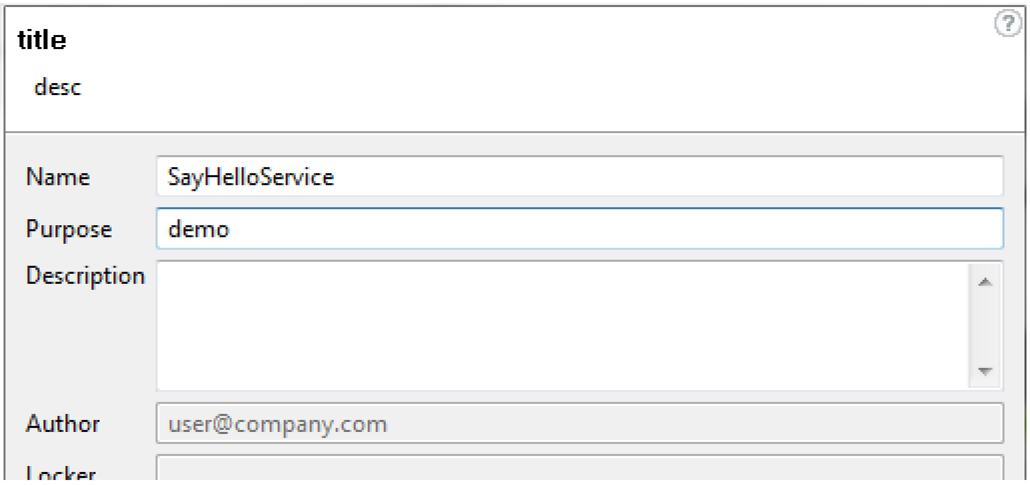
3.3.1. Creating a service

Note that in this section, we create a WSDL which defines the external contract to the Web service that clients can use. (Please see Section 1.3, "*Web Services Support*" for more on this).

1. Now, we start to create the service. Right-click **Services** in the left hand menu and select **Create Service**.



2. Enter the name (`SayHelloService`) and purpose (`Demo`) of the service, and click **Next**.

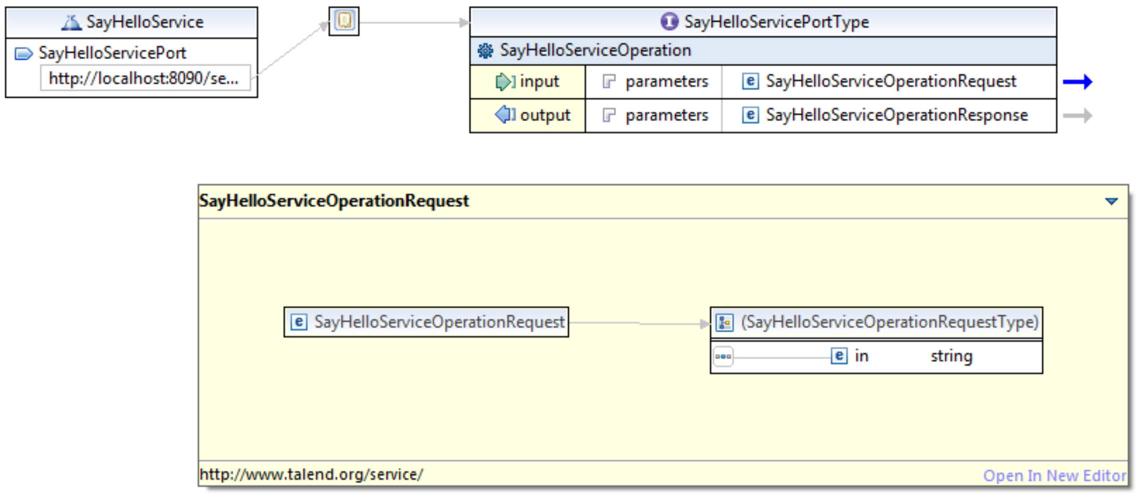

title

desc

Name	SayHelloService
Purpose	demo
Description	
Author	user@company.com
Locker	

3. The next screen is the **Assign WSDL** screen. Select **Create new WSDL** and click **Finish** to return to the main screen.

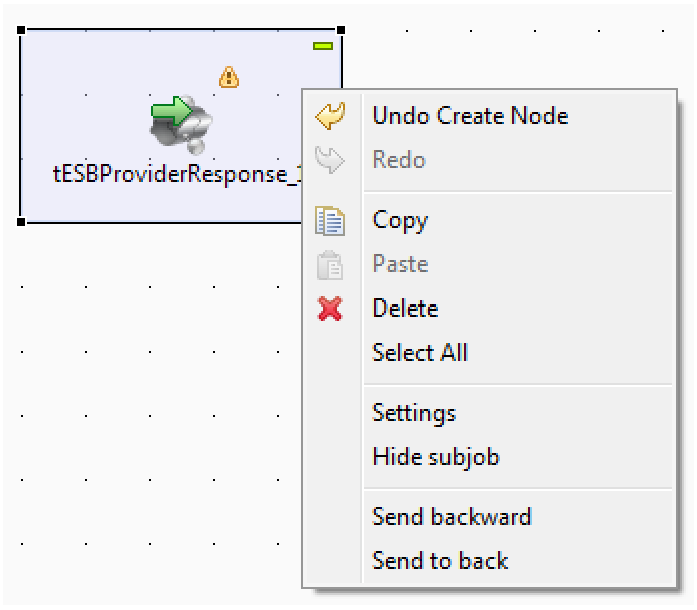
4. Now the main screen has a **SayHelloService_0.1.wsdl** tab displayed. This WSDL now contains a new port (**SayHelloServicePort**) for the service, and default request and response operations (**SayHelloServiceOperationRequest** and **SayHelloServiceOperationResponse** respectively). Hover over the grey arrows to the right of the operations to display their parameters:


SayHelloService
SayHelloServicePort
http://localhost:8090/se...
SayHelloServicePortType
SayHelloServiceOperation
input
parameters
SayHelloServiceOperationRequest
output
parameters
SayHelloServiceOperationResponse
SayHelloServiceOperationRequest
SayHelloServiceOperationRequest
(SayHelloServiceOperationRequestType)
in
string
http://www.talend.org/service/
Open In New Editor

It is possible to make changes to the operations - add new operations and edit existing ones. However, the default operations are enough for this demo example.

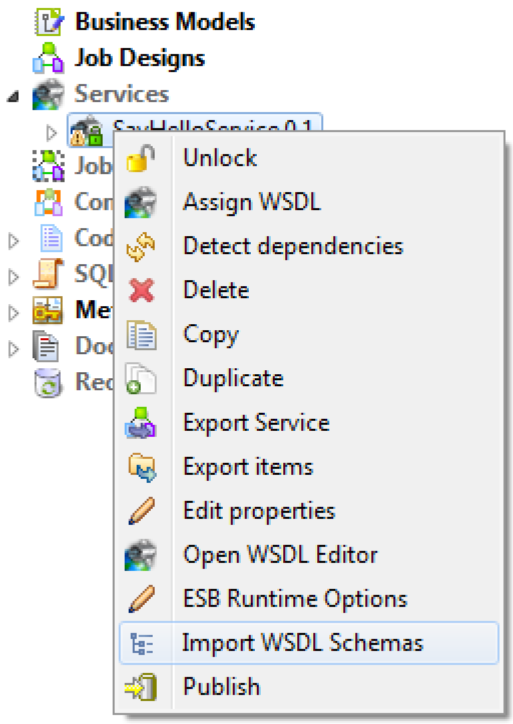
3.3.2. Creating the implementation and making it available

To modify the grid components in the following sections:

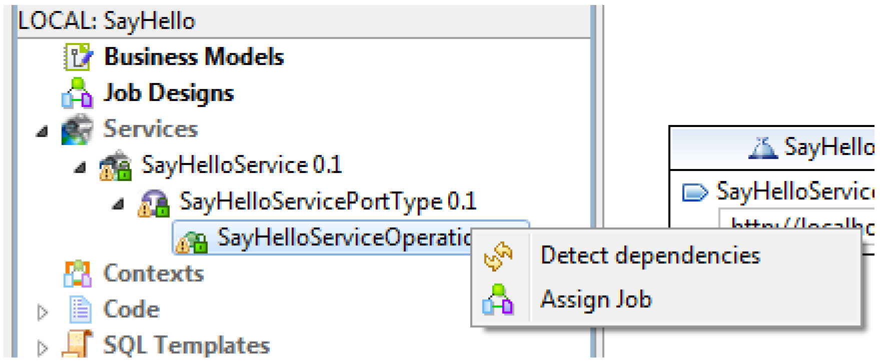


- Click on the border of a square to move the shape itself around the grid.
- Undo or edit the changes these changes by right-clicking the square and selecting the appropriate actions.
- Click the center of a square for actions particular to the component (for example, examining details of the component).

1. First, we make the implementation details, and the WSDL Request / Response data types accessible to other components. So, right-click **Services** > **SayHelloService 0.1** and select **Import WSDL Schemas**. This exports the WSDL metadata from the service and imports it into our repository (File XML). This allows us to share these implementation details of the operations across services and other components.

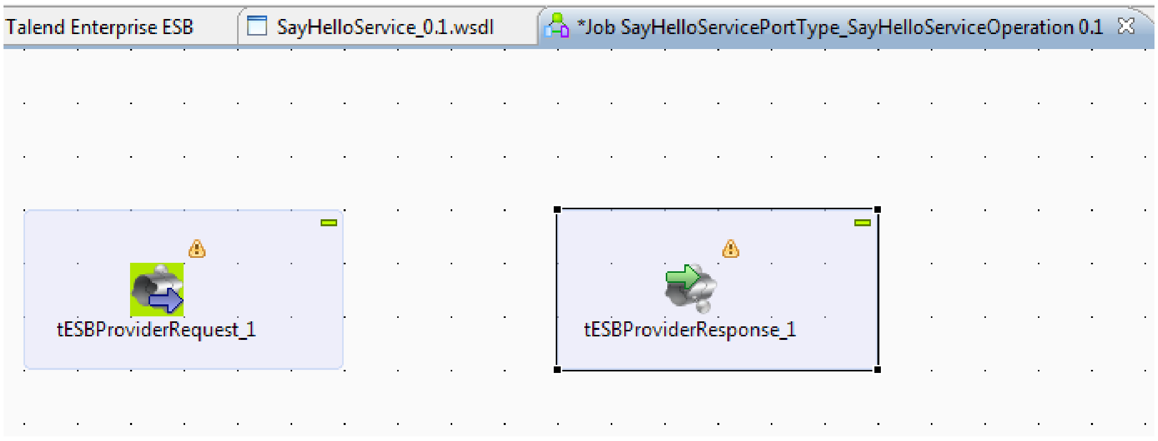


2. Next, we implement the operation - expand the elements displayed in **SayHelloService 0.1** and right-click **SayHelloServiceOperation 0.1** and select **Assign Job**, and then select **Create a new Job and Assign it to this Service Operation**.



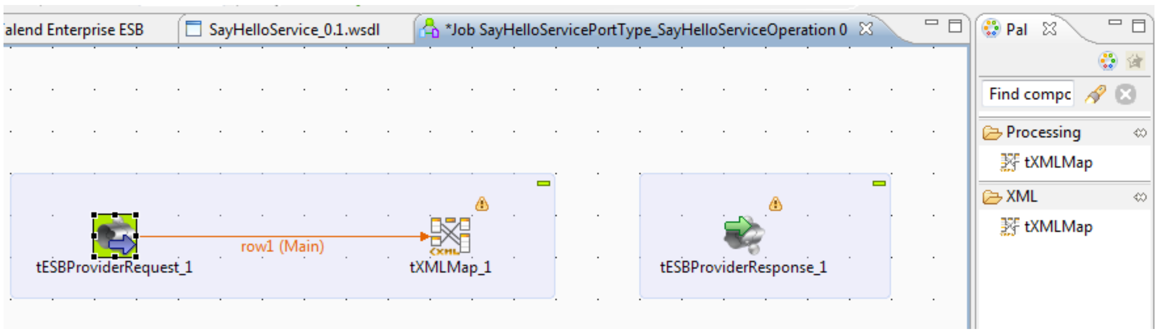
Keep the default name and prefix **SayHelloServicePortType_SayHelloServiceOperation**, and click **Finish**. This creates a new job, which allows us implement the operations using the components in the main grid.

3. Now we start with the default template of the **tESBProviderRequest** and **tESBProviderResponse**. Separate the operations on the grid by clicking the top left of the **tESBProviderResponse_1** square, and drag it to the right.



4. We now add some business logic. `tXMLMap` is a component that transforms and routes data from single or multiple sources to single or multiple destinations. Perform a search for the **tXMLMap** component in the Palette on the right hand side. There may be two instances found under different sections, but they are both the same, so choose either. Drag and drop it between the two operations.

5. Right-click the center of **tESBProviderRequest_1** and select "Row", then "Main" and drop the end of the line on tXMLMap_1.



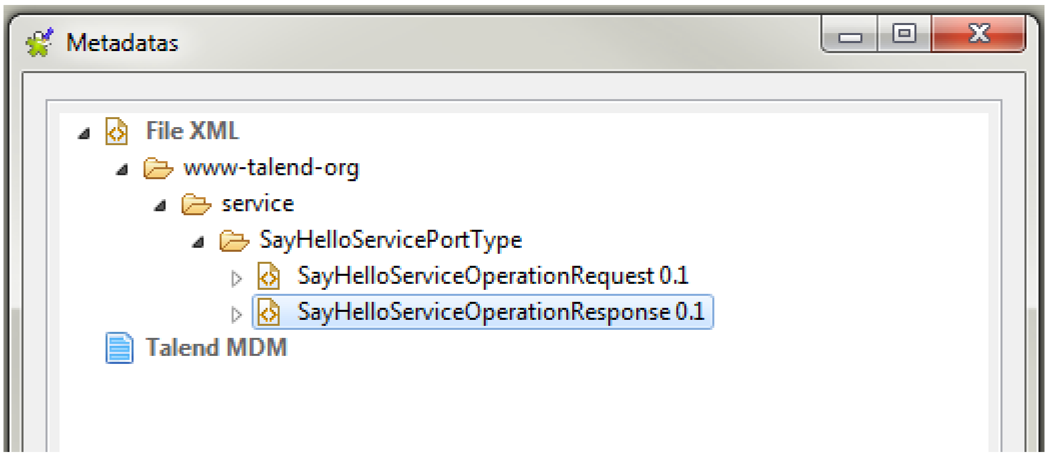
6. Next, right-click **tXMLMap_1** and select **Row**, then **Main** and drop the end of the line on **tESBProviderResponse_1**. Give it the name **`Response`**, and click **Ok**. Click the default **Yes** when asked if you wish to import the schemas.

3.3.3. Implementing the logic

1. Now we start implementing the logic. Double-click **tXMLMap_1**, which opens a new screen. **tXMLMap** is routing the information from the request to the response, and we make use of the existing schema information from the WDSL.

2. Under **main :row1** on the left hand side, right-click **payload** and select **Import from Repository**. In the Metadatas screen that appears, navigate from **File XML** to **SayHelloServiceOperationRequest0.1**. Click **Ok**.

3. In the same way, on the right hand side, import the default response type, right-click payload then **Import from repository** > **File XML** and select **SayHelloServiceOperationResponse 0.1**.

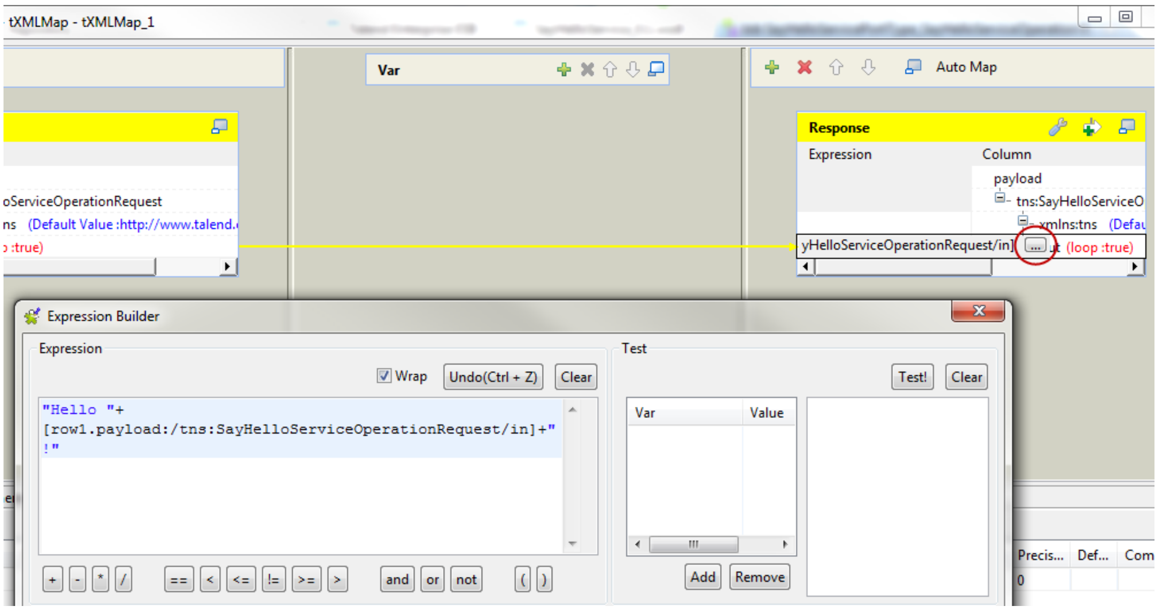

Metadatas
File XML
www-talend-org
service
SayHelloServicePortType
SayHelloServiceOperationRequest 0.1
SayHelloServiceOperationResponse 0.1
Talend MDM

Click **Ok**.

So, the request and response operations are implemented from the existing schemas.

4. Next we simply link the input from one to the output of the other. Left-click **in** on the left hand side and drag it to the **out** expression in the response on the right hand side.

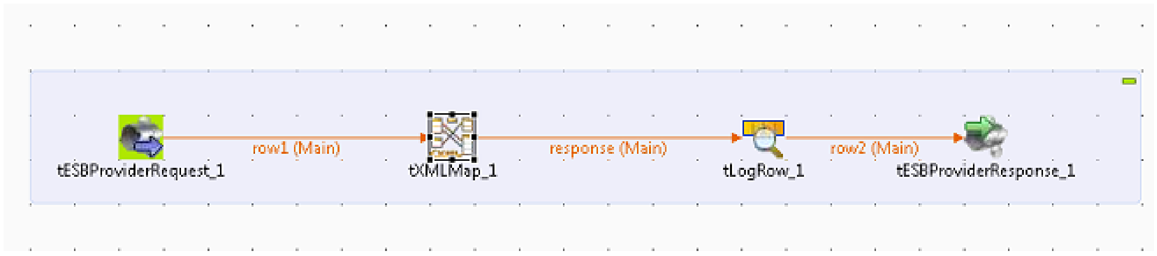
5. Next, we modify the default expression that is sent. On the right hand side, under **Expression**, click the **HelloServiceRequest** value, and double-click the "**...**" button beside it. Edit the expression (which will evaluate to a name) by clicking in the field, and add **`"Hello " +`** before what is there, and **`+"!"`** after it.



```
"Hello "+
[row1.payload:/tns:SayHelloServiceOperationRequest/in]+"
!"
```

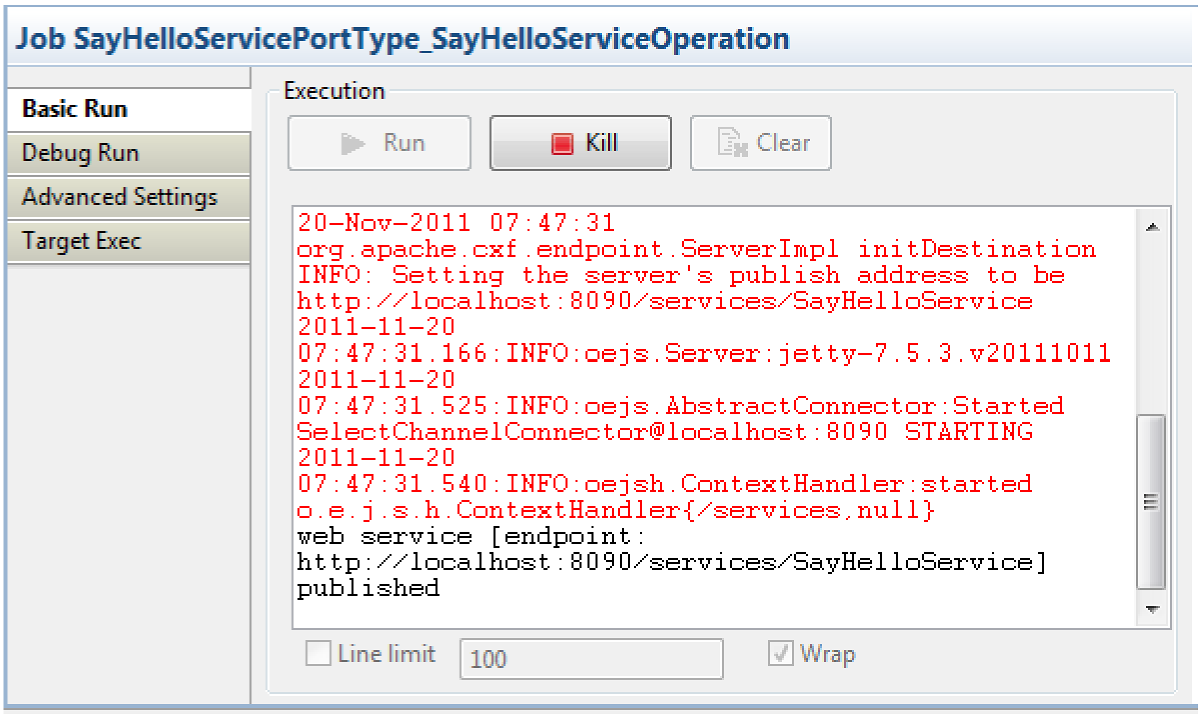
Click **Ok**. You will see the updated expression now on the right hand side. Click **Ok** to return to the main job design screen.

6. Finally, in order to see more as the job executes, we add some logging information. This is done simply by searching for **tLog** in the palette, and dragging **tLogRow** from the Palette on the right hand side and dropping it between the **tXMLMap_1** and **tESBProviderResponse_1**. That is the implementation of the `SayHelloServiceOperation` complete.



3.3.4. Checking the service is working

We do a quick check now to make sure this part is working, by clicking the tab **Run (Job SayHello...)** in middle section in the bottom half of the screen, and then clicking the **Run** button.



```
20-Nov-2011 07:47:31
org.apache.cxf.endpoint.ServerImpl initDestination
INFO: Setting the server's publish address to be
http://localhost:8090/services/SayHelloService
2011-11-20
07:47:31.166:INFO:oejs.Server:jetty-7.5.3.v20111011
2011-11-20
07:47:31.525:INFO:oejs.AbstractConnector:Started
SelectChannelConnector@localhost:8090 STARTING
2011-11-20
07:47:31.540:INFO:oejsh.ContextHandler:started
o.e.j.s.h.ContextHandler{/services,null}
web service [endpoint:
http://localhost:8090/services/SayHelloService]
published
```

This builds the job, and the log output shows that the web_service has been assigned a port 8090 and has been published for other services to use.

Select and copy “http://localhost:8090/services/SayHelloService” for later use.

In Studio you can test each Service operation which you implement in a Job Design individually, to quickly test the implementation. If you would like to test the entire service with multiple operations, you need to export the service and deploy in the runtime. It is not possible to test a service with multiple operations in Studio.

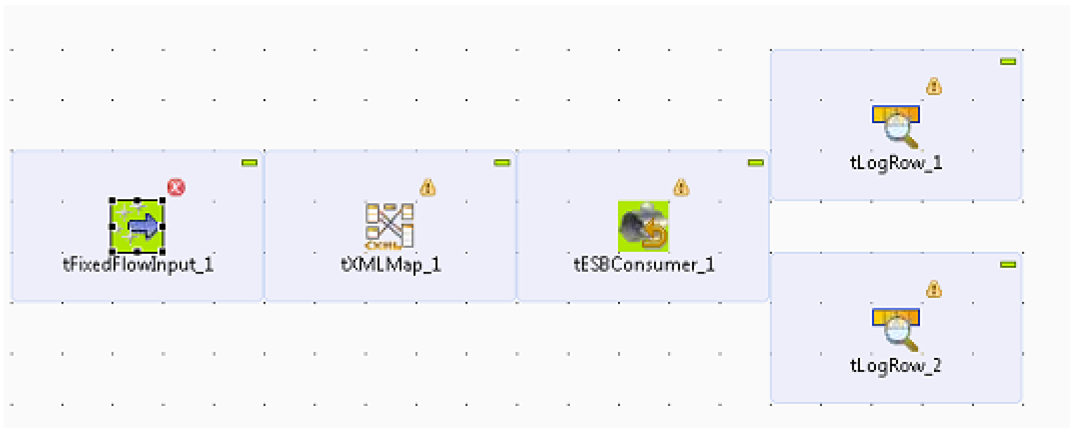
3.4. SayHello consumer

This section creates a "SayHello" consumer, and calls the service with it.

3.4.1. Creating the "SayHello" consumer

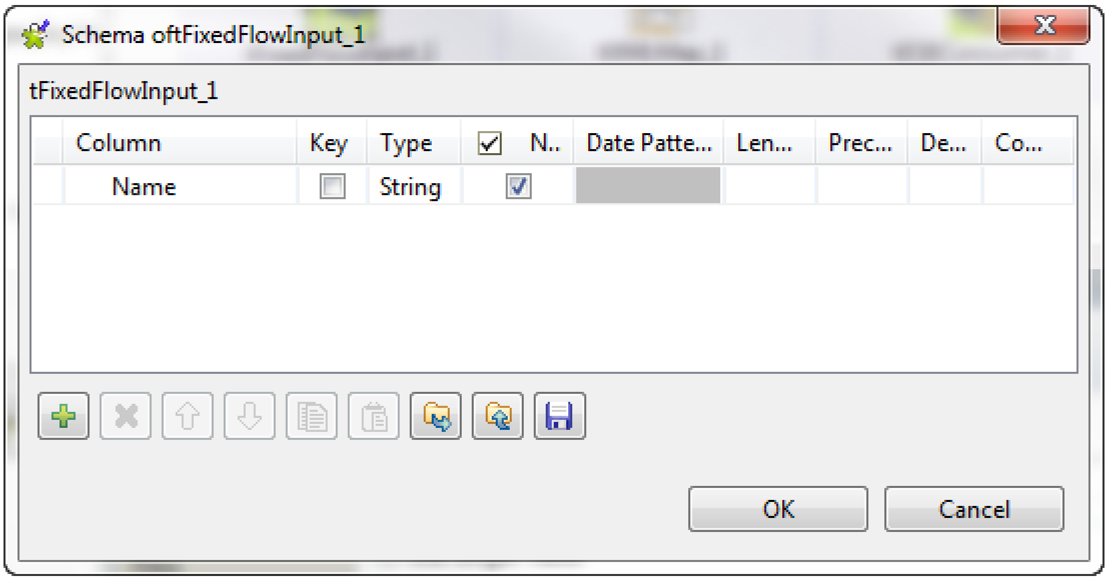
1. Now we're going to easily test this by creating a small consumer job. Right-click **Job Designs** > **Create Job**. In the name field, type **`SayHelloConsumer`**, and the purpose is **`Demo`**. Click **Finish**.

2. Now in the **SayHelloConsumer 0.1** tab, search for **tFixedFlowInput** in the palette, and drag and drop it onto the grid. Similarly, drag and drop **tXMLMap**, **tESBConsumer**, and **tLogRow** (twice) as shown below.



Note: `tFixedFlowInput` generates as many lines and columns as you want using context variables, and `tESBConsumer` calls a specified method from the invoked Web service, and returns a class, based on parameters.

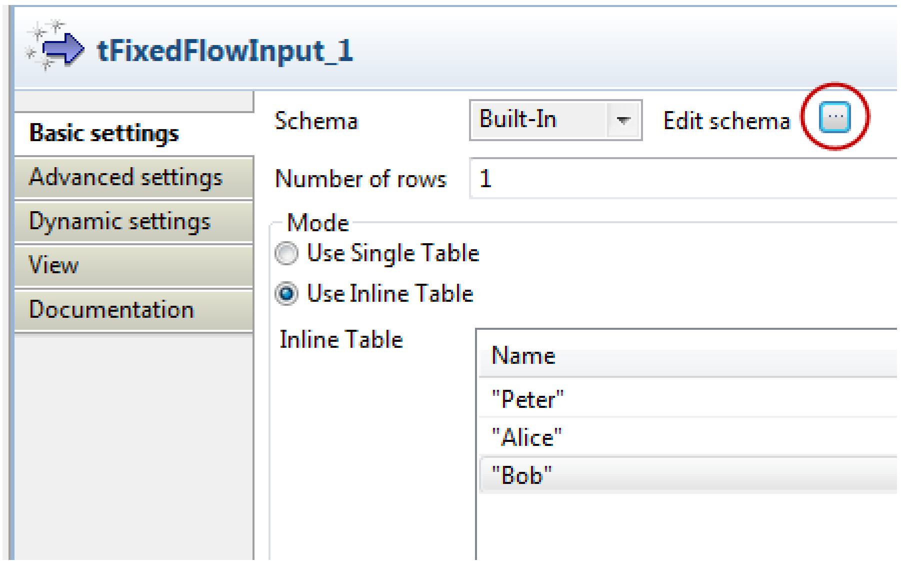
3. Now we are ready to start implementing. Click the center of **tFixedFlowInput_1** and in Component tab below, select **Use Inline Table**. Then click the **"..."** button next to **Edit schema** to open the Schema editing window.



Column	Key	Type	✓ N..	Date Patte...	Len...	Prec...	De...	Co...
Name	☐	String	✓					

In this window, click **"+"** to add a string argument, and rename **newColumn** to **Name** and click **Ok** to close this window.

4. Returning to the **tFixedFlowInput_1** Component tab, use the **“+”** button to add sample rows, and successively replace the “newline” text with names “Peter”, “Alice” and “Bob”.


tFixedFlowInput_1

Schema: Built-In
Edit schema
Number of rows: 1
Mode
Use Single Table
[✓ Use Inline Table]
Inline Table

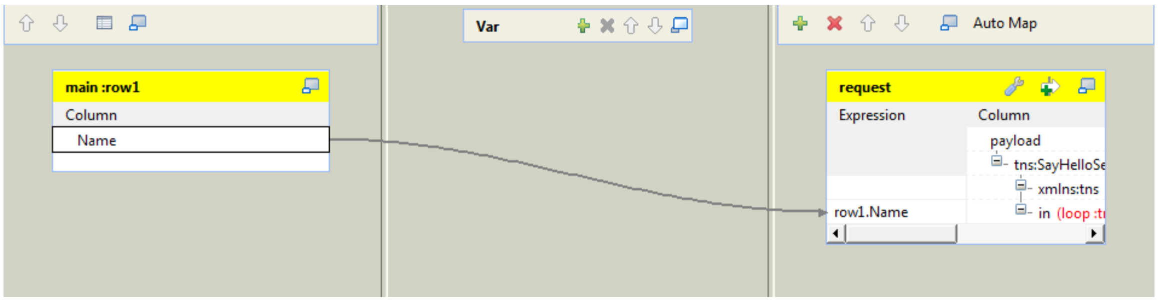
Name
"Peter"
"Alice"
"Bob"

This is the example data that the consumer will send to the SayHelloService.

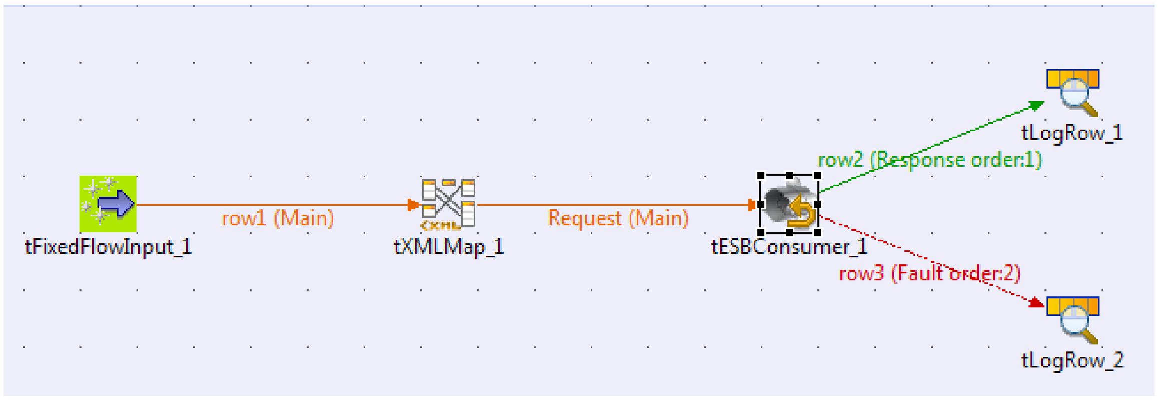
5. Now, in the main **Job SayHelloConsumer 0.1** tab, link the outputs as before, by right-clicking and dragging **tFixedFlowInput_1 > row > main** to **tXMLMap_1**. Then right-click **tXMLMap_1 > row** with a new Output name of **`request`** (click the default **yes** to get the schema of the target component), and drop the end on **tESBConsumer_1**.

6. Now double-click the **tXMLMap** to implement it as before. On the right hand side, click **payload > Import From Repository**. In the Metadatas screen that appears navigate from **File XML** to **SayHelloServiceOperationRequest 0.1**, and click it. This enables us to call the service operation. Click **Ok**.

7. Now left-click and drag the **Name** on the left hand side to the **in** parameter on the right hand side and click **Ok** to return to the main screen.



8. Finally, take care of the logging information. Right-click the center of **tESBConsumer_1**, drag and select **Row > Response** and drop the end on **tLogRow_1** so that any responses should go there. Similarly, right-click **tESBConsumer_1**, select **Row > fault** and drop the end on **tLogRow_2** so that any faults should go there.

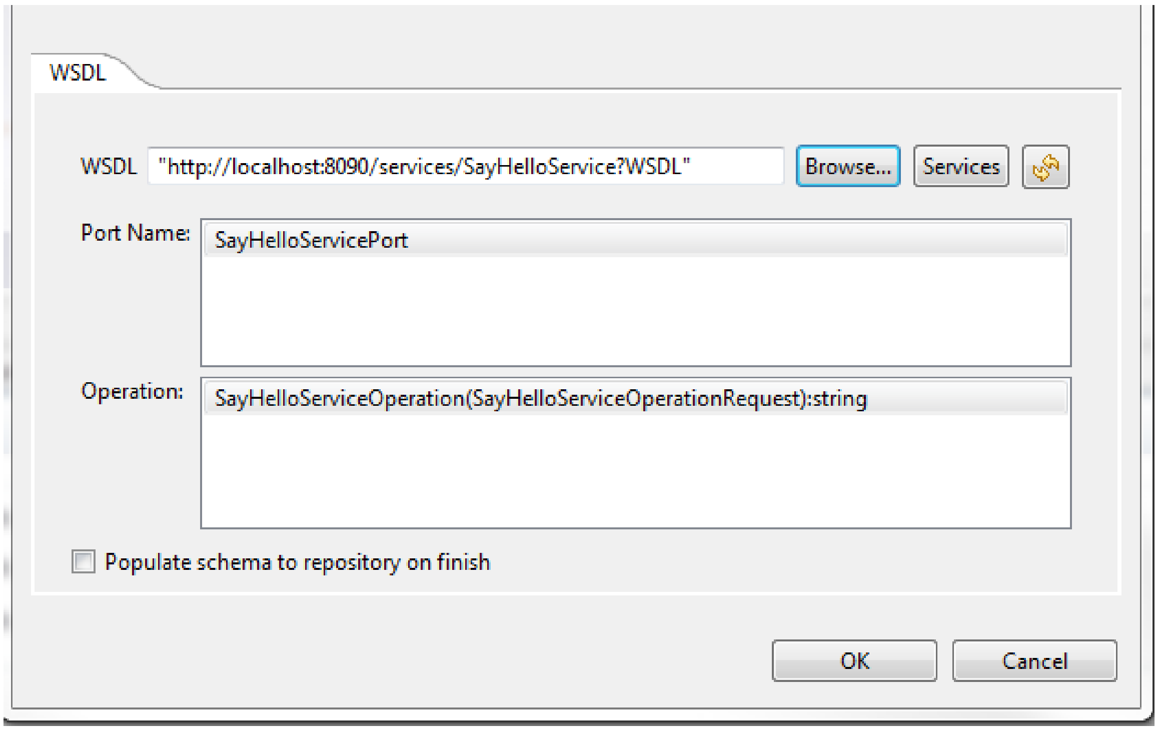


That is the consumer job implementation complete.

In summary, `tFixedFlowInput` generates “Peter”, “Alice”, “Bob”, these will be passed by `tXMLMap` to `tESBConsumer`, which will do three corresponding invocations on the target provider.

3.4.2. Running the consumer

1. Now, we need to point the consumer at the correct WSDL endpoint for the Service. So we go to the **Component** tab, and edit the endpoint there to reference the correct service. Click the **"..."** button next to Service Configuration, and the WSDL configuration window opens. Paste in "http://localhost:8090/services/SayHelloService" to replace the service address there, giving a full address of "http://localhost:8090/services/SayHelloService?WSDL" and click the refresh button to the right to load the information. Click **Ok**.



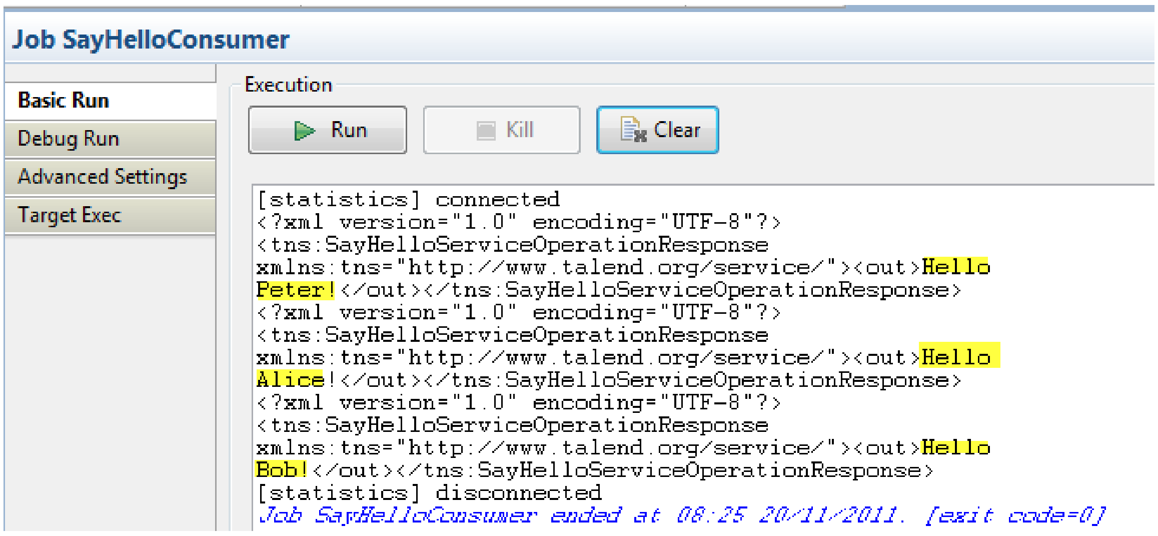
WSDL "http://localhost:8090/services/SayHelloService?WSDL"

Port Name: SayHelloServicePort

Operation: SayHelloServiceOperation(SayHelloServiceOperationRequest):string

☐ Populate schema to repository on finish

2. Finally, we run the consumer job. Click the **Run (Job SayHelloConsumer)** tab, and click **Run**. The job builds and executes, and we see in the output the three names in a Hello message.



```
[statistics] connected
<?xml version="1.0" encoding="UTF-8"?>
<tns:SayHelloServiceOperationResponse
xmlns:tns="http://www.talend.org/service/"><out>Hello
Peter!</out></tns:SayHelloServiceOperationResponse>
<?xml version="1.0" encoding="UTF-8"?>
<tns:SayHelloServiceOperationResponse
xmlns:tns="http://www.talend.org/service/"><out>Hello
Alice!</out></tns:SayHelloServiceOperationResponse>
<?xml version="1.0" encoding="UTF-8"?>
<tns:SayHelloServiceOperationResponse
xmlns:tns="http://www.talend.org/service/"><out>Hello
Bob!</out></tns:SayHelloServiceOperationResponse>
[statistics] disconnected
Job SayHelloConsumer ended at 08:25 20/11/2011. [exit code=0]
```

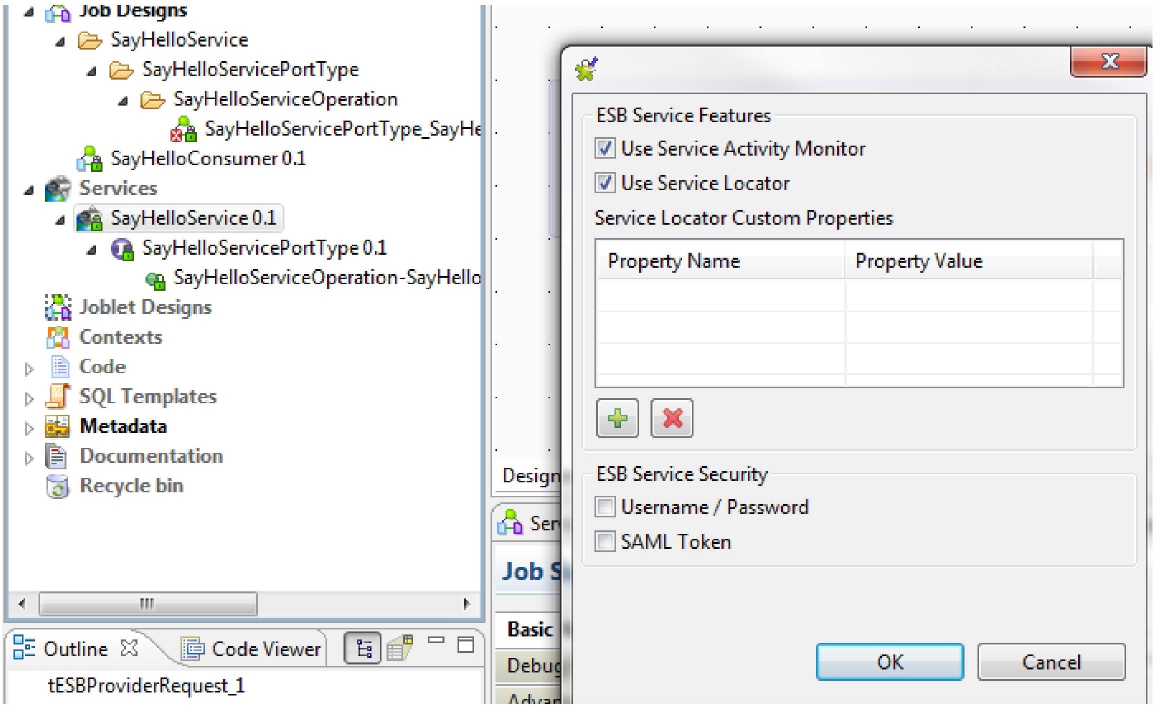
So, we have successfully created and run a "SayHello" consumer and provider in Studio.

3.5. Exporting the service and running it in an Talend Runtime container

In this section, we export the SayHello service to run in an OSGi container, the Talend Runtime container.

Before we export the service, first start a Talend Runtime container, as described in Section 2.6, *"Starting a Talend Runtime container"*

1. We may configure some settings before exporting. Under Services, right-click **SayHelloService 0.1** and select **ESB Runtime Options**. You can optionally enable Talend ESB specific settings, but they are not needed for this demo. Here we have selected the Service Activity Monitoring and Service Locator.



ESB Service Features

[✓] Use Service Activity Monitor

[✓] Use Service Locator

Service Locator Custom Properties

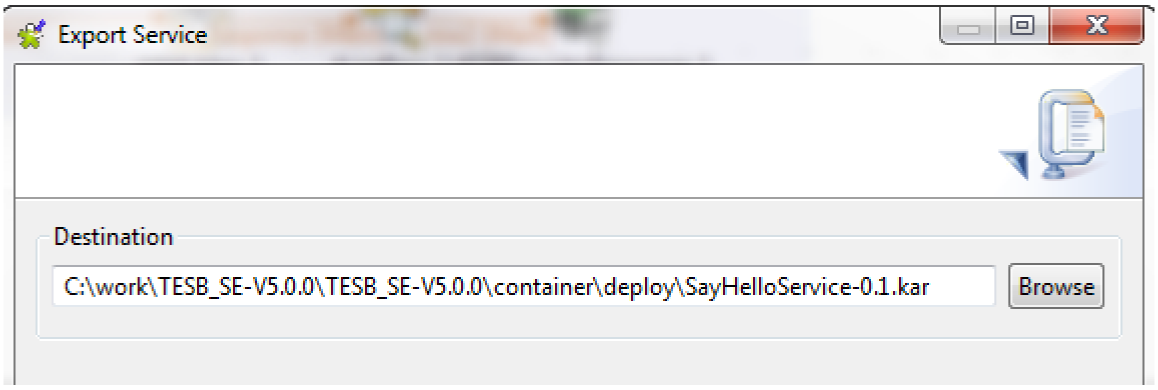
Property Name	Property Value

ESB Service Security

[ ] Username / Password

[ ] SAML Token

2. Now we export the service. Under **Services**, right-click **SayHelloService 0.1** and select **Export Service**. In the **Save As** window, we can specify the `deploy` folder within the Talend Runtime container `deploy` subdirectory, so that the service is directly ready to run and click **Finish**. The service is saved as a file `SayHelloService-0.1.kar`. We show here an example on Windows:



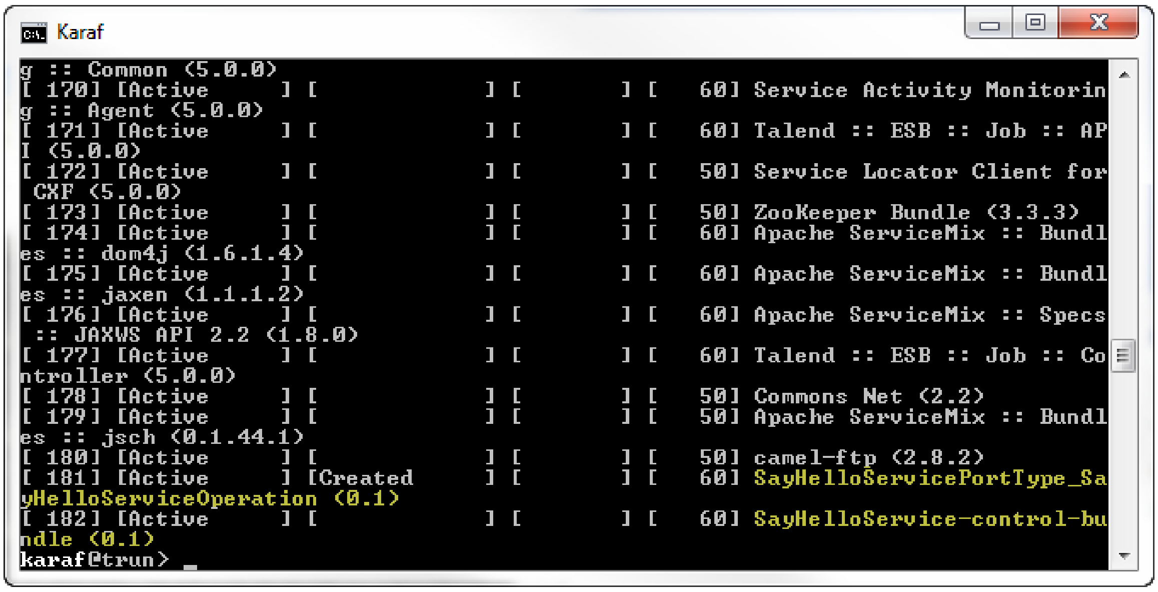
Export Service

Destination

C:\work\TESB_SE-V5.0.0\TESB_SE-V5.0.0\container\deploy\SayHelloService-0.1.kar

This process builds and exports the job to the deploy directory.

3. Since this a dynamic loading environment, the service starts running automatically. If we now look at the container window, and type **`list`**, we will see it there.



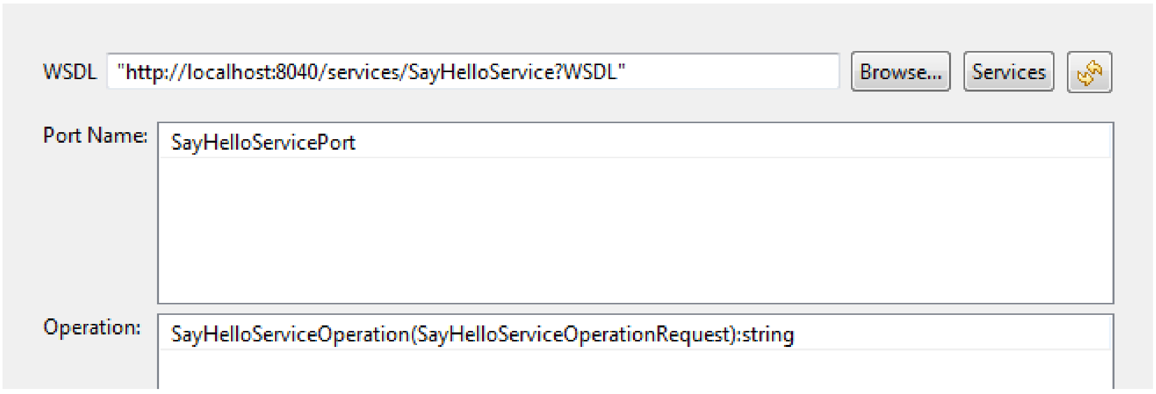
```
g :: Common (5.0.0)
[ 170] [Active     ] [            ] [       ] [   60] Service Activity Monitorin
g :: Agent (5.0.0)
[ 171] [Active     ] [            ] [       ] [   60] Talend :: ESB :: Job :: AP
I (5.0.0)
[ 172] [Active     ] [            ] [       ] [   50] Service Locator Client for
 CXF (5.0.0)
[ 173] [Active     ] [            ] [       ] [   50] ZooKeeper Bundle (3.3.3)
[ 174] [Active     ] [            ] [       ] [   60] Apache ServiceMix :: Bundl
es :: dom4j (1.6.1.4)
[ 175] [Active     ] [            ] [       ] [   60] Apache ServiceMix :: Bundl
es :: jaxen (1.1.1.2)
[ 176] [Active     ] [            ] [       ] [   60] Apache ServiceMix :: Specs
 :: JAXWS API 2.2 (1.8.0)
[ 177] [Active     ] [            ] [       ] [   60] Talend :: ESB :: Job :: Co
ntroller (5.0.0)
[ 178] [Active     ] [            ] [       ] [   50] Commons Net (2.2)
[ 179] [Active     ] [            ] [       ] [   50] Apache ServiceMix :: Bundl
es :: jsch (0.1.44.1)
[ 180] [Active     ] [            ] [       ] [   50] camel-ftp (2.8.2)
[ 181] [Active     ] [Created     ] [       ] [   60] SayHelloServicePortType_Sa
yHelloServiceOperation (0.1)
[ 182] [Active     ] [            ] [       ] [   60] SayHelloService-control-bu
ndle (0.1)
karaf@trun> _
```

4. Now we check it is working, by starting the consumer.

5. The port that the service is running at has changed, and now uses the container port, which is by default 8040, so we need to update the port. Under **Job Designs**, click **Job SayHelloConsumer 0.1**. Click the middle of **tESBConsumer_1**. Then go to the **Component** tab.

6. Click the **"..."** button next to Service configuration, which opens a WSDL settings window.

7. We need to update the port number to use the Talend Runtime container port - so change 8090 to 8040, and click the refresh button.



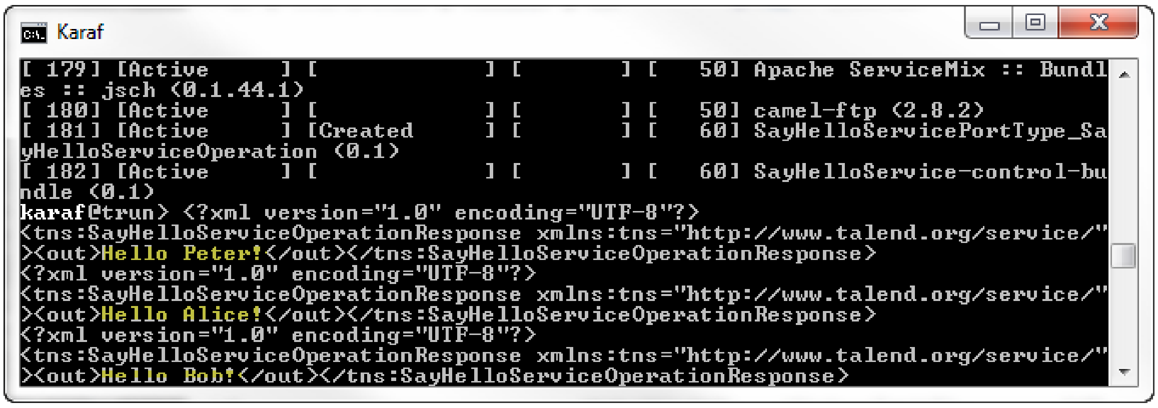
WSDL "http://localhost:8040/services/SayHelloService?WSDL"

Port Name: SayHelloServicePort

Operation: SayHelloServiceOperation(SayHelloServiceOperationRequest):string

8. Now run the consumer job as before from the **Job** tab, and we see the same output as before.

9. If we look at the output in the Talend Runtime container, we also now see the Hello messages generated by the server running in the container.



```
[ 179] [Active     ] [            ] [       ] [   50] Apache ServiceMix :: Bundl
es :: jsch (0.1.44.1)
[ 180] [Active     ] [            ] [       ] [   50] camel-ftp (2.8.2)
[ 181] [Active     ] [Created     ] [       ] [   60] SayHelloServicePortType_Sa
yHelloServiceOperation (0.1)
[ 182] [Active     ] [            ] [       ] [   60] SayHelloService-control-bu
ndle (0.1)
karaf@trun> <?xml version="1.0" encoding="UTF-8"?>
<tns:SayHelloServiceOperationResponse xmlns:tns="http://www.talend.org/service/"
><out>Hello Peter!</out></tns:SayHelloServiceOperationResponse>
<?xml version="1.0" encoding="UTF-8"?>
<tns:SayHelloServiceOperationResponse xmlns:tns="http://www.talend.org/service/"
><out>Hello Alice!</out></tns:SayHelloServiceOperationResponse>
<?xml version="1.0" encoding="UTF-8"?>
<tns:SayHelloServiceOperationResponse xmlns:tns="http://www.talend.org/service/"
><out>Hello Bob!</out></tns:SayHelloServiceOperationResponse>
```

So that is how to create a simple SayHello consumer and service, running them Studio, and also running the service in the Talend Runtime container.

3.6. SayHelloRoute example

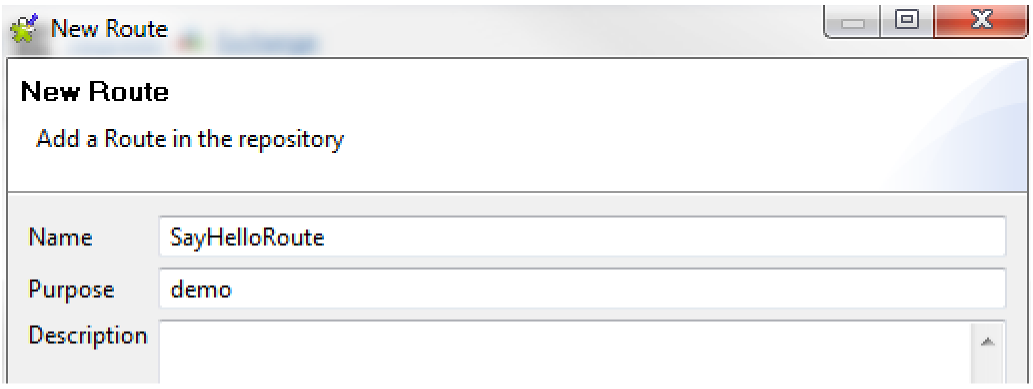
In this example, we extend our existing SayHello consumer and provider. We will have two instances of the provider, and using the Mediation perspective, we build a route that filters the Hello messages by name, so that messages with the name "Alice" in them go to one provider, and other names go to the second provider.

Finally, we will run the consumer from the first example, and show the messages coming from the consumer, and being routed to the correct service.

3.6.1. Creating the route

1. First, we switch from the Integration perspective to the Mediation perspective by clicking **Mediation** in the top right hand corner.

2. We create a new route by right-clicking **Routes** > **Create Routes**. Give the name **`SayHelloRoute`** and purpose is **`demo`**, and click **Finish**.



New Route

New Route

Add a Route in the repository

Name: SayHelloRoute

Purpose: demo

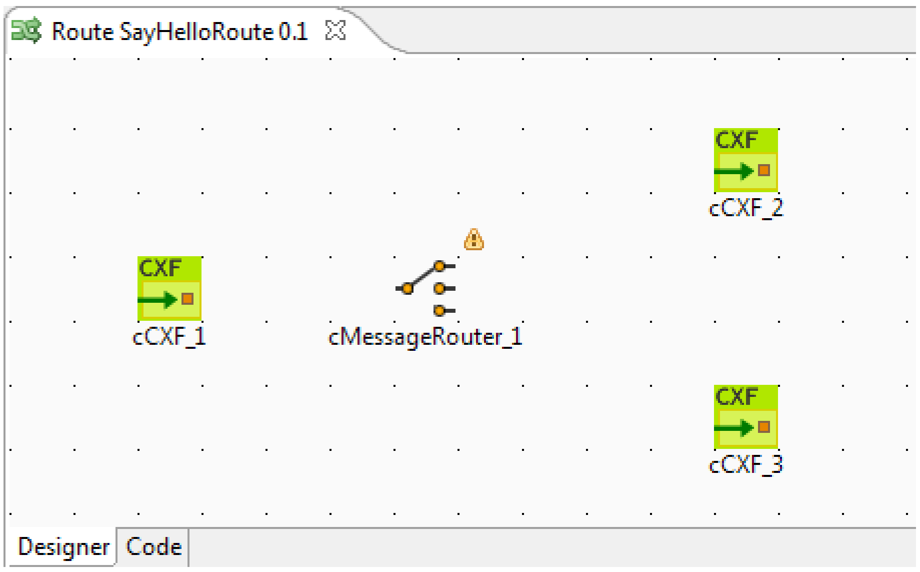
Description:

3. Click **Routes** > **SayHelloRoute 0.1** to display the route grid. Now you will notice that the palette has changed from the one in the Integration perspective to the Mediation perspective. Now we will create a typical content-based router, dragging and dropping components from the palette to the route grid.

   The request message is coming in from the consumer, so we drag and drop the **Messaging** > **cCXF** component, which intercepts messages coming into server endpoints.

   Then, as we are doing a content-based route, we drag and drop the **Routing** > **MessageRouter** component, which reroutes messages depending on a set of conditions.

   Then we get two more instances of **Messaging** > **cCXF** for the two target services.



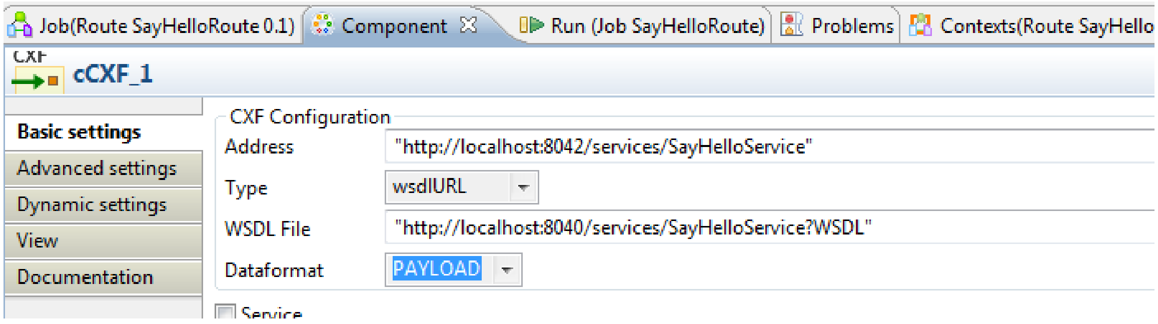
Having multiple **cCXF** components with the same label in a Route is not supported

It is recommended to label each component with a unique name to better identify its role in the Route.

Having duplicate labels may cause problems for the code generation of some components. For example, when using the **cCXF** component, if you have two components with the label **cCXF_1**, this will cause errors in the code generation phase.

4. To implement this, we just need to add some parameters. So, click cCXF_1, and then click the Component tab below:

   - In the address component, we paste in the previous service address “http://localhost:8042/services/SayHelloService”, and update the port to be 8042, since the new service will be listening on this port.
   - We use the WSDL file from the original service. We use the http:// address to get the live background service information. It's in "http://localhost:8040/services/SayHelloService?WSDL", for all three services.
   - The dataformat is PAYLOAD, that is, we are looking at the message body.



Address "http://localhost:8042/services/SayHelloService"
Type wsdlURL
WSDL File "http://localhost:8040/services/SayHelloService?WSDL"
Dataformat PAYLOAD
Service

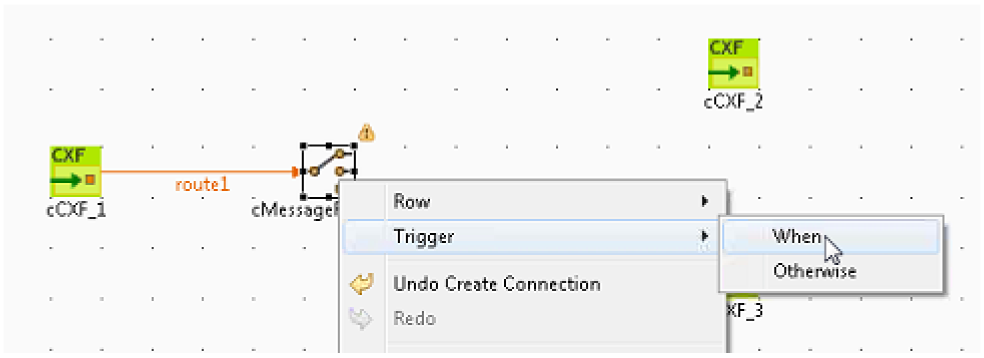
At any point, you can save the current information by selecting **File** > **Save** or clicking the **Save** icon in the top left hand corner.

5. We repeat the previous step for the other components: *cCXF_2* - except that the port number in the Address field is 8040, and for *cCXF_3*, where the port number in the Address field is 8041.

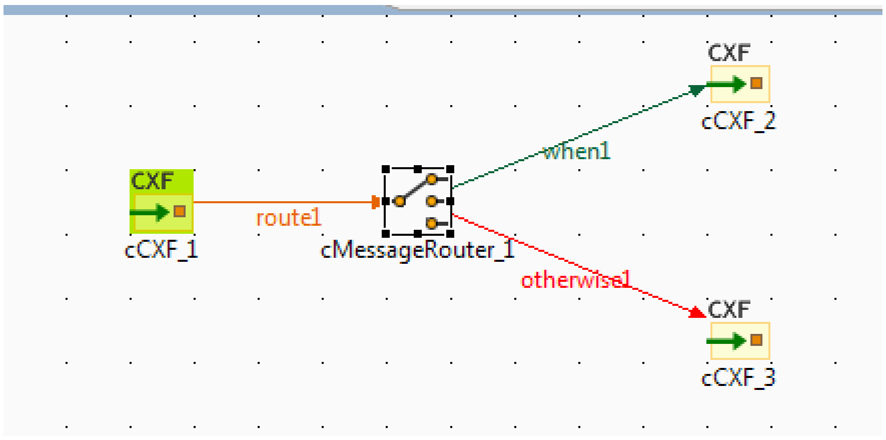
   So, in summary, we get a request in on port 8042 (*cCXF_1*), and we send it to either port 8040 (*cCXF_2*) or 8041 (*cCXF_3*), depending on the contents of the message.

6. So now, we start connecting the elements. Connect *tCXF_1* to *cMessageRouter_1* by right-clicking the center of **tCXF_1** and selecting **Row** > **Route** and dropping the end onto **cMessageRouter_1**.

7. Then we create a **When** trigger for one service, by right-clicking **cMessageRouter_1**, selecting **Trigger > When** and dropping the end on **cCXF_2**.

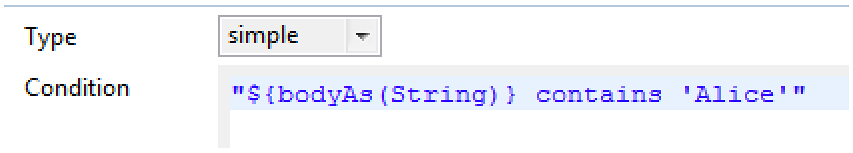


8. Similarly, we add an **Otherwise** trigger from **cMessageRouter_1** to **cCXF_3** by right-clicking **cMessageRouter_1**, selecting **Trigger > Otherwise** and dropping the end on **cCXF_3**. This results in the following:



9. We need to add the **When** condition. Right-click the **when1** line, which brings up a small dialog.

   Select Type **simple**, and the Condition is "${bodyAs(String)} contains 'Alice'".



Type: simple

Condition: "${bodyAs(String)} contains 'Alice'"

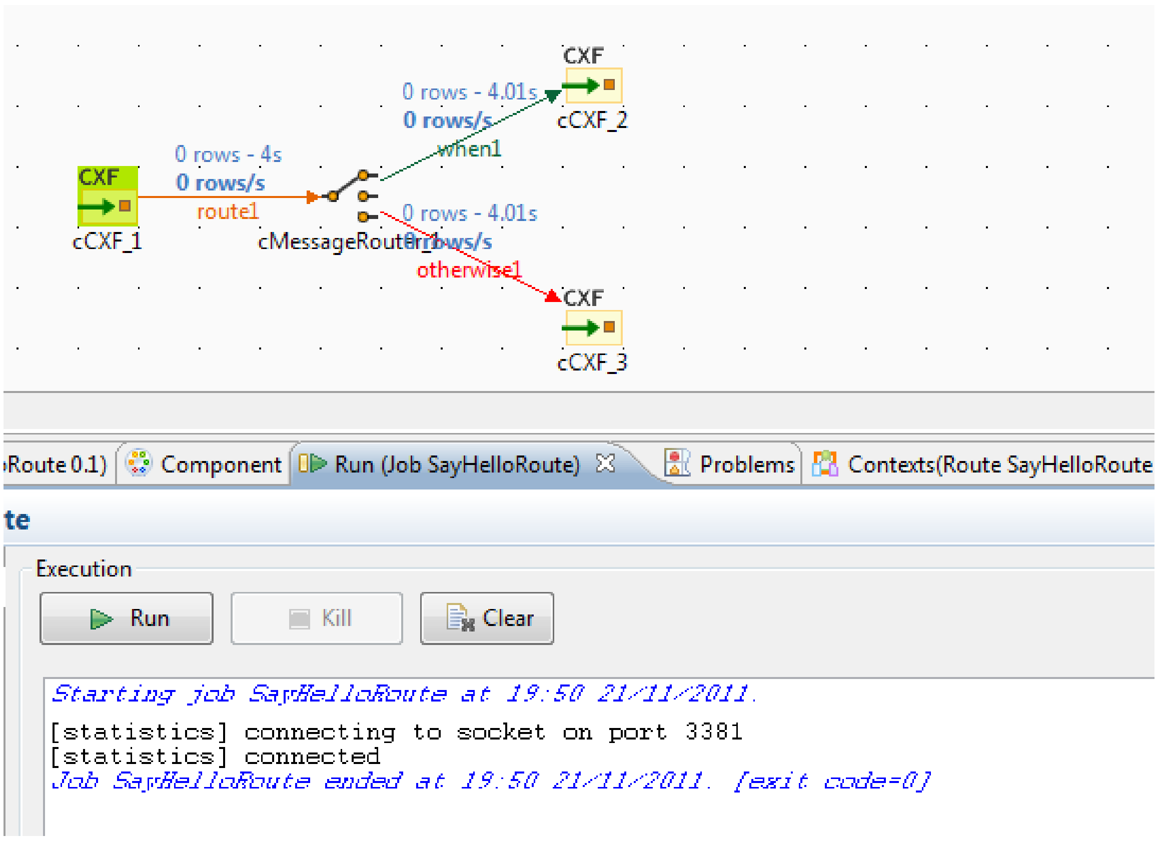
   That is, any messages with "Alice" in the body will be routed to a service in a Talend Runtime container that listens on port 8040.

3.6.2. Running the services

1. First, we need to create a second, alternate container - see Section 2.7, *"Creating an alternate container"* for details on how to this. Check that the default container is running - restart it if necessary, and start the alternate container.

2. Then we need to take a copy of the kar file of the service we created (Talend ESB installation directory `/container/deploy/SayHelloService-0.1.kar`), and paste it into the deploy folder in the alternate container `/alternate-container/deploy`. (This may be done already if you copied the alternate-container after creating the SayHelloService).

3. To do a simple check that all this has been done correctly, before we add the consumer, we will use the route execution in the studio. Click the "Run (Job SayHelloRoute) tab" and click Run:

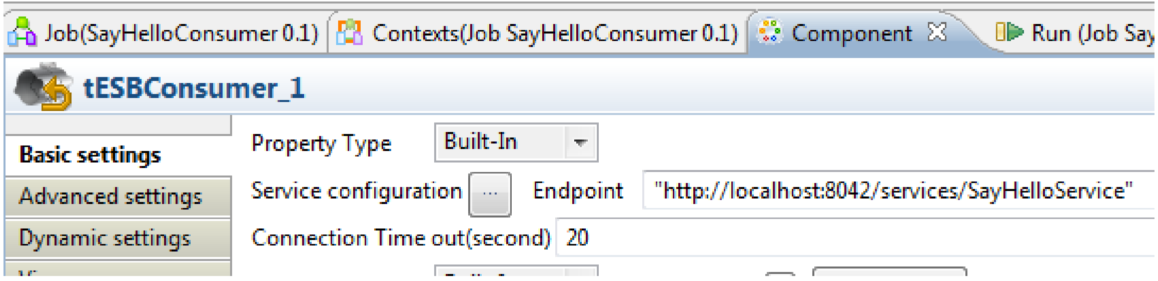


```
Starting job SayHelloRoute at 19:50 21/11/2011.
[statistics] connecting to socket on port 3381
[statistics] connected
Job SayHelloRoute ended at 19:50 21/11/2011. [exit code=0]
```

This checks that the CXF configuration information is syntactically correct. It also gives an output of "0 rows" on the grid, which reflect the flow of messages, so that you know that the parts are connecting. Leave the job running.

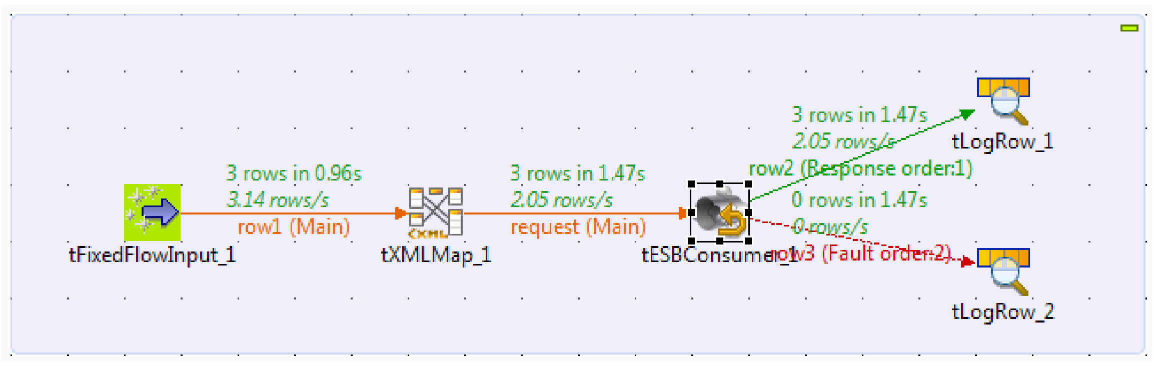
If you get syntax errors, then click on each component in turn, and examine it in the Component tab. Check in particular that the double quotes are all there, and that the port numbers are correct.

4. Now, we switch back to our consumer to run the demo for real. Click **Integration perspective** in the top right hand corner. Then click "SayHelloConsumer 0.1" under "Job Designs"; then the consumer information is displayed on the main grid. We need to update the port number, so click the Component tab, and the **"..."** button next to Service Configuration. Update the port number to be 8040, and click the refresh button to retrieve the WSDL information. Now update the endpoint to be that of the route - http://localhost:8042/services/SayHelloService.

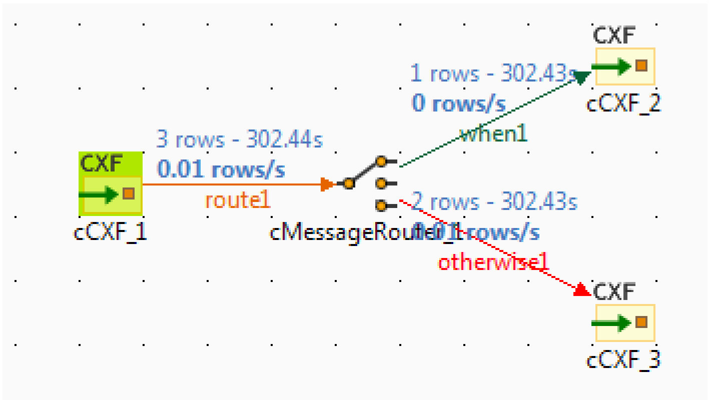


Property Type: Built-In
Service configuration ... Endpoint "http://localhost:8042/services/SayHelloService"
Connection Time out(second) 20

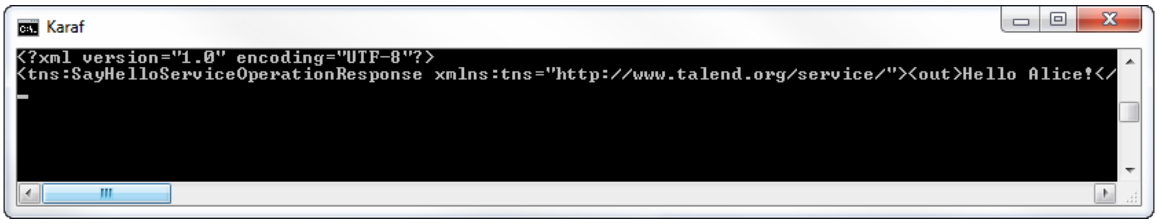
5. We can now send a request by running the consumer job by clicking on the tab "Run (Job SayhelloRoute)" and click Run. We see the "Hello Peter", "Hello Alice", "Hello Bob" output in the consumer output as before. The main grid shows "3 rows", that is, the three messages being passed along with none going to fault.



6. Now we look at the "Route SayHelloRoute 0.1" tab, we will see that 1 message went to the cCXF_2 provider and 2 messages when to the cCXF_3 provider.

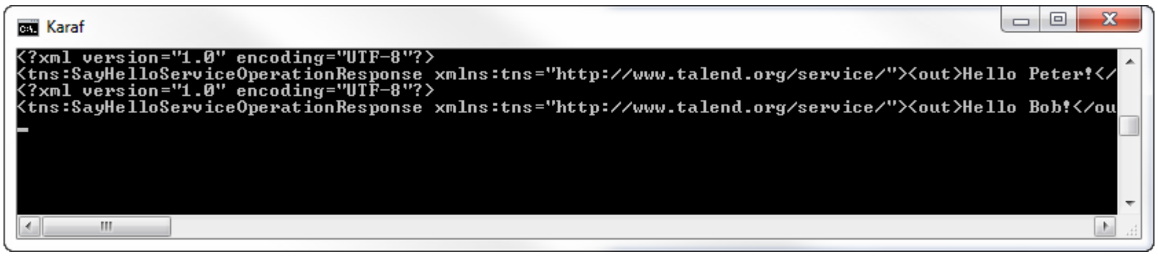


7. if we look at the output in the two containers, we also see that the messages have been routed correctly: This is the output for one container:



```
<?xml version="1.0" encoding="UTF-8"?>
<tns:SayHelloServiceOperationResponse xmlns:tns="http://www.talend.org/service/"><out>Hello Alice!</
_
```

and this is the output for the other:



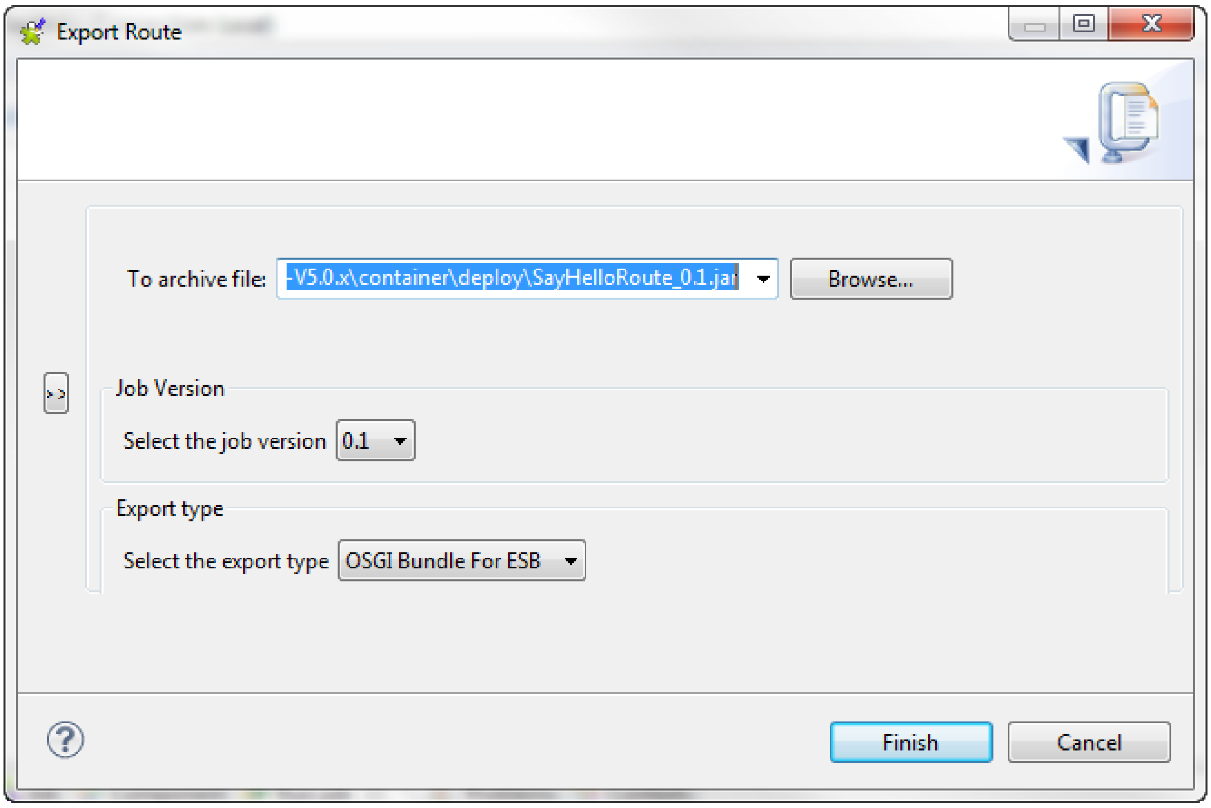
```
<?xml version="1.0" encoding="UTF-8"?>
<tns:SayHelloServiceOperationResponse xmlns:tns="http://www.talend.org/service/"><out>Hello Peter!</
<?xml version="1.0" encoding="UTF-8"?>
<tns:SayHelloServiceOperationResponse xmlns:tns="http://www.talend.org/service/"><out>Hello Bob!</ou
_
```

8. That is the SayHelloRoute demo completed.

3.6.3. Exporting Routes

You can export a route in a similar way to exporting jobs; they are exported as jar files. For example, under Routes, right-click SayHelloRoute 0.1, and select "Export Route".

Specify a path for the jar file. Then specify the type of output - the default is "OSGi Bundle for ESB", but you can also save it as an Autonomous Route to run in other environments.


Export Route

To archive file: -V5.0.x\container\deploy\SayHelloRoute_0.1.jar

Job Version

Select the job version 0.1

Export type

Select the export type OSGI Bundle For ESB

This screen shows the route being exported to a `/container/deploy` directory in a Talend Runtime so that it is directly deployed, but it can also be stored elsewhere and copied to the deploy directory later.

Chapter 4. A real-world Rent-a-Car demo example

This chapter describes the simplest version of the Rent-a-Car demo.

In the course of working through this example you will learn how to:

- Compile and build Apache CXF services (Rent-a-Car)
- Start an OSGi container
- Deploy services using commands at an OSGi container console

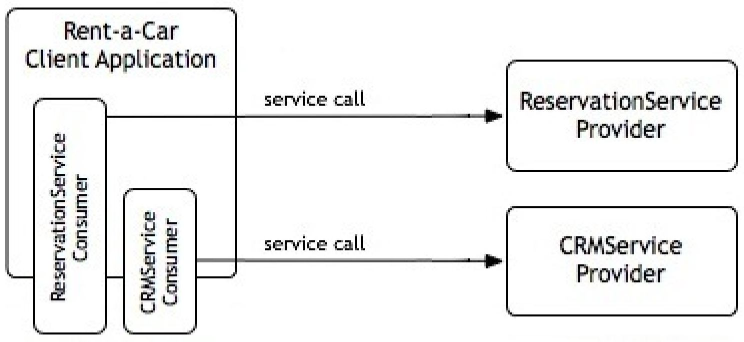
By default, all of the Rent-a-Car demos are enabled with JMX.

The Rent-a-Car demo code can be built with different options (Maven profiles) in order to to use additional functionality, and these are discussed in later chapters - this chapter uses the default, basic build.

4.1. Overview of Rent-a-Car demo example

For this demo example we have chosen a common business use case scenario: a simplified real-world example from the domain of car rental companies. This uses the functionality of the Customer Relationship Management (CRM) service to supply information about the customer, and the reservation service to reserve a car.

Figure 4.1. Rent-a-Car client application and services

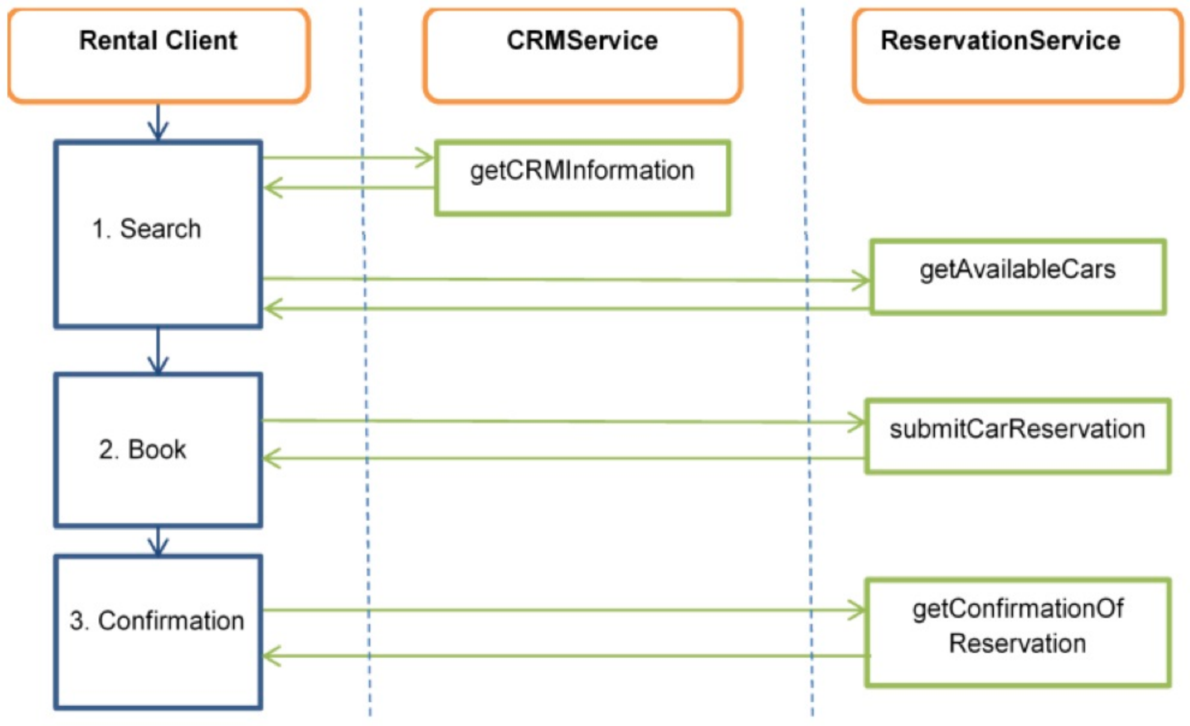

Rent-a-Car Client Application
ReservationService Consumer
CRMService Consumer
service call
service call
ReservationService Provider
CRMService Provider

In this scenario, there are two main services that will be demonstrated, `ReservationService` and `CRMService`. You will create and orchestrate these services with the help of a simple graphical Rent-a-Car client application, simply called the Rental Client.

The Rental Client provides the basic functionality to search for the cars available within a given time period. To implement its business logic the Rental Client calls operations from both the CRM Service and the Reservation Service as shown in the diagram below.

1. The **search** takes into account the status (silver, gold, platinum) of the customer you select. This customer profile information is passed to the reservation service and is used to determine the outcome of the search.
2. From the list of results you can select the car that best fulfills the customer requirements and then make a reservation simply by clicking on the **Book** button.
3. At the end of the submission process, the booking **confirmation** is displayed along with a Reservation ID.

Figure 4.2. Rental Client service calls



4.2. Building the demo

We use the term `<TalendRuntimePath>` for the directory where Talend Runtime is installed. This is typically the full path of either `Runtime_ESBSE` or `Talend-ESB-V5.2.x`, depending on the version of the software that is being used (see Section 2.2, *"Downloading, extracting the software"*). Please substitute appropriately.

The Talend Runtime examples are in the `<TalendRuntimePath>/examples/talend` directory.

4.2.1. Rent-a-Car demo structure

This section describes the directory structure of the demo; this consists of the two services and a demo application, plus some support files. These are found in the directory `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/`:

Directory or file name	Description
`app-reservation/`	commands and simple UI client which let users invoke the services step-by-step.
`crmservice/`	Customer Relationship Management (CRM) Service which implements the `getCRMInformation` and `getCRMStatus` operations.
`reservationservice/`	Reservation Service which implements `getAvailableCars`, `submitCarReservation` and `getConfirmationOfReservation` operations.

Directory or file name	Description
`features/`	the feature files which will be used to install the Rent-a-Car demo to an OSGi container.
`soapui/`	optional project files for the soapUI tool.
`pom.xml`	used by the Maven build process.
`settings.xml`	advanced functionality (NOT used by basic demo - see Section 8.5, "*Uploading the demo to a local Archiva repository using Maven files*" for details).

Now we look in detail at the services and client subdirectories:

1. first we look at `crmservice`:

Directory or file name	Description
`common/`	This directory contains the `CRMService.wsdl` file which is used to generate the initial code.
`client/`	This is an example client application that shows a CXF client invoking the CXF endpoint.
`client-sl-sam/`	This directory contains the `CRMService` client enabled to be used in Service Locator and Service Activity Monitoring scenario.
`client-sts/`	This directory contains the `CRMService` client enabled to be used in Security Token Service scenario.
`client-all/`	This directory contains the `CRMService` client enabled to be used with all features.
`service/`	`CRMService` service implementation.
`service-endpoint-jmx/`	the basic `CRMService` endpoint
`service-endpoint-sl-sam/`	the `CRMService` endpoint enabled to be used in Service Locator and Service Activity Monitoring scenario.
`service-endpoint-sts/`	the `CRMService` endpoint enabled to be used in Security Token Service scenario.
`service-endpoint-all/`	`CRMService` endpoint enabled to be used with all features.

2. `reservationservice` has the same structure, except that the parts relate to the ReservationService

3. `app-reservation` has code to build the commands and simple UI client:

Directory or file name	Description
`src/`	source code
`target/`	target directory for the build
`pom.xml`	used by the Maven build process

4.2.2. Building the Rent-a-Car demo example

From the base directory of the ESB example, that is `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/`, the Maven project file `pom.xml` can be used to build the demo. After changing to this directory, Maven is invoked identically from a Linux terminal or the Windows command line:

```
mvn clean install
```

Running this command, the demo is built, and OSGi bundles are created for deployment into an OSGi container.

You need to build the demo in the root directory, as the build system is configured for this - running it in a subdirectory may generate build errors.

Run the mvn command in the root directory (`<TalendRuntimePath>/examples/talend/tesb/rent-a-car`) to build it.

4.3. Deploying the demo

This section describes how to deploy the Rent-a-Car demo in the Talend Runtime container. This involves a number of steps. (As an introduction, Section 2.6, *"Starting a Talend Runtime container"* gives an overview of starting an OSGi container).

1. Create an alternate OSGi container - see Section 2.7, *"Creating an alternate container"*
2. Start the default OSGi container - see Section 2.6, *"Starting a Talend Runtime container"*
3. Deploy the demo services (see Section 4.3.1, *"Deploying the demo services"*)
4. Deploy the demo client application (see Section 4.3.2, *"Deploying the demo client application"*)

4.3.1. Deploying the demo services

The services feature must be installed in the primary container, not in the alternate container because it relies on the primary's port 8040 to be opened for incoming HTTP calls.

Note: Commands in this document refer to version "5.2.1" of the Talend ESB. If you are using another version, replace the version sub-string **`.../5.2.1/...`** by the number of your version.

1. Add the Rent-a-Car feature location into the primary (`<TalendRuntimePath>/container/`) OSGi container, not in the alternate-container (see warning above).

   At the OSGi console prompt (`karaf@trun>`), type the following command:

   ```
   features:addurl  mvn:org.talend.esb.examples.rent-a-car/features/5.2.1/
   xml
   ```

   This command allows you to use the relevant bundles directly from the Maven repository where they have been installed by the Maven build.

2. Install the Rent-a-Car services feature into the OSGi container.

   Type the following console command:

   ```
   features:install tesb-rac-services
   ```

3. Check feature installation success.

   Type the following console command:

   ```
   list
   ```

   You will see the demo bundles in the list of installed bundles - as you can see, both `CRMService` and `ReservationService` bundles are installed:

```
 ID   State    Blueprint   Spring   Level  Name
[137] [Active ] [       ] [       ] [   60] CRMService Common (5.2.1)
[138] [Active ] [       ] [       ] [   60] ReservationService Common
(5.2.1)
[139] [Active ] [       ] [       ] [   60] CRMService Service (5.2.1)
[140] [Active ] [       ] [       ] [   60] CRMService Service Endpoint
(5.2.1)
[141] [Active ] [       ] [       ] [   60] ReservationService Service
(5.2.1)
[142] [Active ] [       ] [       ] [   60] ReservationService Service
Endpoint (5.2.1)
```

4.3.2. Deploying the demo client application

The demo "Rental Client" application can be deployed in the same OSGi container as the services or in the alternate container (the later simulates a more realistic scenario where client and services are in different containers).

1. If the alternate container is used for deployment of the demo application, add the Rent-a-Car feature location into this OSGi container. The feature location needs to be added only once in each container, even if the container is shut down afterwards and re-started, as state is preserved between runs. However, if you are unsure whether you installed a feature already, adding the feature location again will not cause any problems.

   At the OSGi console prompt, type the following command:

   ```
   features:addurl  mvn:org.talend.esb.examples.rent-a-car/features/5.2.1/
   xml
   ```

   If you have already run a previous example, you may need to run the following command in order to ensure that internal bundle caches are updated:

   ```
   features:refreshurl
   ```

2. Install the Rent-a-Car application feature into the OSGi container.

   Type the following console command:

   ```
   features:install tesb-rac-app
   ```

3. Check feature installation success.

   Type the following console command:

   ```
   list
   ```

   You will see the demo bundles in the list of installed bundles (note you will only see the service bundles installed if you are running both in the same container):

```
 ID   State    Blueprint   Spring   Level  Name
[137] [Active ] [       ] [       ] [   60] CRMService Common (5.2.1)
[138] [Active ] [       ] [       ] [   60] ReservationService Common
(5.2.1)
[139] [Active ] [       ] [       ] [   60] CRMService Client (5.2.1)
[140] [Active ] [       ] [       ] [   60] ReservationService Client
(5.2.1)
[141] [Active ] [Created] [       ] [   60] App Reservation (5.2.1)
```

4.4. Running the demo

4.4.1. Use case scenario for Rent-a-Car

This is the sequence of events in the Rent-a-Car demo:

1. A customer enters prerequisite data into a form such as her name, the pick-up and return dates, the preferred range of rental rates, and submits her query.
2. Next, the application calls the `CRMService` to retrieve the customer data and customer status information.
3. The customer is presented a list of available cars and selects one car from the list.
4. Finally, the application calls the `ReservationService` to submit the reservation request and then displays a reservation confirmation notice to the customer.

4.4.2. Starting the application

At this point, `ReservationService`, `CRMService`, and `app-reservation` (the client) are already deployed in Talend Runtime. Start the OSGi console as described previously. If demo services and demo application are deployed in two different containers, both must be started.

In fact, there are GUI and command line versions of the Rental Client. Thus, on the console of the container with the demo application you will see:

```
---TALEND ESB: Rent-a-Car (OSGi) Commands---
car:gui        (Show GUI)
car:search <user> <pickupDate> <returnDate>
   (Search for cars to rent, date format yyyy/mm/dd)
car:rent <pos>
   (Rent-a-Car listed  in search result of carSearch)
```

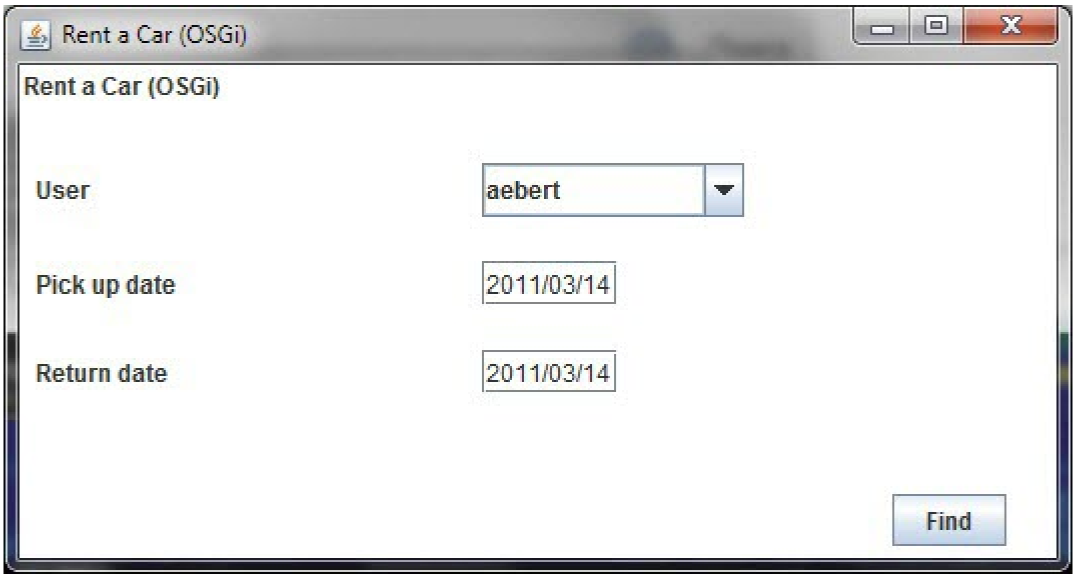
First we will look at the GUI interface, and then look at running the commands.

4.4.3. Using the "Rental Client GUI"

To start the GUI, type **`car:gui`** in the karaf alternate-container console window, that is **`karaf@trun>car:gui`**.

Select "aebert" from the drop-down box.

Figure 4.3. Rental Client GUI

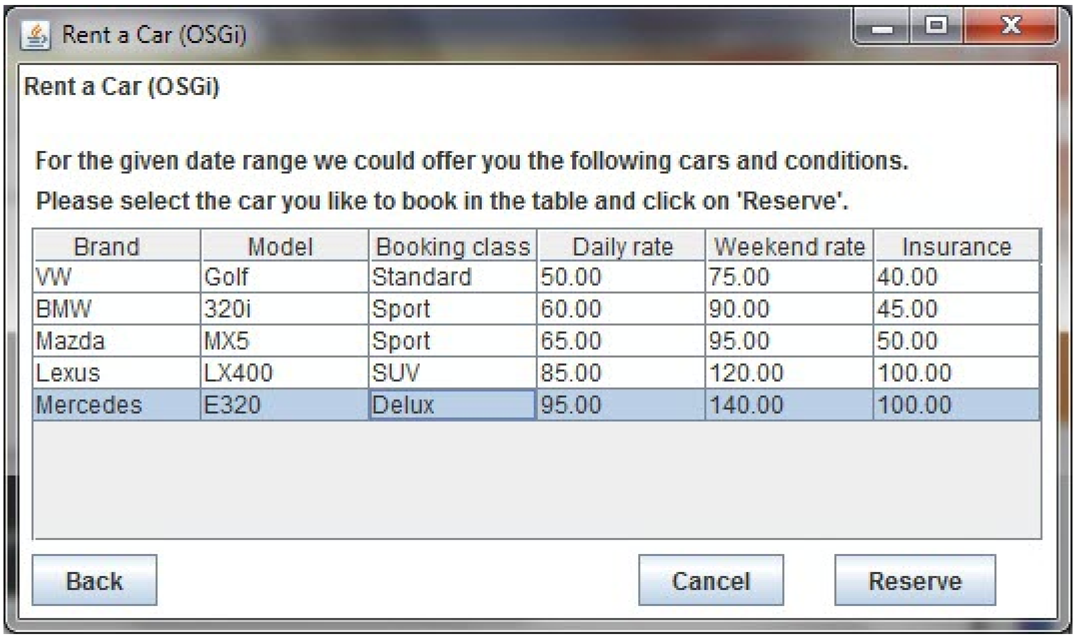

Rent a Car (OSGi)

Rent a Car (OSGi)

User	aebert
Pick up date	2011/03/14
Return date	2011/03/14

Click **Find** to see the results. You will see the list of available cars.

Figure 4.4. List of available cars


Rent a Car (OSGi)

Rent a Car (OSGi)

For the given date range we could offer you the following cars and conditions.

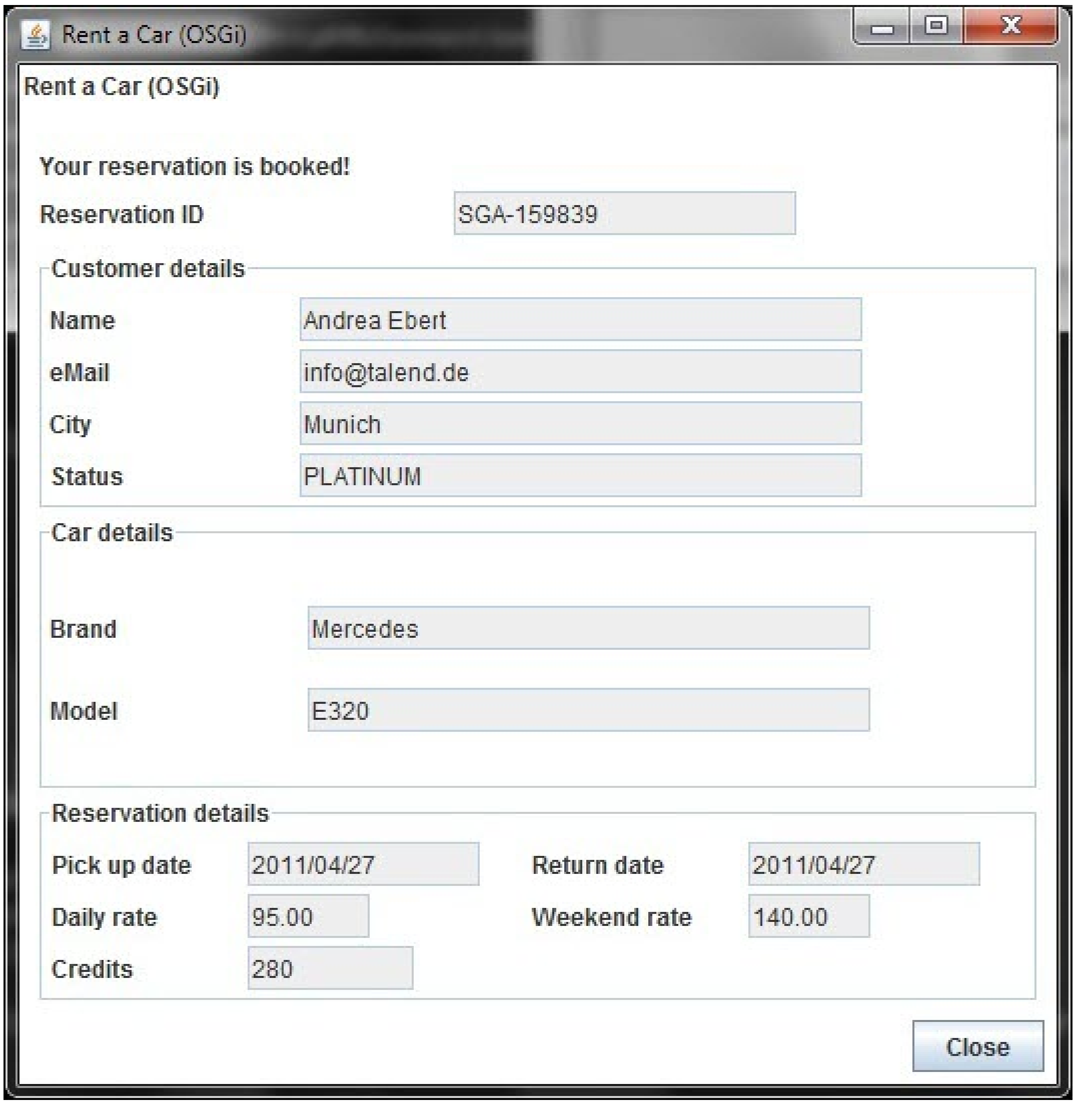
Please select the car you like to book in the table and click on 'Reserve'.

Brand	Model	Booking class	Daily rate	Weekend rate	Insurance
VW	Golf	Standard	50.00	75.00	40.00
BMW	320i	Sport	60.00	90.00	45.00
Mazda	MX5	Sport	65.00	95.00	50.00
Lexus	LX400	SUV	85.00	120.00	100.00
Mercedes	E320	Delux	95.00	140.00	100.00

Click the highlighted line or select any other option and click **Reserve**.

You will now see the confirmation page:

Figure 4.5. Confirmation of booking



Rent a Car (OSGi)

Rent a Car (OSGi)

Your reservation is booked!

Reservation ID	SGA-159839

Customer details

Name	Andrea Ebert
eMail	info@talend.de
City	Munich
Status	PLATINUM

Car details

Brand	Mercedes
Model	E320

Reservation details

Pick up date	2011/04/27	Return date	2011/04/27
Daily rate	95.00	Weekend rate	140.00
Credits	280		

Click **Close** to stop the application.

4.4.4. Using the command lines

4.4.4.1. Using the "search" command

To use the command line version of the client application, in the console type:

```
car:search aebert 2011/01/26 2011/01/26
```

You will see the list of available cars:

```
Found 5 cars.

Car details
```

```
1 VW Golf Standard 50.00 75.00 40.00
2 BMW 320i Sport 60.00 90.00 45.00
3 Mazda MX5 Sport 65.00 95.00 50.00
4 Lexus LX400 SUV 85.00 120.00 100.00
5 Mercedes E320 Delux 95.00 140.00 100.00
```

4.4.4.2. Using the "rent" command

Now, in the console type: **`car:rent 2`**

You can now see the confirmation:

```
Reservation ID SGA-686277

Customer details
----------------
Name: Andrea Ebert
eMail: info@talend.de
City: Munich
Status: PLATINUM

Car details
-----------
Brand: BMW
Model: 320i

Reservation details
-------------------
Pick up date: 2011/04/27
Return date: 2011/04/27
Daily rate: 60.00
Weekend rate: 90.00
Credits: 210
Thank you for renting a car with Talend :-)
```

4.5. Uninstalling the demo

This section describes uninstalling the demo, from the container, and from the alternate-container if you used one.

If you want to uninstall only the demo services, execute this command at the OSGi console:

`features:uninstall tesb-rac-services`

Then, you can install them again as described in Section 4.3.1, “*Deploying the demo services*”.

If you want to uninstall only the demo application, execute this command at the OSGi console:

`features:uninstall tesb-rac-app`

Then, you can install it again as described above.

If you want to uninstall all the demo features, for example, because you want to try other scenarios from this guide, execute following commands at the OSGi console (which includes the previous commands - not all of these will be installed in both containers if you are using two containers):

```
features:uninstall tesb-rac-services
```

```
features:uninstall tesb-rac-app
```

```
features:uninstall tesb-rac-common
```

Then enter **`features:list`** to ensure that all features have actually been uninstalled. You should see this in the console:

```
[uninstalled] [5.2.1    ] tesb-rac-common         repo-0
[uninstalled] [5.2.1    ] tesb-rac-services       repo-0
[uninstalled] [5.2.1    ] tesb-rac-app            repo-0
```

You can also remove Rent-a-Car feature location from the OSGi container(s).

At the OSGi console prompt, type the following command:

```
features:removeurl mvn:org.talend.esb.examples.rent-a-car/features/
5.2.1/xml
```

Chapter 5. Importing the demo into Eclipse

In this section, we'll use the Maven Eclipse Plugin (http://maven.apache.org/plugins/maven-eclipse-plugin/) to generate the Eclipse project files for the Rent-A-Car example. (The *Talend ESB Development Guide* describes how to install this plugin).

5.1. Building the project files

Go to the base directory of the Talend ESB example `<TalendRuntimePath>/examples/talend/tesb/rent-a-car`

If you haven't built the demo yet, run:

```
mvn clean install
```

Then run:

`mvn eclipse:eclipse`.

The Eclipse project files will now be created.

5.2. Importing into Eclipse

In the Eclipse IDE:

1. From the main menu bar, select File > Import.... The Import wizard opens.
2. Select General > Existing Projects into Workspace and click **Next**.

3. Choose Select root directory and click the associated **Browse** to locate the `examples/talend/tesb/rent-a-car` directory.

4. Back in the **Import Projects** window, under **Projects** select the project or projects that you would like to import, and then click **Finish**.

The Talend ESB example projects you selected will be imported into your Eclipse workspace. At this stage, you can now see the source code, modify it, and take advantage of all other Eclipse features.

5.3. Running the examples

Typically, the examples will still be run using Maven commands from a command-line window. If desired, Eclipse plugins for Maven exist to allow for running Maven commands within the IDE.

For more information on developing with Talend ESB and Eclipse, please see *Talend ESB Development Guide*.

Chapter 6. Enabling the Service Locator and Service Activity Monitoring for the demo

This chapter describes how to enable both of these for one Rent-a-Car demo:

- the Service Locator component, to publicly register the service endpoints
- the Service Activity Monitoring (SAM) to monitor service calls, typically for collecting usage statistics and fault monitoring

First we do a technical overview of both of these components, and then we do a step-by-step description of running the Rent-a-Car demo with this functionality enabled.

If you have Talend ESB, there is GUI functionality provided by the Talend Administration Center, for viewing the Service Locator and 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.

6.1. Technical overview of the Service Locator

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

For more detailed information on the Service Locator, beyond this manual, please see *Talend ESB Infrastructure Services Configuration Guide*.

The Service Locator consists of two parts:

1. The Service Locator *server* hosting an endpoint repository

2. the *CXF feature* used to enable usage of the locator for CXF service consumers and providers.

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

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

It is also possible to restrict access to the Service Locator (for example, to restrict updates permissions), please see the *Talend ESB Infrastructure Services Configuration Guide* for more details.

Note: the Service Locator server implementation is based on proven open source technology - Apache ZooKeeper. To learn more about Apache ZooKeeper, see http://zookeeper.apache.org.

6.2. Technical overview of Service Activity Monitoring

6.2.1. Service Activity Monitoring Agent and Server

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

For more detailed information on Service Activity Monitoring, beyond this manual, please see *Talend ESB Infrastructure Services Configuration Guide*.

Service Activity Monitoring consists of two parts:

- Agents (sam-agent) which gathers and sends monitoring data
- A Service Activity Monitoring Server which processes and stores the data

The sequence of how these are used is as follows:

1. The Agent creates events out of requests and replies from both the service consumer and provider side.

2. The events are first collected locally and then sent to the Service Activity Monitoring Server periodically (so as not to disturb the normal message flow).

3. When the Service Activity Monitoring Server receives events from the Agent, it optionally uses filters and/or handlers on those events and stores them into a database.

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

- The Agent is packaged as a JAR that needs to be on the classpath of the service consumer and provider.
- The Service Activity Monitoring Server is deployed as a WAR in a servlet container and needs access to a database.

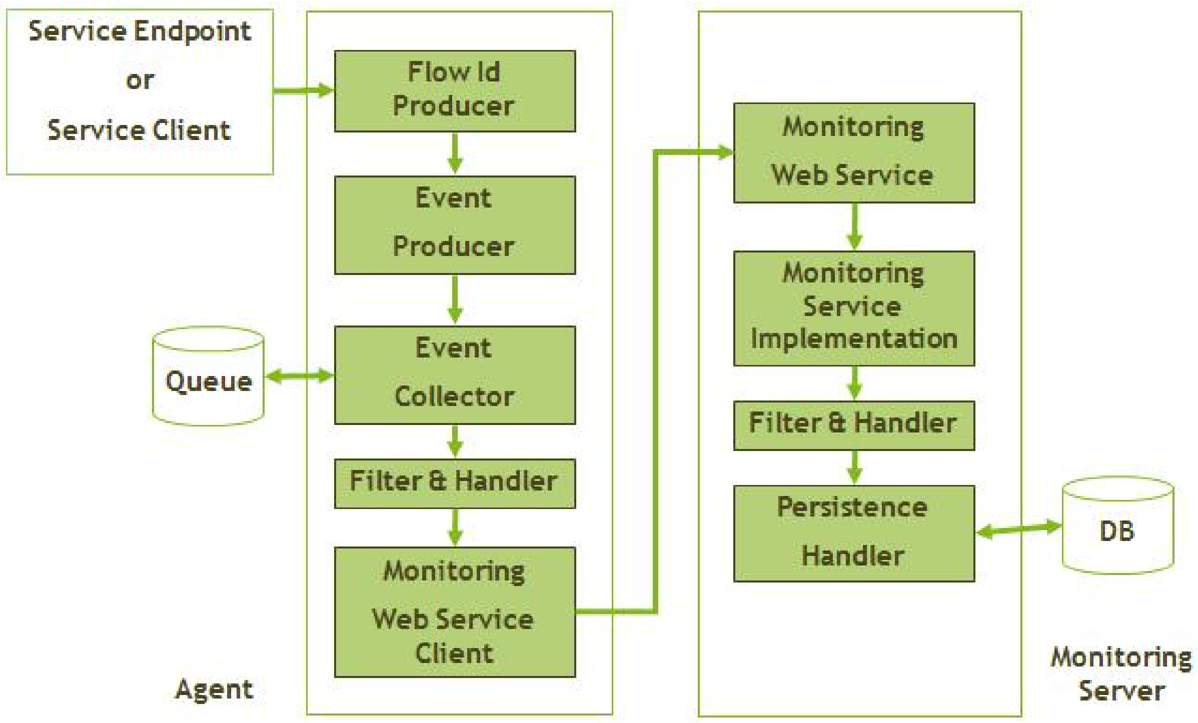
One service call can generate four events: For example: A consumer is sending a request (REQ_OUT), the service receives request (REQ_IN), the service sends response (RESP_OUT) and the consumer receives response (RESP_IN).

An Agent can be configured to collect all four events in this service call, on both the consumer and provider side. For further event processing all of these events will get the same "flow id".

6.2.2. Architecture of Service Activity Monitoring

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. The Server will receive events and store them into the database. A web service is used as the interface between the Agent and the Service Activity Monitoring Server.

Figure 6.1. Architecture of Service Activity Monitoring



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).

6.3. Overview of the demo

This section describes how to deploy the Rent-a-Car demo in the Talend Runtime container. This involves a number of steps, similar to those in the basic Rent-a-Car demo.

1. Build the demo - see Section 6.4, "*Building the demo with functionality enabled*"
2. Create an alternate OSGi container - see Section 2.7, "*Creating an alternate container*"
3. Start the default OSGi container - see Section 2.6, "*Starting a Talend Runtime container*"
4. Check that features from previous demo runs have been uninstalled - see Section 6.5, "*Uninstalling previous features*".
5. Deploy the Talend ESB services (see Section 6.6, "*Installing the Service Locator and Service Activity Monitoring* ").

   ⚠ It is important for the Rent-a-Car demo that these are installed in the default container (using port 8040), as the demo files are configured for that.
6. Deploy the demo services and client application in one container (see Section 6.8, "*Running the demo in one container*"). Here we concentrate on looking at Service Activity Monitoring functionality, since the Service Locator is largely unused in this scenario.
7. Then re-run the demo using two containers, while stopping one of the example services. This will show the role of the Service Locator in service failover (see Section 6.9, "*Running the demo in two containers - simulating server failover*"), and we will see client re-connect to another service, using this functionality.

6.4. Building the demo with functionality enabled

We use Maven with the combined Service Activity Monitoring and profile.

1. Go into the directory `<TalendRuntimePath>/examples/talend/tesb/rent-a-car`
2. The `pom.xml` file there is used by Maven to build the demo.
3. Run the Maven command on either Linux or Windows:

   ```
   mvn install -Pslsam
   ```

   Running this command will build the demo with the required functionality and create OSGi bundles to be deployed to an OSGi container.

Note: These changes occur when you build the examples with the Service Activity Monitoring Agent and the Service Locator enabled:

- The jaxws:features of the Service Activity Monitoring Agent and also the Service Locator will be added into Spring `beans.xml` file of the related client and service-endpoint, for example:

  ```
  <import resource="classpath:META-INF/tesb/locator/beans-osgi.xml" />
  <import resource="classpath:META-INF/tesb/agent-osgi.xml" />
     ...
  <jaxws:features>
      <bean class="org.talend.esb.servicelocator.cxf.LocatorFeature">
          <property name="selectionStrategy"
                  value="randomSelectionStrategy"/>
      </bean>
      <ref bean="eventFeature"/>
  </jaxws:features>
  ```

- The `MANIFEST.MF` file of related client and service-endpoint will be updated by adding Require-Bundle: sam-common,sam-agent,locator.

6.5. Uninstalling previous features

Start the default and alternate-containers if they're not already running.

If you have already used the container for running previous examples, you should ensure that all features from previous examples have been uninstalled including feature `tesb-rac-common` which is implicitly loaded as dependency. Otherwise the installation of the features for the locator example will fail. If in doubt about whether previous features are still installed, enter the following commands in both containers:

```
features:uninstall tesb-rac-services
```

```
features:uninstall tesb-rac-common
```

If you installed Rent-a-Car applicaton in this container you should uninstall it too:

```
features:uninstall tesb-rac-app
```

These commands uninstall the remaining bundles from the basic example.

You can also remove uninstalled features for the previous example from OSGi container:

```
features:removeurl   mvn:org.talend.esb.examples.rent-a-car/features/5.2.1/
xml
```

This command removes features for the simple scenario from the container.

6.6. Installing the Service Locator and Service Activity Monitoring

6.6.1. Installing and starting the Service Locator

In order to enable Service Locator for the Rent-a-Car example, the Service Locator has to be installed and deployed. This may be activated in two alternative ways:

- installed as a feature in container, such as the Talend Runtime container (see Section 6.6.1.1, “*Installing the Service Locator as a feature in an OSGi container*”). We use this option in the demo.
- deployed as WAR file in Tomcat or other Servlet container (see Section 6.6.1.3, “*Installing the Service Locator in a Servlet container*”).

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

6.6.1.1. Installing the Service Locator as a feature in an OSGi container

The OSGi version of the Service Locator can be started simply by entering the following at the *default* container console:

```
tesb:start-locator
```

6.6.1.2. Configuration parameters for the Service Locator in an OSGi container

Configuration parameters for the Service Locator are located in the file `<TalendRuntimePath>/container/etc/org.talend.esb.locator.cfg`. Here is an example:

```
# Configured zookeeper endpoints (divided by a comma if several instances
# uses). The service locator client will one by one pick an endpoint to
# connect to the service locator until a connection is established.
locator.endpoints=localhost:2181

# Endpoint prefix property is needed because we run services in
# a container where the endpoints is only relative to the container.
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
```

For further information about Service Locator properties configuration please read the *Talend ESB Infrastructure Services Configuration Guide*.

6.6.1.3. Installing the Service Locator in a Servlet container

You can also install and run the Service Locator as a standalone server.

1. To start the Service Locator you need to provide a new configuration file.

   Here are the settings for our example; create the new config file for a standalone Service Locator: `<TalendRuntimePath>/zookeeper/conf/zoo.cfg` with the following content:

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

   The `dataDir` value may need to be updated (it is the location to store the in-memory database snapshots - see *Talend ESB Infrastructure Services Configuration Guide* for more details).

2. Now, the Service Locator server can be started and stopped with the scripts from the `zookeeper/bin` directory.

   Under Linux, ensure execution rights for the Service Locator startup scripts:

   ```
   chmod a+x zookeeper/bin/*.sh
   ```

   From directory `<TalendRuntimePath>`, the Service Locator is started by the following command:

   Linux: **`./zookeeper/bin/zkServer.sh start`** in Windows command line: **`zookeeper\bin\zkServer.cmd start`**

3. It can be stopped by the following command:

   Linux: **`./zookeeper/bin/zkServer.sh stop`**

   Windows: **`zookeeper\bin\zkServer.cmd stop`**

For more information please read *Talend ESB Infrastructure Services Configuration Guide*.

6.6.1.4. Service Locator logging output

After enabling the Service Locator, we can see additional OSGi INFO logs (see output below from `<TalendRuntimePath>/container/log/tesb.log`) during the deployment of demo services and clients into OSGi container.

Initiating client connections produces logs from Zookeeper similar to those below:

```
18:41:37,843 | INFO  | ExtenderThread-8 | ZooKeeper |
org.apache.zookeeper.ZooKeeper    373 | 36 - org.apache.hadoop.zookeeper -
3.3.2 |
Initiating client connection, connectString=locator_host1:2181 sessionTimeo
ut=5000
watcher=org.talend.esb.locator.ServiceLocator$WatcherImpl@1e26d9b

18:41:37,843 | INFO  | d-8-SendThread() | ClientCnxn  |
org.zookeeper.ClientCnxn$SendThread 1041 | 36 - org.apache.hadoop.zookeeper
- 3.3.2 |
Opening socket connection to server locator_host1/192.168.150.201:2181

18:41:37,843 | INFO  | d(sop-td57:2181) | ClientCnxn  |
org.zookeeper.ClientCnxn$SendThread  949 | 36 -  org.apache.hadoop.zookeepe
r - 3.3.2 |
Socket connection established to locator_host1/192.168.150.201:2181,
initiating session

18:41:37,844 | INFO  | d(sop-td57:2181) | ClientCnxn  |
org.zookeeper.ClientCnxn$SendThread  738 | 36 -  org.apache.hadoop.zookeepe
r - 3.3.2 |
Session establishment complete on server locator_host1/192.168.150.201:2181
,sessionid = 0x32e71447e6b004c, negotiated timeout = 4000
```

Also, we can judge how well the feature is working by watching the Locator logs in the console window:

```
18:41:38,765 | INFO  | ExtenderThread-8 | ServiceLocator |
talend.esb.locator.ServiceLocator  177 |  - -  | Register endpoint
http://localhost:8888/soap/CRMServiceProvider/ for service
 {http://services.talend.org/CRMService}CRMServiceProvider.

18:44:07,250 | INFO  | xtenderThread-12 | ServiceLocator |
talend.esb.locator.ServiceLocator  177 |  - -  | Register endpoint
http://localhost:8888/soap/ReservationServiceProvider/ for service
 {http://services.talend.org/ReservationService}ReservationServiceProvider.
```

6.6.2. Installing and starting the SAM Server

In order to enable Service Activity Monitoring for the Rent-a-Car example, the Service Activity Monitoring Server has to be installed and deployed. This may be activated in two alternative ways:

- installed as a feature in container, such as the Talend Runtime container (see Section 6.6.2.1, "*Installing Service Activity Monitoring as a feature in an OSGi container*"). We use this option in the demo.
- deployed as WAR file in Tomcat or other Servlet container (see Section 6.6.2.3, "*Installing Service Activity Monitoring in a Servlet container*")

While you can have multiple instances of the Service Activity Monitoring Server running at a time, for clarity, we recommend you have one instance running for this demo.

If you want to use the same Service Activity Monitoring Server from multiple containers, please update the `service.url` property in file `<TalendRuntimePath>/container/etc/org.talend.esb.sam.agent.cfg` in each container. (This file is created in the container when the Service Activity Monitoring Agent is installed).

6.6.2.1. Installing Service Activity Monitoring as a feature in an OSGi container

The Service Activity Monitoring Server can be installed as a feature and run in the Talend Runtime container.

To start the Service Activity Monitoring Server, type in these console commands on the *default* Talend Runtime container:

```
tesb:start-sam
```

Then, the Service Activity Monitoring Server will be started. you can check with this URL in browser: http://localhost:8040/services/MonitoringServiceSOAP?wsdl,which should list its <wsdl:definitions>.

Service Activity Monitoring Server - Database re-creation activated by default

When a Service Activity Monitoring Server is installed in Talend Runtime, by default, `org.talend.esb.sam.server.cfg` has the database recreation activated; this shows errors in the console log for each subsequent restart of the container ("can't create table ..."). To avoid this startup logging error please change the `db.recreate` property in `org.talend.esb.sam.server.cfg` to false (db.recreate=false) after you install the Service Activity Monitoring Server.

tesb:start-sam will start Derby database on localhost with default 1527 port, and install tesb-datasource-derby feature and tesb-sam-server feature. If you are using another Database (for example MySQL, H2, Oracle), please follow the instructions of DataSource and Service Activity Monitoring Server installation/configuration in the *Talend ESB Infrastructure Services Configuration Guide*.

6.6.2.2. Configuring Service Activity Monitoring in an OSGi container

In a Talend Runtime container, if you want to configure some properties for Service Activity Monitoring access (for example `service.url`), please edit:

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

6.6.2.3. Installing Service Activity Monitoring in a Servlet container

If you don't wish to install Service Activity Monitoring as a feature in the OSGi container, there are a number of options and configurations for using Service Activity Monitoring with your application code - see the section on Service Activity Monitoring Installation in the *Talend ESB Infrastructure Services Configuration Guide* for full details and examples.

For example, in the Servlet container, if you want to configure some Service Activity Monitoring Agent properties (for example, service.url), please edit the `agent.properties` in your classpath:

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

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

service.url=http://localhost:8040/services/MonitoringServiceSOAP
service.retry.number=3
service.retry.delay=5000
```

Note that the `collector.scheduler.interval` parameter must be greater than 0.

6.7. Installing the Rent-a-Car examples features

6.7.1. Installing in default container

1. From the Talend Runtime container directory, type in console commands in the default container (please update the version number "5.2.1" if it differs in your installation):

   ```
       features:addurl mvn:org.talend.esb.examples.rent-a-car/features-sl-sam/5.2.1/xml
   ```

2. If previous examples have already been installed, do:

   ```
   features:refreshurl
   ```

3. Then enter the following commands:

   ```
       features:install tesb-rac-services-sl-sam
       features:install tesb-rac-app-sl-sam
   ```

4. Then, you can use the **`list`** command to check if everything installed properly:

   ```
   list
   ```

   For example:

   ```
   [ 172] [Active ] [         ] [ ] [ 80] Service Activity Monitoring ::
   Common (5.2.1)
   [ 173] [Active ] [         ] [ ] [ 80] Service Activity Monitoring ::
   ```

```
Agent (5.2.1)
[ 176] [Active ] [         ] [ ] [ 50] Service Locator Client for CXF
(5.2.1)
[ 186] [Active ] [Created  ] [ ] [ 80] Service Activity Monitoring ::
Datasource-derby (5.2.1)
[ 187] [Active ] [         ] [ ] [ 80] Apache ServiceMix :: Bundles ::
derbynet (10.8.1.2_1)
[ 188] [Active ] [         ] [ ] [ 80] Service Activity Monitoring ::
Derby Starter (5.2.1)
[ 194] [Active ] [         ] [ ] [ 80] Service Activity Monitoring ::
Server (5.2.1)
[ 195] [Active ] [         ] [ ] [ 80] CRMService Common (5.2.1)
[ 196] [Active ] [         ] [ ] [ 80] ReservationService Common
(5.2.1)
[ 197] [Active ] [         ] [ ] [ 80] CRMService Service (5.2.1)
[ 198] [Active ] [         ] [ ] [ 80] CRMService Service Endpoint
Locator and SAM (5.2.1)
[ 199] [Active ] [         ] [ ] [ 80] ReservationService Service
(5.2.1)
[ 200] [Active ] [         ] [ ] [ 80] ReservationService Service
Endpoint SL and SAM (5.2.1)
[ 201] [Active ] [         ] [ ] [ 80] CRMService Client Locator and
SAM (5.2.1)
[ 202] [Active ] [         ] [ ] [ 80] ReservationService Client
Locator and SAM (5.2.1)
[ 203] [Active ] [         ] [ ] [ 80] App Reservation (5.2.1)
```

6.7.2. Installing in alternate container

This is not used the first time we run the demo, so you can shutdown this container after installing the features if you wish.

1. Start the alternate container.

2. In the alternate container console, type in console commands:

   ```
   features:addurl  mvn:org.talend.esb.examples.rent-a-car/features-sl-sam/
   5.2.1/xml
   ```

3. If previous examples have already been installed, enter:

   ```
   features:refreshurl
   ```

4. Then enter:

   ```
   features:install tesb-rac-services-sl-sam
   ```

   Note: only install these features, not `tesb-rac-app-sl-sam`.

   Then, you can use the **`list`** command to check if everything installed properly:

   ```
   list
   ```

   For example:

   ```
   [152][Active][ ][ ][60] Service Activity Monitoring :: Common(5.2.1)
   [153][Active][ ][ ][60] Service Activity Monitoring :: Agent(5.2.1)
   [154][Active][ ][ ][60] CRMService Common (5.2.1)
   ```

```
[155][Active][ ][ ][60] ReservationService Common (5.2.1)
[156][Active][ ][ ][60] CRMService Client SAM (5.2.1)
[157][Active][ ][ ][60] ReservationService Client SAM(5.2.1)
[158][Active][Created][ ][60] App Reservation (5.2.1)
```

6.8. Running the demo in one container

Here we deploy the demo services and client application in one container (see Section 6.8, "*Running the demo in one container*"). We concentrate on looking at Service Activity Monitoring functionality, since the Service Locator is largely unused in this scenario.

Please stop the alternate-container if it's running, as we will use that in the next run of the demo.

Run the client application as in the basic Rent-a-Car demo (see Section 4.4, "*Running the demo*" for full details):

`car:gui`

Then click **Find**

This step is sufficent for the demo. You will then see output like this in the default container console:

```
###############################################
getCRMInformation() invoked ... request data:
org.talend.services.crm.types.LoginUserType@fc0795
###############################################
###############################################
getCRMInformation() invoked ... response data:
org.talend.services.crm.types.CustomerDetailsType@560854
###############################################
###############################################
getAvailableCars() invoked ... request data:
org.talend.services.reservation.types.RESProfileType@a14297
###############################################
###############################################
getAvailableCars() invoked ... response data:
org.talend.services.reservation.types.RESCarListType@11ad529
###############################################
```

which shows the requests and responses for the `CRMService` and `ReservationService`.

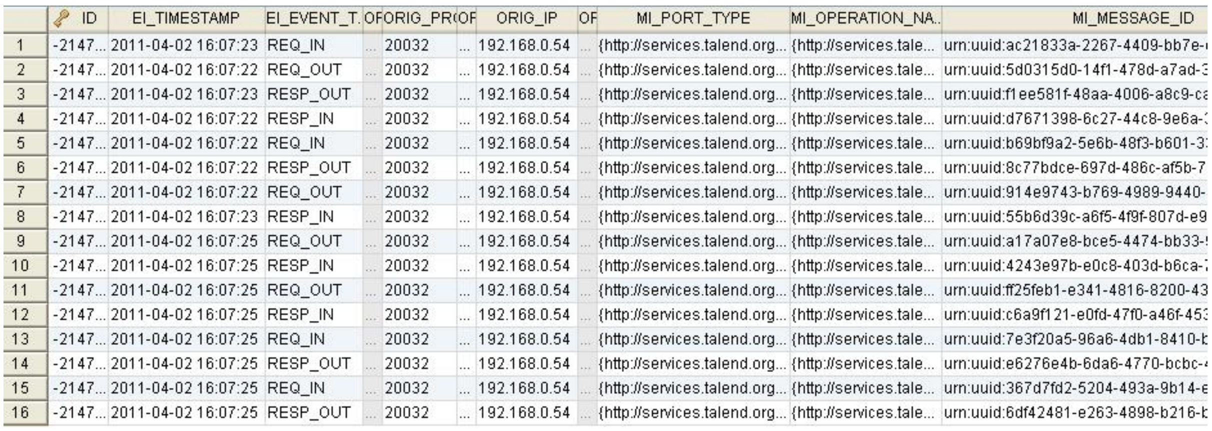
6.8.1. Viewing the events in the Service Activity Monitoring database

When the Rent-a-Car demo is running, you can monitor the EVENTS table and EVENTS_CUSTOMINFO table using DbVisualizer (or other DB tools), using the following settings:

```
Database connection configuration (Default): DB
Driver: org.apache.derby.jdbc.ClientDriver
DB URL: jdbc:derby://localhost:1527/db
DB username: test
DB password: test
```

The DB connection which is configured in DbVisualizer must correspond to the settings in <TalendRuntimePath>/add-ons/sam/sam-server-war/WEB-INF/logserver.properties.

Then, the data of EVENTS table and EVENTS_CUSTOMINFO table can be browsed/monitored in DbVisualizer. For example, the data of EVENTS table will be looked like:



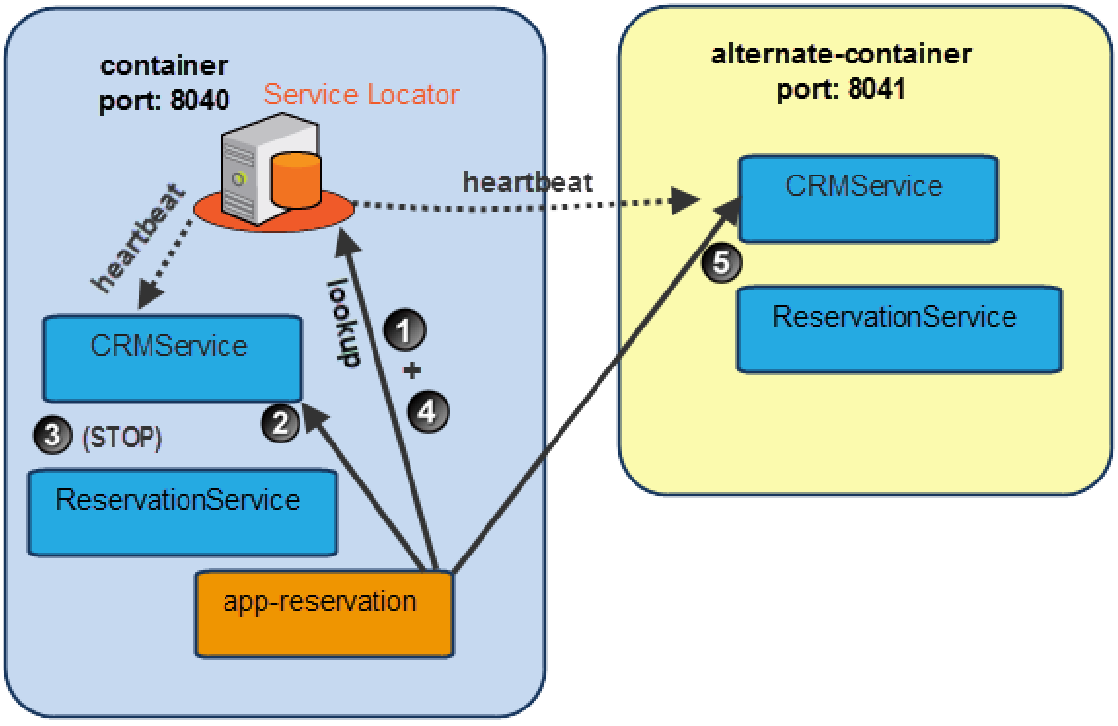
	ID	EI_TIMESTAMP	EI_EVENT_T.	OF	ORIG_PR(	OF	ORIG_IP	OF	MI_PORT_TYPE	MI_OPERATION_NA..	MI_MESSAGE_ID
1	-2147...	2011-04-02 16:07:23	REQ_IN	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:ac21833a-2267-4409-bb7e-
2	-2147...	2011-04-02 16:07:22	REQ_OUT	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:5d0315d0-14f1-478d-a7ad-
3	-2147...	2011-04-02 16:07:23	RESP_OUT	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:f1ee581f-48aa-4006-a8c9-ca
4	-2147...	2011-04-02 16:07:22	RESP_IN	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:d7671398-6c27-44c8-9e6a-
5	-2147...	2011-04-02 16:07:22	REQ_IN	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:b69bf9a2-5e6b-48f3-b601-3
6	-2147...	2011-04-02 16:07:22	RESP_OUT	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:8c77bdce-697d-486c-af5b-7
7	-2147...	2011-04-02 16:07:22	REQ_OUT	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:914e9743-b769-4989-9440-
8	-2147...	2011-04-02 16:07:23	RESP_IN	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:55b6d39c-a6f5-4f9f-807d-e9
9	-2147...	2011-04-02 16:07:25	REQ_OUT	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:a17a07e8-bce5-4474-bb33-
10	-2147...	2011-04-02 16:07:25	RESP_IN	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:4243e97b-e0c8-403d-b6ca-
11	-2147...	2011-04-02 16:07:25	REQ_OUT	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:ff25feb1-e341-4816-8200-43
12	-2147...	2011-04-02 16:07:25	RESP_IN	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:c6a9f121-e0fd-47f0-a46f-45
13	-2147...	2011-04-02 16:07:25	REQ_IN	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:7e3f20a5-96a6-4db1-8410-
14	-2147...	2011-04-02 16:07:25	RESP_OUT	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:e6276e4b-6da6-4770-bcbc-
15	-2147...	2011-04-02 16:07:25	REQ_IN	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:367d7fd2-5204-493a-9b14-e
16	-2147...	2011-04-02 16:07:25	RESP_OUT	...	20032	...	192.168.0.54	...	{http://services.talend.org...	{http://services.tale...	urn:uuid:6df42481-e263-4898-b216-

For more information about the table's schema definition, please look in Appendix A: Event Structure and Appendix B: EVENTS_CUSTOMINFO Structure in *Talend ESB Infrastructure Services Configuration Guide*.

6.9. Running the demo in two containers - simulating server failover

Now we re-run the demo using two containers, while stopping one of the example services. This will show the role of the Service Locator in service failover (see Section 6.9, "*Running the demo in two containers - simulating server failover*"), and we will see client re-connect to another service, using this functionality.

Figure 6.2. Demonstrating server failover



There are a number of steps to this:

1. The app-reservation looks up the CRMService endpoint in the Service Locator.
2. It then connects directly to the CRMService in the default container.
3. Then we stop the CRMService
4. The app-reservation automatically looks up the CRMService endpoint again in the Service Locator.
5. It then connects directly to the CRMService in the alternate-container.

6.9.1. Run the demo

- Run the demo application again in default container console, and type the command **`car:gui`**. While running the application, you will see the same request and responses as before.

  Note: Do not close the GUI application, just click **`Back`** in the GUI.

6.9.1.1. Simulate service failure

Now we simulate service failure

1. Start the alternate container.
2. In the default container, get the details of the CRMService bundle by typing the command **`list`** and note the number of the CRMService Service Endpoint bundle:

   ```
   [ 142] [Active ] [ ] [ ] [ 60] CRMService Service Endpoint Locator and
   SAM (5.2.1)
   ```

   (The number, here "142", will probably be different in your console).
3. Stop it with the command **`stop 142`** (using the bundle number you have found).
4. Then, click "Find" in the application GUI to activate the call to the CRMService.

In the first console you will see the expected warning message:

```
2:30:55,049 | WARN | AWT-EventQueue-0 | ache.cxf.common.logging.LogUtils 36
9 | - - | Interceptor for {http://services.talend.org/CRMService}CRMService
Provider#{http://services.talend.org/CRMService}getCRMInformation has throw
n exception, unwinding now org.apache.cxf.interceptor.Fault: Could not send
Message.
```

The warning is only shown if the log level is set appropriately. But even if the warning doesn't appear you still can verify from the output which container recieves the service call. The log level can be adjusted in `container/etc/java.util.logging.properties`.

Afterwards, the Locator will switch to the CRMService in the second container; and in the alternate container console you will see something like:

```
###############################################
getCRMInformation() invoked ... request data:
org.talend.services.crm.types.LoginUserType@1142653
###############################################
###############################################
getCRMInformation() invoked ... response data:
org.talend.services.crm.types.CustomerDetailsType@1e03fce
###############################################
```

In first console:

```
###############################################
getAvailableCars() invoked ... request data:
org.talend.services.reservation.types.RESProfileType@1d510be
###############################################
###############################################
getAvailableCars() invoked ... response data:
org.talend.services.reservation.types.RESCarListType@13e49a8
###############################################
```

Now we can see the ReservationService working in the first container, and the CRMService in the second container.

6.10. Uninstalling the demo features

In order to clean up both OSGi containers for the any subsequent running of examples, all features of the locator example need to be uninstalled.

1. In the default container, enter the following commands at the console:

   ```
   features:uninstall tesb-rac-services-sl-sam
   features:uninstall tesb-rac-common-sl-sam
   features:uninstall tesb-rac-app-sl-sam
   ```

2. In the alternate container, enter the following commands at the console:

   ```
   features:uninstall tesb-rac-services-sl-sam
   features:uninstall tesb-rac-common-sl-sam
   ```

3. Enter **`features:list`** to ensure that all features have actually been uninstalled.

   From Talend Runtime container, you will see:

   ```
   [uninstalled] [5.2.1] tesb-rac-common-sl-sam repo-0
   [uninstalled] [5.2.1] tesb-rac-services-sl-sam repo-0
   [uninstalled] [5.2.1] tesb-rac-app-sl-sam repo-0
   ```

   From alternate container, you will see:

   ```
   [uninstalled] [5.2.1] tesb-rac-common-sl-sam repo-0
   ```

6.11. Manual configuration issues

6.11.1. Service Locator Spring configuration

This section describes configuration steps that need to be done manually to get the locator feature enabled for CXF services and clients. Let's look at the Spring configuration of the basic demo example and consider the necessary changes needed to enable locator features.

To enable the Locator feature, import locator beans in the Spring configuration files:

`<import resource="classpath:tesb/locator/beans.xml" />` for the servlet container ,

`<import resource="classpath:tesb/locator/beans-osgi.xml" />` for the OSGi container.

We should add the Service Locator feature to the Spring configurations represented by the `beans.xml` files. The Service Locator feature should be added to each endpoint and client in the example configuration.

6.11.1.1. Service provider side configuration

Add the Service Locator feature to the endpoint bean in the `beans.xml` in exactly the same way as a standard CXF feature.

You can see these changes in the following files:

`<TalendRuntimePath>/examples/talend/tesb/rent-a-car/reservationservice/service-endpoint-sl-sam/src/main/resources/META-INF/spring/beans.xml`

`<TalendRuntimePath>/examples/talend/tesb/rent-a-car/crmservice/service-endpoint-sl-sam/src/main/resources/META-INF/spring/beans.xml`

For example:

```
<import resource="classpath:META-INF/tesb/locator/beans-osgi.xml" />
...
<jaxws:endpoint id="CRMService"
    ...
    <jaxws:features>
        <bean class="org.talend.esb.servicelocator.cxf.LocatorFeature"/">
    </jaxws:features>
</jaxws:endpoint>
```

For more information about Service Locator properties please read the *Talend ESB Infrastructure Services Configuration Guide*.

6.11.1.2. Client side configuration

Add the Service Locator feature to the client bean in the `beans.xml` file in exactly the same way as in a service bean.

You can see these changes in the following files:

```
<TalendRuntimePath>/examples/talend/tesb/rent-a-car/reservationservice/
client-sl-sam/src/main/resources/META-INF/spring/beans.xml
<TalendRuntimePath>/examples/talend/tesb/rent-a-car/crmservice/client-sl-
sam/src/main/resources/META-INF/spring/beans.xml
```

Another important point is to configure the JAX-WS client address. The locator protocol should be used in client endpoint configuration:

```
address="locator://more_useful_information"
```

For more information about Service Locator properties please read the *Talend ESB Infrastructure Services Configuration Guide*.

Chapter 7. Enabling WS-Security for the demo

This chapter describes building and running the Rent-a-Car demo with WS-security enabled.

7.1. Technical overview of the Security Token Service (STS)

For more information about the usage of Security Token Service, please read the *Talend ESB STS User Guide* and the STS chapter in *Talend ESB Infrastructure Services Configuration Guide*.

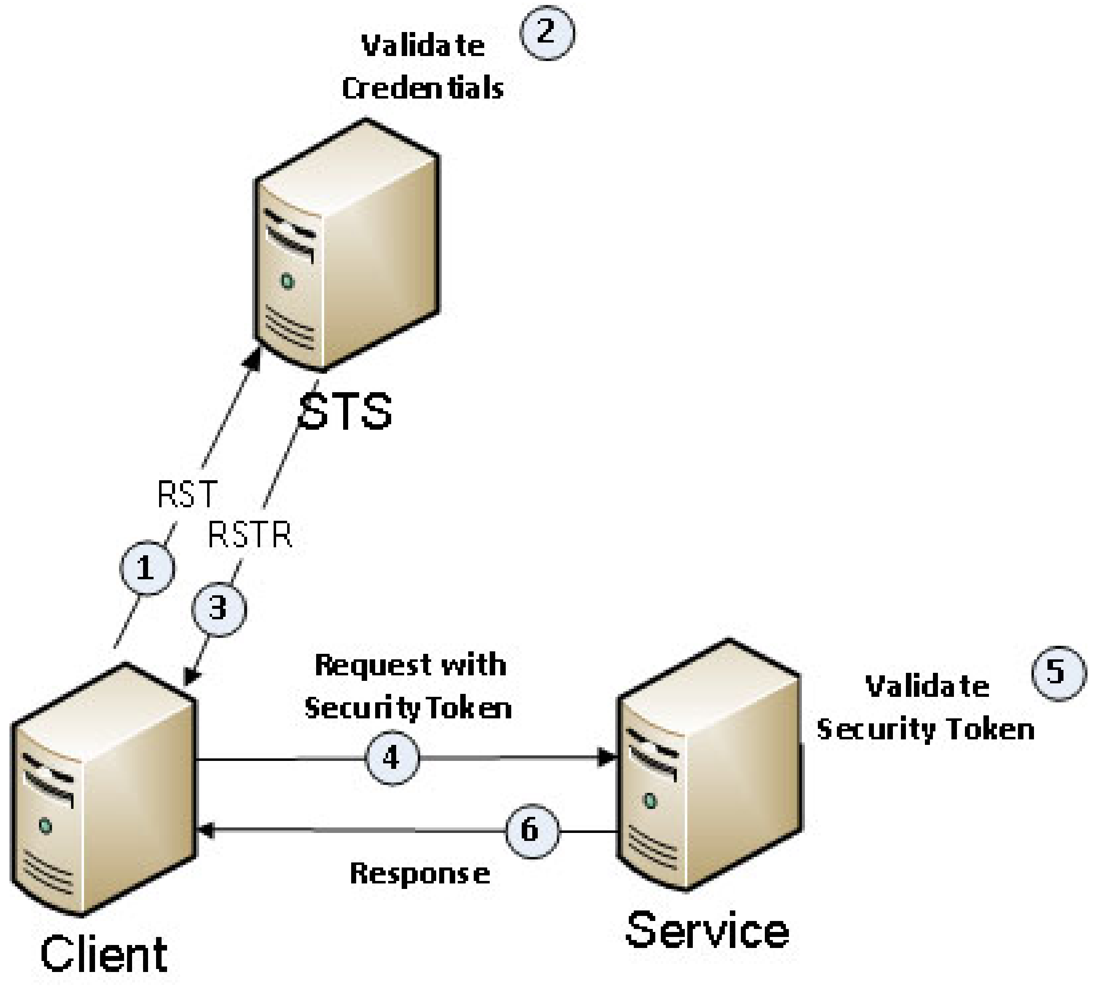
In a heterogeneous environment, Web services need to authenticate service clients to control their access by using WS-Security (Web Services security). When negotiating trust between service clients and service providers, an authentication broker can provide a common access control infrastructure for a group of applications. Typically, the authentication broker issues signed security tokens which are used by clients to authenticate themselves at the service.

The Security Token Service is a service for providing such an authentication broker. It issues Security Tokens based on the WS-Trust, a standardized specification of Web services based on WS-Security.

This is useful, for example, to establish a trust relationship between a client and a web service, particularly if they are in different security domains. The Security Token Service is used to issue a security token, that is, a collection of claims such as name, role, and authorization code, for the client to access the service. The message receiver only must know the STS certificate for verifying the token signature to get a trusted statement of the authentication information.

7.1.1. A sample Security Token Service scenario

This section describes a typical interaction with the Security Token Service.



1. The client sends an authentication request to the Security Token Service (Request Security Token - RST message).
2. The Security Token Service validates the client's credentials.
3. The Security Token Service issues a security token to the client (Request Security Token Response - RSTR message). The RSTR contains a security token, such as an XML Security Assertion Markup Language (SAML) token.
4. The client initializes and sends a request message, containing the token, to the Service.
5. The Service attempts to verify that the security token was issued by a trusted Security Token Service by checking the corresponding STS certificate. On success accepts it (essentially as equivalent to a "valid login"), and processes the request.
6. The service initializes and sends a response message to the client.

The Security Assertion Markup Language (SAML) tokens provide cross-platform interoperability and exchange security information between clients and services in different security domains. The receiver of the message with the token only needs to know the corresponding STS certificate in order to verify the token and able to use the authentication information from the token.

7.2. Installation and settings

In order to enable WS-security for the Rent-a-Car example, the Security Token Service server has to be installed and deployed. A sample implementation of the Security Token Service server is available with the Talend software; this may be activated in two alternative ways:

- installed as a feature in container, such as the Talend Runtime container (see Section 7.3, *“Building the demo with WS-Security enabled”*)
- deployed as WAR file in Tomcat or other Servlet container (see *Talend ESB STS User Guide*)

To prevent the "Illegal key size or default parameters" exception, please check that the Java(TM) Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6 is installed. Otherwise, download it, and replace `$JAVA_HOME/jre/lib/security/local_policy.jar` and `US_export_policy.jar` with new the jars to use 256-bit keys encryption in the STS sample.

The sample keys distributed with the Rent-a-Car demo should NOT be used in production. For more information on how to replace the keys used, see the STS chapter in *Talend ESB Infrastructure Services Configuration Guide*.

7.2.1. Required containers

1. Ensure the default port for the OSGi HTTP Service is configured in both containers below. For example, in the Talend Runtime container, the following property should be set in the `./container/etc/org.ops4j.pax.web.cfg` file:

   `org.osgi.service.http.port=8040`

2. We will be using the default container, as described in Section 2.6, *“Starting a Talend Runtime container”*, but don't start it for now.
3. Create an alternate container, if needed, as described in Section 2.7, *“Creating an alternate container”*.
4. Configure logging to override the default, if you need it, as described in the next section.
5. Finally, start the container (and the alternate container if neccessary).

7.2.2. Configuring logging

The STS functionality is enabled in the standard Rent-a-Car in `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/`.

For the Rent-a-Car Security Token Service (STS) sample, automatic logging is enabled by default in the client and service configuration files.

To disable automatic logging in the sample, remove the following XML node from `beans.xml` in both services and clients of the Rent-a-Car Security Token Service sample:

```
<cxf:bus>
    <cxf:features>
        <cxf:logging>
    <cxf:features>
<cxf:bus>
```

Log messages from clients and Security Token Service are stored in the alternate container:

```
./alternate-container/log/tesb.log
```

Log messages from services are stored in the same directory in the primary container.

7.3. Building the demo with WS-Security enabled

1. **Build the STS feature, and demo:** Run the following command from directory `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/`:

   ```
   mvn install -Psts
   ```

2. Install the STS feature in a container in the default Talend Runtime container, by executing the following command at the console:

   ```
   tesb:start-sts
   ```

   The STS service will start automatically. To make sure that it is running, execute the following command in the console:

   ```
   list
   ```

   and find two additional bundles: Apache CXF STS Core and Talend :: ESB :: STS :: CONFIG which enable the STS functionality.

3. Now, install the examples themselves:

4. In the default container console, type:

   ```
   features:addurl    mvn:org.talend.esb.examples.rent-a-car/features-sts/
   5.2.1/xml
   ```

   When the STS example has finished being installed, and the cursor returns, type:

   ```
   features:refreshurl
   features:install tesb-rac-services-sts
   ```

   Again, wait for the cursor to return.

5. In the alternate container Talend Runtime container console, type in:

   ```
   features:addurl    mvn:org.talend.esb.examples.rent-a-car/features-sts/
   5.2.1/xml
   ```

   When the examples have been installed, and the cursor returns, type:

   ```
   features:refreshurl
   features:install tesb-rac-app-sts
   ```

   Again, wait for the cursor to return.

   Now, the STS sample is installed and ready to run.

6. Please run the Rent-a-Car client as normal (for example, by running `car:gui` in the default container - see Chapter 4, *A real-world Rent-a-Car demo example*). You can then observe the interactions with the Security Token Service.

7.4. Uninstalling the demo features

1. When you are finished running the demo, uninstall the STS features:

   In the alternate container, type in:

   `features:uninstall tesb-rac-app-sts`

   `features:uninstall tesb-rac-common-sts`

2. In the primary container, type in:

   `features:uninstall tesb-rac-services-sts`

   `features:uninstall tesb-rac-common-sts`

7.5. Manual configuration of the STS based Authentication in the demo

This section describes the Rent-a-Car STS sample configuration and the steps to do manually to get the STS feature enabled for CXF services and clients.

The sample STS client and service are configured using CXF.

For more information about client and service configuration, read: http://cxf.apache.org/docs/ws-trust.html and *Talend ESB STS User Guide*.

To enable STS in the client configuration:

Add the STS server client into the custom service JAX-WS client configuration, as in the following sample:

```
<jaxws:client id="CRMServiceClient"
 name="{http://services.talend.org/CRMService}CRMServiceProvider"
 xmlns:serviceNamespace="http://services.talend.org/CRMService"
 serviceClass="org.talend.services.crmservice.CRMService"
 serviceName="serviceNamespace:CRMServiceProvider"
 endpointName="serviceNamespace:CRMServiceProvider"
 wsdlLocation="classpath:/model/crmservice-wsdls/CRMService-sts.wsdl">
  <jaxws:properties>
  .........
    <entry key="ws-security.sts.client">
        <bean class="org.apache.cxf.ws.security.trust.STSClient">
         <constructor-arg ref="cxf" />
         <property name="wsdlLocation"
             value="http://localhost:8040/SecurityTokenService/UT?wsdl" />
         <property name="serviceName"
             value="{http://docs.oasis-open.org/ws-sx/ws-trust/200512/wsdl}
             SecurityTokenServiceProvider" />
         <property name="endpointName"
             value="{http://docs.oasis-open.org/ws-sx/ws-trust/200512/wsdl}
             SecurityTokenServiceSOAP" />
         <property name="properties">
            <map>
  ........
```

```
            </map>
        </property>
        </bean>
       </entry>
    </jaxws:properties>
</jaxws:client>
```

That listing demonstrates the sample JAX-WS client with:

- enabled STS client bean (<entry key="ws-security.sts.client">)
- The STS client includes wsdlLocation for the STS service(property name="wsdlLocation")
- STS service name (property name="serviceName")
- STS service endpoint name (property name="endpointName")
- the list of additional STS client parameters (property name="properties") for example certificate settings

The STS service WSDL can be accessed from the running STS service via http://localhost:8040/SecurityTokenService/UT?wsdl.

You can see these changes in the following files:

- The CRM client is implemented in the `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/crmservice/client-sts` folder.
- The CRM client configuration is located in the: `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/crmservice/client-sts/src/main/filtered-resources/META-INF/spring/beans.xml` file.
- The Reservation client is implemented in the `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/reservationservice/client-sts` folder.
- The Reservation client configuration is located in the: `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/reservationservice/client-sts/src/main/filtered-resources/META-INF/spring/beans.xml` file.
- The CRM service is implemented in the `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/crmservice/service-endpoint-sts`.
- The service configuration is located in the: `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/crmservice/service-endpoint-sts/src/main/resources/META-INF/spring/beans.xml` file.
- The Reservation service is implemented in the `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/reservationservice/service-endpoint-sts`.
- The service configuration is located in the: `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/reservationservice/service-endpoint-sts/src/main/resources/META-INF/spring/beans.xml` file.

7.5.1. Policies and WSDL

WS-SecureConversation can in theory be used with multi-party message exchanges, but the most common usage is for a client communicating with a single server. When used in this configuration, the STS that supplies the SCT to the client is co-located with the server, accessed at the same endpoint address. This means that the web services code on the server needs a way to distinguish between messages intended for the STS and those intended for the service itself; the action used on the request serves that purpose.

A major part of the WS-Security performance cost comes from the wide use of asymmetric encryption. Asymmetric encryption is a useful tool because it works with key pairs. Each key in the pair can be used to encrypt messages that the other key can decrypt. The owner of a key pair can make one key publicly available so that anyone can use it to encrypt messages to the owner securely, and also decrypt messages from the owner (thereby verifying the sender's identity).

WS-SecureConversation uses WS-Policy and WS-SecurityPolicy configurations that are similar to those used by the basic WS-Security. A big difference is that when WS-SecureConversation is used, the policy must cover two separate exchanges — that between the client and the STS, and that between the client and the actual service. This is handled in the policy description by using a nested policy for the exchange with the STS, while the main body of the policy applies to the exchange between the client and the service.

Policy configuration of Rent-a-Car STS sample is represented in two different style as included in the WSDL of service and as external policy attachment.

The policy attachment technique gives the ability to store the policies in an external attachment without touching the WSDL of the service. The policy attachment usage is represented in CRMservice and files:

Endpoint configuration is represented in: `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/crmservice/client-sts/src/main/filtered-resources/META-INF/spring/beans.xml`

External policy attachment is represented in: `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/crmservice/client-sts/src/main/resources/saml.policy`

For additional information about policy attachment, read: http://www.w3.org/Submission/WS-PolicyAttachment/ and http://cxf.apache.org/docs/how-it-works.html

The other way to store policy is policy inclusion into WSDL, and it is represented in ReservationService WSDL file:

`<TalendRuntimePath>/examples/talend/tesb/rent-a-car/reservationservice/common/src/main/resources/model/ReservationService-sts.wsdl`

For additional information about policies using in CXF, please read: http://cxf.apache.org/docs/ws-securitypolicy.html and http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html.

7.5.2. Keystores and models for the client and service

Note that the keys here are not meant for production use, they are self-signed and naturally should have passwords different from those finally used.

The sample client keystore and configuration for the CRM service is located in `crmservice/client-sts/src/main/resources`. It contains two files: `clientstore.jks` and `clientKeystore.properties`.

The sample client keystore and configuration for the Reservation service is located in `reservationservice/client-sts/src/main/resources`. It contains two files: `clientstore.jks` and `clientKeystore.properties`.

The sample service keystore and configuration for the CRM service is located in `crmservice/service-endpoint-sts/src/main/resources`. It contains two files: `servicestore.jks` and `serviceKeystore.properties`.

The sample service keystore and configuration for the Reservation service is located in `reservationservice/service-endpoint-sts/src/main/resources`. It contains two files: `servicestore.jks` and `serviceKeystore.properties`.

For additional information for keystore creation and usage read Java keytool documentation and OpenSSL tutorial: http://www.openssl.org/ and the *Talend ESB STS User Guide*.

Chapter 8. Enabling the Talend Artifact Repository for the demo

This chapter describes adding the use of repository management software to the demo, including the Talend Artifact Repository.

8.1. 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 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 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.

8.2. More information

For information on Apache Archiva, see *Talend ESB Infrastructure Services Configuration Guide* and http://archiva.apache.org/.

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

For information on the Talend Artifact Repository, see *Talend Enterprise ESB Installation Guide*.

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

8.3. Chapter structure

This chapter describes the Talend Artifact Repository and how to enable it for the demo (see Figure 1.3, “Talend Enterprise and Platform operating principles” for an overview of how the Talend Artifact Repository interacts with other components).

- We can publish routes and services designed in regular Eclipse to Archiva, using Maven (see Section 8.5, “*Uploading the demo to a local Archiva repository using Maven files*”), both in Talend ESB Standard Edition and Talend ESB.
- In Talend ESB Standard Edition, we can deploy code manually into the Talend Runtime with a Maven Repository Manager, such as Sonatype Nexus or Apache Archiva (see Section 8.6, “*Deploying the demo manually in Talend ESB Standard Edition with Nexus*”).
- In Talend ESB, we can additionally publish Data Services and Mediation Routes directly to the Talend Artifact Repository, from ESB or Data Services Studio, using a menu option (Section 8.8, “*Deploying the demo with Talend Administration Center to Talend Runtime*”).
- When interacting with the Talend Artifact Repository, the Talend Administration Center can deploy a feature to the Talend Runtime regardless of how it was published to the repository (Section 8.8, “*Deploying the demo with Talend Administration Center to Talend Runtime*”).
- The examples work with both Archiva and Talend Artifact Repository; however in terms of configuration information, the examples shown are related to the Talend Artifact Repository (for example: URL, Repository names, username and password).

8.4. Downloading and installing Archiva

8.4.1. 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/.

8.4.2. Talend ESB

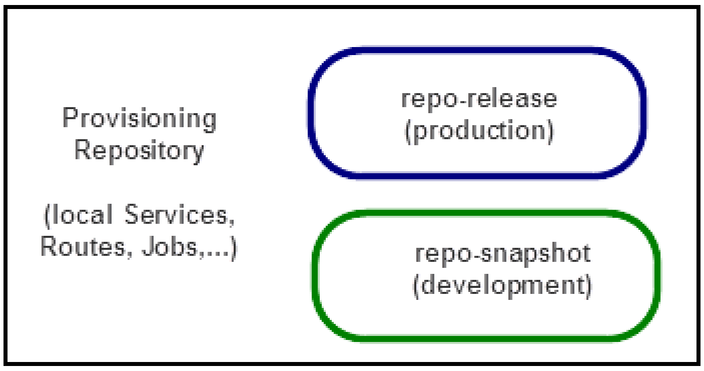
If you are using Talend ESB, then see Section 8.7, “*Preparing Talend Administration Center to deploy the demo to Talend Runtime*” for details on how to install the Talend Artifact Repository (based on Achiva).

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

8.5. Uploading the demo to a local Archiva repository using Maven files

The Maven project files can be used to deploy Rent-a-Car example into an Archiva repository.

Figure 8.1. Talend Artifact Repository provisioning repositories


Provisioning Repository
(local Services, Routes, Jobs,...)
repo-release (production)
repo-snapshot (development)

1. In both your Maven `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/settings.xml` file and your `$user.dir/.m2/settings.xml` file add the section:

```
<servers>
        ....
    <server>
        <id>archiva.repo-release </id>
        <username>tadmin </username>
        <password>tadmin </password>
    </server>
    <server>
        <id>archiva.repo-snapshot </id>
        <username>tadmin </username>
        <password>tadmin </password>
    </server>
          ....
</servers>
```

2. Also check that the `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/pom.xml` file contains sections with the correct Archiva link:

```
<distributionManagement>
  <repository>
    <id>archiva.repo-release</id>
    <name>Internal Release Repository</name>
    <url>
      dav:http://localhost:8082/archiva/repository/repo-release
    </url>
  </repository>

  <snapshotRepository>
    <id>archiva.repo-snapshot</id>
```

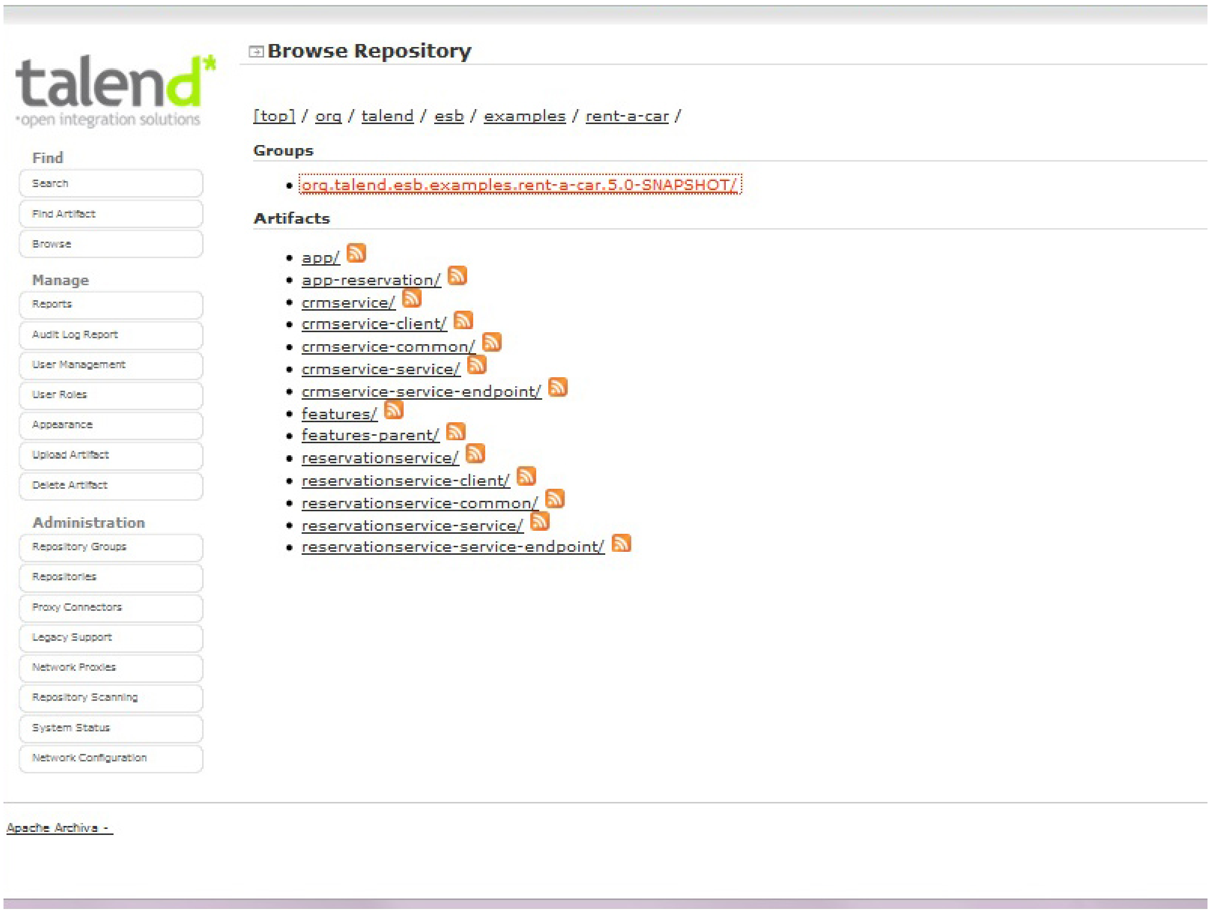
```
      <name>Internal Snapshot Repository</name>
      <url>dav:http://localhost:8082/archiva/repository/repo-snapshot
          </url>
    </snapshotRepository>
  </distributionManagement>
```

3. Using Apache Maven with either Linux or Windows, run the following command from the directory `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/`:

   `mvn deploy`

   Note: this step assumes that you have built the demo previously. If not, then please run **`mvn install`** first. See Section 4.1, *"Overview of Rent-a-Car demo example"* for the beginning of instructions on running the demo.

4. Check on the link **`http://localhost:8082/archiva/browse`**, that the Rent-a-Car sample: **`org.talend.esb.examples.rent-a-car/`** is successfully deployed.



Browse Repository

[top] / org / talend / esb / examples / rent-a-car /

Groups

- org.talend.esb.examples.rent-a-car.5.0-SNAPSHOT/

Artifacts

- app/
- app-reservation/
- crmservice/
- crmservice-client/
- crmservice-common/
- crmservice-service/
- crmservice-service-endpoint/
- features/
- features-parent/
- reservationservice/
- reservationservice-client/
- reservationservice-common/
- reservationservice-service/
- reservationservice-service-endpoint/

8.6. Deploying the demo manually in Talend ESB Standard Edition with Nexus

It is possible to deploy the demo manually in Talend ESB Standard Edition with a Maven Repository Manager. Here we use Sonatype Nexus, but you can use Apache Archiva to deploy if you use the pom definitions as shown in Section 8.5, *"Uploading the demo to a local Archiva repository using Maven files"*. Nexus acts as a sort of shared server of Maven artifacts repositories. Download and learn more about Nexus at:

http://nexus.sonatype.org/download-nexus.html

For more information on the syntax below, please see *Talend ESB Infrastructure Services Configuration Guide* and http://archiva.apache.org/.

1. Deploy `nexus-webapp.war` into Tomcat, and start Tomcat.
2. In your Maven `settings.xml` file add the section:

   ```
   <server>
       <id>nexus</id>
       <username>deployment</username>
       <password>deployment123</password>
   </server>
   ```

3. In the Maven project file `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/pom.xml`, change <distributionManagement> section to:

   ```
   <distributionManagement>

     <!-- use the following if you're not using a snapshot version. -->
     <repository>
       <id>nexus</id>
       <name>RepositoryProxy</name>
       <url>
         http://localhost:8080/nexus-webapp/content/repositories/releases
       </url>
     </repository>

     <!-- use the following if you ARE using a snapshot version. -->
     <snapshotRepository>
       <id>nexus</id>
       <name>RepositoryProxy</name>
       <url>
         http://localhost:8080/nexus-webapp/content/repositories/snapshots
       </url>
     </snapshotRepository>
   </distributionManagement>
   ```

4. In the file `<TalendRuntimePath>/container/etc/org.ops4j.pax.url.mvn`, add the Nexus repository to parameter **`org.ops4j.pax.url.mvn.repositories`**

   ```
   org.ops4j.pax.url.mvn.repositories= \
   .....
   http://localhost:8080/nexus-webapp/content/repositories/releases, \
   http://localhost:8080/nexus-webapp/content/repositories/\
   snapshots@snapshots
   ```

5. Run the following command from directory `<TalendRuntimePath>/examples/talend/tesb/rent-a-car/` :

   `mvn deploy`

6. Check on link http://localhost:8080/nexus-webapp/content/repositories/, the Rent-a-Car sample: **`org.talend.esb.examples.rent-a-car`** is successfully deployed.
7. Now you can install rent-a-car feature with command:

```
features:addurl   mvn:org.talend.esb.examples.rent-a-car/features/5.2.1/
xml
```

and work with it as usual.

8.7. Preparing Talend Administration Center to deploy the demo to Talend Runtime

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

For more information on installing and using the Talend Administration Center, including software prerequisites, see *Talend Enterprise ESB Installation Guide*.

For more detailed descriptions of the ESB Conductor feature, see *Talend Administration Center User Guide*.

The Talend Administration Center is a web-based administration application for all aspects 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 interoperates with it.

This section summarises the installation of the Talend Administration Center, and the Talend Artifact Repository, and how to use them in conjunction with the Rent-a-Car demo.

1. In order to run this example, please follow the instructions in the *Talend Enterprise ESB Installation Guide* to install and run the following components in the normal order.

   Note: these are the minimum components needed, and are sufficient for the examples here, but you may install more components if you wish. (In particular, the Commandline component is not needed).

   So, here is the list of components you need:

   - SVN Server, to store all your project data. (Note: installation and usage of SVN is only required if you like to use a central code repository. It is not mandatory for the examples here and also not required for the next steps).
   - Tomcat application server
   - Talend Administration Center Web application (effectively a war file deployed in Tomcat)
   - Talend Artifact Repository (effectively a zip file containing Archiva)

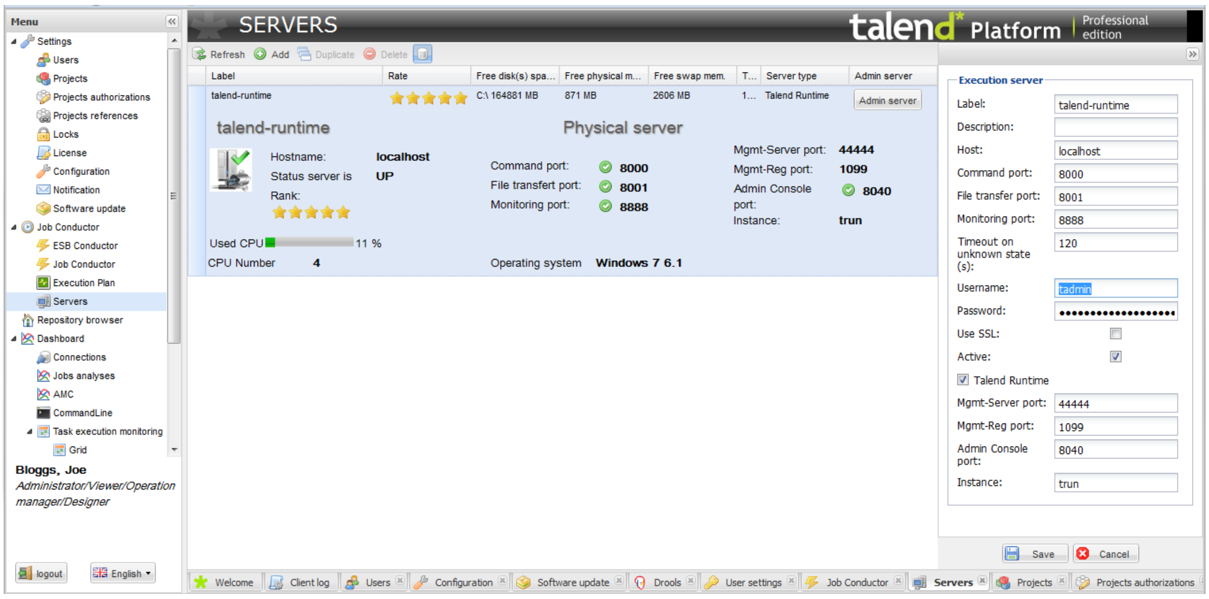
2. As a quick checklist, after the installation:

   - Talend Runtime and Tomcat are running.
   - The Talend Administration Center is deployed in Tomcat and is running on http://localhost:8080/org.talend.administrator, and your license is installed.
   - You should also have created the new user in Talend Administration Center, with appropriate roles, and you need to be re-logged in as the new user in order to see the ESB pages (for more information, see *Talend Administration Center User Guide*).
   - The Talend Artifact Repository (Archiva) is started by extracting the ZIP file in the Talend Administration Center package and running **archiva.bat console** (Windows) **archiva.sh console** (Linux). It is running on http://localhost:8082/archiva/. (It includes extra pre-configured repositories compared with standard Archiva).

3. Please ensure that a Talend Runtime container is running (Section 2.6, “*Starting a Talend Runtime container*”). Now, on the Servers page (under Job Conductor on the left hand side), create a new server corresponding to this Talend Runtime container: select Add and specify the following parameters:

   - Host: localhost
   - Command port: 8000
   - File transfer port: 8001
   - Monitoring port: 8888
   - Username: tadmin
   - Password: tadmin

   Select the Talend Runtime check box. And click Save.



Label	Rate	Free disk(s) spa...	Free physical m...	Free swap mem.	T...	Server type	Admin server
talend-runtime	★★★★★	C:\ 164881 MB	871 MB	2606 MB	1...	Talend Runtime	Admin server

talend-runtime

Physical server

Hostname:	localhost
Status server is	UP
Rank:	★★★★★
Command port:	8000
File transfert port:	8001
Monitoring port:	8888
Mgmt-Server port:	44444
Mgmt-Reg port:	1099
Admin Console port:	8040
Instance:	trun
Used CPU	11 %
CPU Number	4
Operating system	Windows 7 6.1

Execution server

Label:	talend-runtime
Description:	
Host:	localhost
Command port:	8000
File transfer port:	8001
Monitoring port:	8888
Timeout on unknown state (s):	120
Username:	tadmin
Password:	•••••••••••••••••••
Use SSL:	☐
Active:	☑
☑ Talend Runtime	
Mgmt-Server port:	44444
Mgmt-Reg port:	1099
Admin Console port:	8040
Instance:	trun

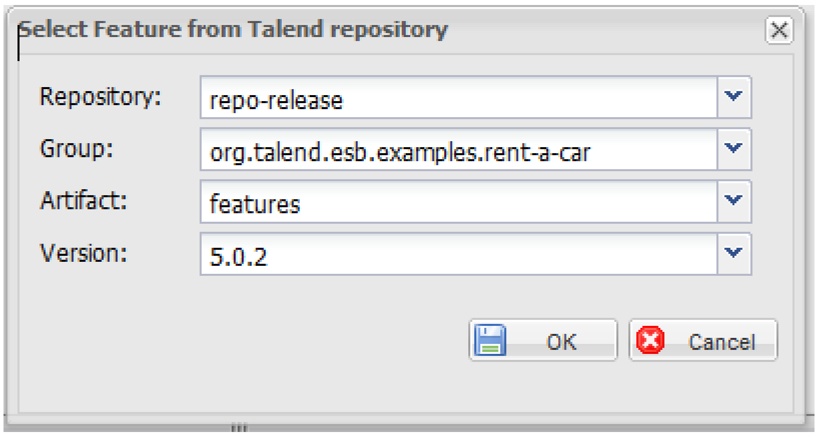
Save Cancel

8.8. Deploying the demo with Talend Administration Center to Talend Runtime

At this point, we assume that the demo has been uploaded to the Talend Artifact Repository either manually (see Section 8.5, “*Uploading the demo to a local Archiva repository using Maven files*”) or from the Studio, using a menu option (see the *Talend ESB Studio User Guide* or *Talend Data Services Studio User Guide*, depending on your product or subscription).

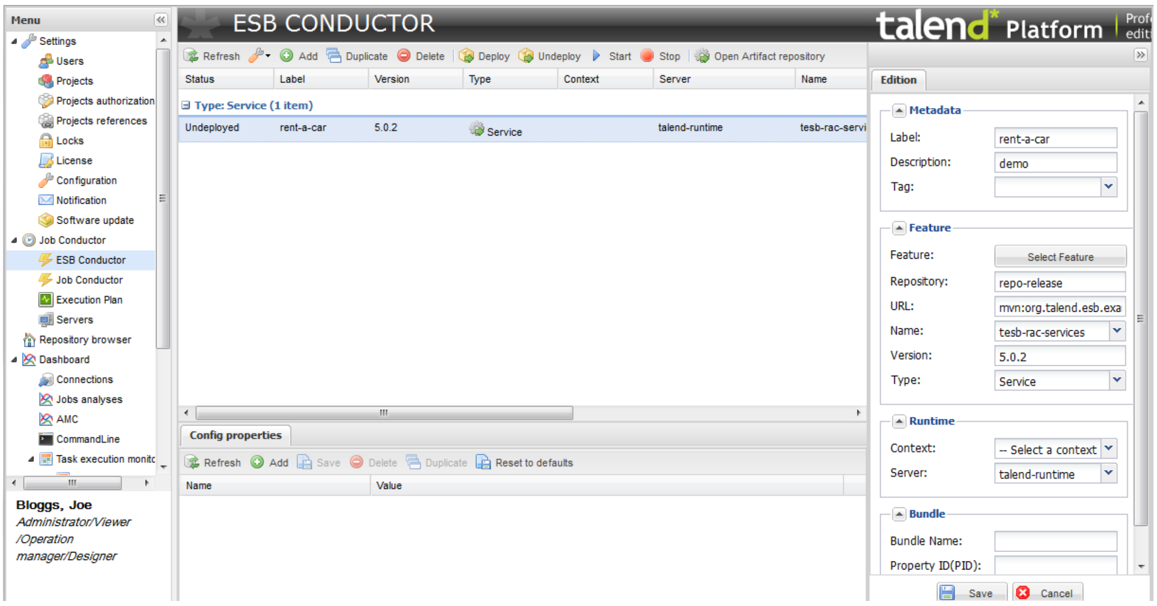
Now we deploy the demo with the Talend Administration Center (for more details on the steps here, see *Talend Administration Center User Guide*).

1. In the Talend Administration Center, from the menu, select Job Conductor, and then ESB Conductor.
2. Then create a new deployment task. Click the Add button and fill all the fields in the right panel of screen:
3. In the Metadata section, fill in the label ("rent-a-car") and description ("demo")
4. Then click on "Select Feature", and fill in the options like these:



Repository:	repo-release
Group:	org.talend.esb.examples.rent-a-car
Artifact:	features
Version:	5.0.2

Click Ok. In the runtime section, type in `talend-runtime` for the server name. Leave the rest of the parameters at their default values, as shown below:



Bloggs, Joe
Administrator/Viewer /Operation manager/Designer

Status	Label	Version	Type	Context	Server	Name
Type: Service (1 item)						
Undeployed	rent-a-car	5.0.2	Service		talend-runtime	tesb-rac-servi

Name	Value

Metadata

Label:	rent-a-car
Description:	demo
Tag:	

Feature

Feature:	Select Feature
Repository:	repo-release
URL:	mvn:org.talend.esb.exa
Name:	tesb-rac-services
Version:	5.0.2
Type:	Service

Runtime

Context:	-- Select a context
Server:	talend-runtime

Bundle

Bundle Name:	
Property ID(PID):	

5. The deployment task appears in the list. Now, click on 'Deploy' and the feature will be deployed to the selected Server (Talend OSGi container) and will be started automatically.

The services (the OSGi Feature: `tesb-rac-services`) are now deployed on the `talend-runtime` Server and they are automatically started and can now be used. If you type **list** at the console of the container, you will see that the services are active.

You can now, for example, run the Rent-a-Car client application, which uses the services. See Section 4.1, *"Overview of Rent-a-Car demo example"* for more details.

For more details on the ESB Conductor see *Talend Administration Center User Guide*.

Glossary

This glossary explains some of the terms in use in this manual.

Artifacts	Generated classes, interfaces and other files, which support communication between pairs of Java client and service endpoints. They can also be stored (and retrieved) using the Talend Artifact Repository, or Apache Archiva.
JAAS	See Java Authentication and Authorization Service.
Java Authentication and Authorization Service	This is security functionality at a user and application level to provide authentication and authorization services for clients and services.
JMS	See Java Message Service.
Java Message Service	is a standarized Java API for sending messages between two or more applications. Two types of communication are supported: point-to-point (direct communication from a producer to a specified consumer) and publish/subscribe (based on topics to which consumer(s) subscribe).
Java Management Extensions	web-based services for application and network management and monitoring. On the application or service side is a JMX MBean instanciation that supplies the information on memory, active threads, logging, and so on. On the client side is jconsole, which is a GUI JMX client displaying the information.
JMX	See Java Management Extensions.
Karaf	Apache Karaf is lightweight OSGi container onto which various components, clients and services can be dynamically deployed. See Also OSGi.
OSGi	The OSGi technology is a set of specifications that provide a framework so that applications can have components that are dynamically loaded.
SAML 2.0	See Security Assertion Markup Language.
Security Assertion Markup Language	is based on XML, and typically uses SOAP over HTTP to transmit information. SAML is an open standard for exchanging authentication and authorization data between between an identity provider (which provides authentication services to a client) and a service provider (which allows and authenticated user to invoke a service for which they are authorized to use).
Security Token Service	An independent service that provides security functionality between clients and services who don't neccessarily have a trust relationship. A client authenticates itself with the STS based on policies and requirements defined by the STS. The STS then provides a security token (example: a SAML token) that the client then uses to talk to the target service. The service can validate that token to make sure it really came from the trusted STS.
SOAP	is a protocol specification for exchanging structured information (using XML format). The protocol includes the facility to define how messages are encoded and processed. It is structured as a SOAP envelope, which encloses a header and message body. It may be used with a variety of transport protocols, for example, over HTTPS, making it ideal to be used for security purposes.
STS	See Security Token Service.
Web Services Description Language WSDL	A web interface to a service, providing information to client about how to interact with the service. It provides information such as port number and operation signatures (independent of implementation).

WSDL	See Web Services Description Language WSDL.