Data Service and Routing
Examples
Copyright

Adapted for 7.3.1. Supersedes previous releases.

Copyright © 2020 Talend. All rights reserved.

The content of this document is correct at the time of publication.

However, more recent updates may be available in the online version that can be found on Talend Help Center.

Notices

Talend is a trademark of Talend, Inc.

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

End User License Agreement

The software described in this documentation is provided under Talend's End User Software and Subscription Agreement ("Agreement") for commercial products. By using the software, you are considered to have fully understood and unconditionally accepted all the terms and conditions of the Agreement.

How to create a data service

To create a data service using the Integration perspective of Talend Studio, you have to design a data service Job that addresses all of the different sources and targets required for data integration processes and combines them with Web services.

The Integration perspective of Talend Studio provides the Services item to help create a data service Job from a given WSDL defined in the Repository tree view.

The following sections present a scenario to illustrate how to create a WSDL file and the data service Jobs to provide and consume a Web service.

Discovering the scenario

To illustrate the way Talend Studio combines data integration with Web services, find below a real-life example scenario. In this scenario, you will define an airport Web service by creating a WSDL file, send a request to this Web service to get country codes and retrieve the response from the Web service for a further use. To do this, you need to create two data service Jobs:

• one Job that will give access to the Web service via a WSDL, to send a request and retrieve the response - the data service provider.
• one Job that will actually send data to request the Web service - the data service consumer.

Defining the Web service

From the Services item of the Repository tree view, you are able to define the Web service of interest by creating a WSDL file or by importing an existing WSDL file.

In this scenario, we will create a new WSDL file to define the airport Web service.

For more information on how to create a WSDL file from scratch, see the How to create a Service and How to edit a WSDL file sections of the Talend Studio User Guide at https://help.talend.com.

Procedure

1. On the Repository tree view, right-click the Services node and from the contextual menu, select Create Service.
2. In the pop-up wizard, enter the information corresponding to the fields you need to complete. In this scenario, enter airport in the Name field. Click Next to go to the next step.

3. Select the Create new WSDL option.
4. Click **Finish** to validate this creation.

The service opens in the design workspace with a basic WSDL skeleton, which contains one service, one binding and one port type of one operation.

5. Right-click the port *airportPort* in the WSDL skeleton and select **Show properties**. In its **Properties** view, define its name in the **Name** field, for example, *airportSoap*. In the **Address** field, specify the address where you want the service to be published, for example, *http://localhost:8200/airport.service*.
6. Click the binding in the WSDL skeleton. In its **Properties** view, define the name of the binding, for example *airportSoap*.

7. Click the portType *airportPortType* in the WSDL skeleton. In its **Properties** view, define the name of the portType, for example *airportSoap*.

8. Click the operation *airportOperation* in the WSDL skeleton. In its **Properties** view, set its name as *getAirportInformationByISOCountryCode*.

9. Save the WSDL file. We will use it to build the Web service.

   Then the newly defined Web service with exclamation icon displays under the **Services** node of the **Repository** view. The exclamation icon means that this defined Web service is not yet used.
10. Under the **Services** node, right-click *airport 0.1* and select **Import WSDL Schemas**.
   This option imports the WSDL metadata from the service into the **Repository**, under the **Metadata > File xml**, which allows you to share the operations details across services and other components.

### Creating the data service provider

In this scenario, the data service provider uses the **tESBProviderRequest** and the **tESBProviderResponse** components to create the access to the airport Web service and uses the **tXMLMap** component to join the airport data provided by a given MySQL database into the request-response main flow for publication. The database data is loaded by the **tMysqlInput** component.

### Creating the data service provider Job

**Procedure**

1. Under the **Services** node of the **Repository** tree view, right-click the operation of the newly defined airport Web service and from the contextual menu, select **Assign Job**. In this scenario, this operation is *getAirportInformationByISOCountryCode*.

2. The **Assign Job** wizard opens. Select the **Create a new Job and Assign it to this Service** operation and click **Next**.

   ![Assign Job wizard](image)

3. In the **New Job** view of the wizard, the Job to be created is already named automatically, so simply click **Finish**. A draft Job is opened on the workspace.

### Dropping and linking the components

In the draft Job, a **tESBProviderRequest** and a **tESBProviderResponse** are already selected and configured. **tESBProviderRequest** will send a request to the specified Web service and **tESBProviderResponse** will send back the response corresponding to the request. These two components can be found in the **ESB** group of the **Palette**.

**Procedure**

1. Right-click **tESBProviderRequest**, hold and drag to **tESBProviderResponse** to link these two components.
2. Drop a **tXMLMap** in the middle of the row link from the **Palette** and in the pop-up window, name the output link as, for example, *airport_response*. This will also be used as the name of the output table in the map editor of **tXMLMap**.

Then your data service provider *Job* should look like:

![Data service example](image)

The red cross icon prompts you to configure the **tXMLMap** component.

3. From the **Db Connections** node of the **Repository** tree view, drop the connection to the airport data, the *airport* database table in this example, onto the workspace. Then the **Components** wizard is opened.

  For more information on how to create a database connection in the **Repository**, see the *Centralizing database metadata* section of the Talend Studio User Guide at [https://help.talend.com](https://help.talend.com).

4. Double click **tMysqlInput** in this wizard to create the corresponding component on the workspace and link it to **tXMLMap**.

**Results**

In this scenario, the airport data is composed of airport names and the corresponding country codes. The following figure presents the database table in use.

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR</td>
<td>Roissy</td>
</tr>
<tr>
<td>FR</td>
<td>Orly</td>
</tr>
<tr>
<td>FR</td>
<td>Marignan</td>
</tr>
<tr>
<td>CN</td>
<td>Beijing Capital</td>
</tr>
<tr>
<td>CN</td>
<td>Shanghai Hongqiao</td>
</tr>
</tbody>
</table>

Till now, you need only to configure **tXMLMap** as the other components are already configured automatically.

**Configuring tXMLMap**

**Procedure**

1. On the workspace, double click **tXMLMap** to open its editor. At this moment, the editor should look like:
2. In the main row table of the input flow side (left), right-click the column name *payload* and from the contextual menu, select **Import from Repository**. Then the **Metadatas** wizard is opened. For more information on how to create the XML tree, see the *Using the document type to create the XML tree* section of the Talend Studio User Guide at https://help.talend.com.

3. Expand the **File XML** node in this wizard, select the schema of the request side and click **OK** to validate this selection. In this example, the schema is *getAirportInformationByISOCountryCode*.

4. Do the same to import the hierarchical schema for the response side (right). In this example, this schema is *getAirportInformationByISOCountryCodeResponse*.

5. Then to create the join to the lookup data, click the **CountryAbbrviation** node in the main row of the input side (left), hold and drop it onto the **Exp.key** column of the lookup flow, corresponding to the *id* row.
6. On the table representing the lookup flow, click the wrench icon on the up-right corner to open the setting panel.

7. Set **Lookup Model** as **Reload at each row**, **Match Model** as **All matches** and **Join Model** as **Inner join**.

   ![Diagram showing setting options](image)


   A step-by-step tutorial related to this Join topic is available on the Talend Technical Community Site. For further information, see http://talendforge.org/tutorials/tutorial.php?language=english&idTuto=101.

8. Do the same to open the setting panel on the output side (right) and set the **All in one** option as **true**. This ensures that only one response is returned every time when one request is sent, as, otherwise, the airport data from the given database may provide several airports, thus several responses, to each country code that you send as one request.

   For more information about **All in one**, see the *How to output elements into one document* section of the Talend Studio User Guide at https://help.talend.com.
9. Click the name row in the lookup flow (left), hold and drop it onto the **Expression** column corresponding to the `tns:getAirportInformationByISOCountryCodeResult` node in the XML tree view of the output flow (airport_response in this example).

Then your **tXMLMap** editor should look like:
10. Click **OK** to close the editor and validate this configuration.

**Executing the provider Job**

**About this task**

Now that components are configured, the Job can be executed.

To do so, proceed as follows:

**Note:**

In the real-world practice, you can as well add hierarchical data for lookup.
How to create a data service

Procedure

Press **F6** to run this Job and once launched, the **Run** view is opened for you to read the execution result.

```java
[statistics] connecting to socket on port 3427
[statistics] connected
[INFO ]: crg.apache.cxf.service.factory.ReflectionServiceFactoryBean - Creating Service {http://airportsoap.sooper.de}airport from WSDL: E:/Talend/Talend-Studio-r110103-V5.4.1NB/workspace/PROJECT_20131029/services/airport_0.1.wsdl
[INFO ]: crg.apache.cxf.endpoint.ServerInpl - Setting the server's publish address to be http://localhost:8200/airport service
[INFO ]: crg.eclipse.jetty.server.AbstractConnector - Started SelectChannelConnector@localhost:8200
```

Results

The data service provider Job is executed, and will listen to all requests sent to the Web service until you click the **Kill** button to stop it as by default, the **Keep listening** option in the **Basic settings** view of **tESBProviderRequest** is selected automatically.

What to do next

Now, you have to configure the consumer Job that will send actual requests to the data service provider Job you just created.

Creating the data service consumer

To create your consumer, you need at least these components: an input component allowing to read a data flow, a **tXMLMap** component that will map this flat data to a hierarchical document, the format expected by **ESB** components, the **tESBConsumer** components that will request the corresponding Web service and read its result and the **tLogRow** component that displays the Job execution result.

For this specific scenario, you will use a **tFixedFlowInput** as input component to send a country code request to the **tESBConsumer** component.

Creating the data service consumer Job

Procedure

1. Right-click **Job Designs** in the **Repository** tree view and select **Create Job**.
2. In the dialog box displaying then, only the first field **(Name)** is required. Type in **airportJob** and click **Finish**. An empty Job opens on the main window and you can continue to create the Job of interest.

Dropping and linking the components for the consumer

Procedure

1. Click the **tFixedFlowInput** component from the **Misc** component group of the **Palette**, then click to the left of the design workspace to place it on the design area.
2. Do the same to drop tXMLMap, tESBConsumer and a tLogRow component from their respective
group on the Palette.

3. To link the input components to the mapper, simply right-click tFixedFlowInput, hold and drop it
to tXMLMap.

4. To link tXMLMap to tESBConsumer, right-click tXMLMap, hold and drag to tESBConsumer.
A popup window displays, type in the name you want to give to the output row link: request,
for example and then accept the propagation that prompts you to get the schema from
tESBConsumer.

5. Link the tESBConsumer component to it with a Response row link.

Then the data service consumer Job should look like:

Results

Then you need to configure each of these components.

Configuring the tFixedflowInput component

Procedure

1. Double-click tFixedflowInput to open its Component view.

2. Click the three-dot button next to the Edit schema field to open the Schema window.
3. Click the plus button once to add one **Column** to the schema and name it **country_code**. Keep the **Type** field as **string**.

4. Click **OK** to validate this schema. Then in the **Mode** area of the **tFixedFlowInput** basic settings, the active option should be **Use Single Table** and the **country_code** row is already added automatically to the **Values** table.

5. In the **Value** column of the **Values** table, type in **CN** within quotation marks.

**Configuring the tXMLMap component**

**Procedure**

1. Double-click **tXMLMap** to open its **Map editor**.
2. To import the WSDL schema for the request, do the same as explained earlier when creating the data service provider. In this example, the request schema is \textit{getAirportInformationByISOCountryCode}. For further information, see Configuring tXMLMap on page 9.

3. Click \textit{country_code} in the main flow table of the input side (left), hold and drop it to the Expression column corresponding to the \textit{tns:CountryAbbrviation} node in the XML tree of the request table on the output side (right).

\begin{quote}
Note: In the real-world operations, tXMLMap is able to handle the highly complex transformations of hierarchical data.
For all of the available features of tXMLMap, see the tXMLMap operation section of the Talend Studio User Guide at https://help.talend.com.
\end{quote}

4. Click OK to validate this configuration.

\textbf{Configuring the tESBConsumer component}

\textbf{Procedure}

1. Double-click the \textit{tESBConsumer} component to open its Component view.

2. Click the [...] button next to the Service configuration field to open the WSDL editor.

3. In the editor, browse to the WSDL file provided, the Port Name and Operation fields are automatically filled in with the port(s) and method(s) available in the Web service.
4. Select **airportSoap** in the Port Name field and select **getAirportInformationByISOCountryCode** in the Operation field.

5. Click **OK** to validate the configuration.

   The tLogRow component will automatically retrieve the schema from the previous component. If not, double-click it and click the **Sync columns** button in its Component view.

**Executing the consumer Job**

**Procedure**

To execute this Job, press **F6**.

Once done, the Run view is opened automatically, where you can check the execution result.
Results

This Job sends one country code request to the consumer that requests the Web service through the provider Job. The response is retrieved by the `tESBProviderResponse` and `tESBConsumer` components at the same time.

Note that although the provider Job received some request, it did not stop and is still listening to new requests.
How to create a Route using a data service

This section provides a scenario to illustrate how to create a Route using the data service we created in the previous section.

In this scenario, we will define a Route using the Web service we just created, send a request to this Web service and retrieve the response from the Web service for a further use.

Starting the Apache ActiveMQ server

To send data to request the Web service, we use Apache ActiveMQ as the message broker. We need to launch the ActiveMQ server before executing the Route. For more information about installing and launching ActiveMQ server, see https://help.talend.com/access/sources/content/topic?pageid=install_activemq&EnrichVersion=7.3&afs:lang=en and the Apache Web site http://activemq.apache.org/index.html.

Creating the Route

To create the Route, we need a `cJMSConnectionFactory` component to specify the connection factory for message handling, two `cJMS` components, one to read a JMS message from one queue and send the request to the Web service and the other to retrieve the response from the Web service and save it in another queue, and a `cSOAP` component to connect to the Web service.

Procedure

1. On the Repository tree view, right-click the Route node and from the contextual menu, select Create Route.
2. In the pop-up wizard, enter the information corresponding to the fields you need to complete. In this scenario, enter airportRoute in the Name field and click Finish. The Route is opened in the design workspace.

Dropping and linking the components for the route

Procedure

1. From the Palette, drop a `cMQConnectionFactory`, a `cSOAP`, a `cProcessor`, and two `cJMS` components on to the design workspace.
2. Label the components for better identification of their roles and link them using the Row>Route connection as shown below.
Configuring the components

About this task

Now that the components are linked, their properties should be defined. Label the components for better identification of their roles, Starter for cTimer, Set_message_body for cSetBody, and LogMessages for cLog.

Note:
It is recommended to label each component with a unique name to better identify its role in the Route. This is especially useful for the code generation of some components, for example, the cSOAP component. For more information, see Talend ESB Mediation Components documentation in Talend Help Center.

Procedure

1. Double-click the cTimer component to open its Basic settings view.

   ![cTimer component settings](image)

   - In the Repeat field, enter 1 to generate the message exchange one time. Keep the default settings of the other options.

2. Double-click the cSetBody component to open its Basic settings view.

   ![cSetBody component settings](image)

   - Select Constant in the Language list and enter "Hello World!" in the Expression field as the message body.
How to create a Route using a data service

5. Keep the default settings of the **cLog** component to log the message exchanges.

![cLog component settings](image)

![logMessages component settings](image)

**Executing the Route to implement the data service**

**Procedure**

1. Click the **Run** view to display it and click the **Run** button to launch the execution of the Route. You can also press **F6** to execute it.

   The Route is successfully started.
2. Switch to the ActiveMQ Web console. The incoming queue is already created.

3. Enter the message body as shown below in the incoming queue and send the message.

```xml
<air:getAirportInformationByISOCountryCode>
  <air:CountryAbbreviation>CN</air:CountryAbbreviation>
</air:getAirportInformationByISOCountryCode>
```

4. In the Route designer, we can see that the message payload is sent to the Web service. The Web service get called and the response is sent to another queue, `OUT` as we have configured. The message body of this queue is printed on the console.
We can also view the OUT queue from the ActiveMQ Web console. It has one message in it, as shown below.

**Queues**

<table>
<thead>
<tr>
<th>Name</th>
<th>Number Of Pending Messages</th>
<th>Number Of Consumers</th>
<th>Messages Enqueued</th>
<th>Messages Dequeued</th>
<th>Views</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How to create a soapUI project to test the data service

In the previous section, we have illustrated how to create a consumer Job to request the Web service we created. This section will show you how to create and execute a Test Request for the Web service using SoapUI. You can download SoapUI from http://www.soapui.org/.

In this scenario, we will send a request to the Web service using soapUI to test whether the expected response can be retrieved. To do this, we need to create a soapUI project that will send data to request the Web service.

Creating a soapUI project

From the Projects node of the soapUI Navigator, you are able to create soapUI projects that allow you to execute soapUI testing.

In this scenario, we will create a soapUI project and add the WSDL file that has been used in the previous sections to the project to access the given Web service.

Procedure

1. In the soapUI Navigator, which is the tree structure at the left in the soapUI GUI, right click the Projects node and select New soapUI Project.

2. The New soapUI project wizard opens. Enter a project name in the Name field, for example, airport in this use case. In the Initial WSDL/WADL field, click Browse to navigate to the existing WSDL file airport_0.1.wsdl. Keep the default settings of the other options and click OK.
3. The new project appears in the navigator and the WSDL is added to it. A sample request `Request1` is created for the `getAirportInformationByISOCountryCode` operation.

4. Double click `Request1` to open the code editor to the right of the navigator. The URL of the Web service is shown in the address bar and the sample request `Request1` is shown in the Request panel. The `?` indicates that an input parameter `air:CountryAbbrviation` is needed for this operation. The Response panel is empty.

5. Replace `?` with `CN` as the input in the Request panel and click the icon to submit the request to the specified endpoint URL.
6. The response of the submitted request is shown in the Response panel.

Results

As we have configured in the data service provider Job, Beijing Capital and Shanghai Hongqiao are the airport names corresponding to the country code CN. So here we have got the expected response, which means the Web service can be implemented successfully.
Route communicating with data integration Job

This section provides a scenario of how to use the Camel components not included in the palette in a Route and to use sockets for connecting a Job and a Route.

In this scenario, we will create a data integration Job that listens messages from one socket port and sends out the messages to another port. Then we will create a Route that sends a test message to the listening port and logs message changes of the incoming and outcoming ports.

Creating a data integration Job

Procedure

1. From the Integration perspective of the Talend Studio, create a Job and open it in the design workspace.
   
   For more information on how to create a Job, see the Creating a Job section of the Talend Studio User Guide at https://help.talend.com.

2. From the palette, drag and drop a tSocketInput, a tSocketOutput, and a tLogRow component onto the design workspace. Link them together with the Row > Main connection.

3. Double-click the tSocketInput component to open its Basic settings view in the Component tab.
public void configure() throws java.lang.Exception {
    org.apache.camel.Endpoint endpoint_cTalendJob_1 = endpoint("talend:"
            + esbdemos.routecommunication_0_1.RouteCommunication.class
            .getName() + ":Default&propagateHeader=true")
            .getProperty()
            .setContext("?context="
            + "Default&propagateHeader=true")
            .setProperties(propertiesMap);
    Map<String, String> propertiesMap = new HashMap<String, String>();
    propertiesMap.put("body", propertyToString("Hello World!"));
    endpoint_cTalendJob_1.getCamelContext()
            .setProperties(propertiesMap);
    from(uriMap.get("Starter_cTimer_1"))
            .routeId("Starter_cTimer_1")
            .setHeader("header").constant("FileName").id("cSetHeader_1")
            .to(endpoint_cTalendJob_1).id("cTalendJob_1")
            .setProperties(propertiesMap);
}

4. Define the listening Port number in the Port field, 8900 for example. By default, the local host server is used. Set the amount of time (in seconds) to 30000 in the Timeout field, after which the Job will time out.

5. Double-click the tSocketOutput component to open its Basic settings view in the Component tab.

![Call_Dl_job(cTalendJob_1)](image)

6. Define the Host IP address and the Port number where the data will be passed on to, as shown above.

7. The tLogRow component is used to monitor data processed and does not need any configuration.

8. Press Ctrl+S to save your Job.

### Creating a Route to communicate with the Job

**Procedure**

1. Create a Route and open it in the design workspace.
   For more information on how to create a Route, see the How to create a Route section of the Talend Studio User Guide at [https://help.talend.com](https://help.talend.com).

2. From the palette, drag and drop a cSetBody, and five cMessagingEndpoint components onto the design workspace. Label the components for better identification of their roles and link them together with the Row > Main connection.
Configuring the incoming sub-route

Procedure

1. Double-click the **testMina** component to open its **Basic settings** view in the **Component** tab.

2. In the **URI** field, enter the code "timer:testOneWayMina?delay=1000&period=2000" to define a timer for starting message exchanges. In this use case, we want each message to be delivered after a 1-second delay at a period of 2 seconds.

3. Double-click the **initializeRequest** component to open its **Basic settings** view in the **Component** tab.

4. Select **Constant** from the **Language** list box and type in "test message\n" in the **Expression** field.

5. Double-click the **logRequest** component to open its **Basic settings** view in the **Component** tab.
6. In the **URI** field, enter "log:jobSocketIn" where the incoming message exchanges are logged.

7. Double-click the **jobSocketIn** component and click **Advanced settings** view in the **Component** tab.

8. In this use case, we will use the Camel component camel-mina as the transport. To use this component, click at the bottom of the **Dependencies** list to add a row and select **mina** from the drop-down list. For more information about Mina, see the site [http://camel.apache.org/mina.html](http://camel.apache.org/mina.html). Alternatively, you can use a **cConfig** component and add the library of MINA to the **Dependencies** list of the **cConfig** component. To do so, click at the bottom of the **Dependencies** list to add a row. Select this row and click the [...] button at the end to show the **Select Module** wizard.
Select camel-mina-alldep-2.9.2.jar from the inner modules and click OK to add it to the Dependencies list.

9. Click the Basic settings view in the Component tab of the jobSocketIn component. In the URI field, enter "mina:tcp://localhost:8900 + "?textline=true&sync=false" to send the message to the Mina endpoint of a TCP service on port 8900 as a text line in the InOnly mode.
Configuring the outcoming sub-route

Procedure

1. Double-click the jobSocketOut component to open its Basic settings view in the Component tab.

   ![jobSocketOut](image)

   **Basic settings**
   - URI: "mina:tcp://localhost:8901/?textline=true&sync=false"

2. In the URI field, enter "mina:tcp://localhost:" + 8901 + "?textline=true&sync=false" of the outcoming socket port.

3. Double-click the logResponse component to open its Basic settings view in the Component tab.

   ![logResponse](image)

   **Basic settings**
   - URI: "log:jobSocketOut"

4. In the URI field, enter "log:jobSocketOut" where the outcoming message exchanges are logged.

5. Press Ctrl+S to save your Route.

Viewing the code and executing the Route and the Job

Procedure

1. Click the Code tab at the bottom of the design workspace to check the generated code.
public void initRoute() throws Exception {
    routeBuilder = new org.apache.camel.builder.RouteBuilder() {
        public void configure() throws Exception {
            from(uriMap.get("testMina")).routeId("testMina")
                .setBody().constant("test message\n").id(
                "cSetBody_1")
                    .id(uriMap.get("logRequest")).id(
                "cMessagingEndpoint_2")
                    .id(uriMap.get("jobSocketIn")).id(
                "cMessagingEndpoint_3");
            from(uriMap.get("jobSocketOut")).routeId("jobSocketOut")
                .id(uriMap.get("logResponse")).id(
                "cMessagingEndpoint_5");
        }
    };
    getCamelContext().get(0).addRoutes(routeBuilder);
}

As shown above, the message flow from testMina is given a payload by cSetBody and then sent to logRequest and jobSocketIn. The other message flow is sent from jobSocketOut to logResponse.

2. Press F6 to execute the Route, and then run the Job in the same way. The Route will keep trying to connect to the defined ports until the Job starts.

The message change log of the incoming and outcoming ports is printed in the execution console of the Route.

On the Job side, the message processed is also displayed in console.
Route communicating with data integration Job

Starting job r06_socketPipeJob_edit at 13:52 26/06/2012.

[statistics] connecting to socket on port 3446
[statistics] connected
socket connected
test message
test message
test message
test message
test message
test message
test message
test message
test message
Job r06_socketPipeJob_edit ended at 13:52 26/06/2012.
[exit code=0]
Route using connection pooling

*Talend Studio* in combination with ESB data services and Route based Jobs supports connection pooling in the Oracle, MySQL and generic JDBC database components. This section introduces the use of connection pooling in Jobs with *tRouteInput* component, which can be started from a Route using the *cTalendJob* component. Similar use cases and configurations can also be used with ESB data services.

As an example, the MySQL database components will be used here. The same usage are valid for all the Oracle and generic JDBC database components.

Configuring datasource in Talend Runtime

To use datasource in Talend Runtime, you need to install the MySQL driver in to a container and set the datasource configuration file.


Installing MySQL driver into a container

The Talend Open Studio for ESB package provides JDBC drivers only for the Derby database. To use the MySQL database, its corresponding JDBC driver need to be explicitly installed into the container before installing the datasource. You can install the MySQL driver from a public Maven repository.

Procedure

In a Talend Runtime container, run the following command (change the database version numbers if applicable):

```
bundle:install mvn:mysql/mysql-connector-java/5.1.18
```

For more information on how to install the H2, Oracle, DB2, SQLServer and PostgreSQL JDBC drivers into a container, see the section on datasource installation in the *Talend ESB Infrastructure Services Configuration Guide*. 
**Setting the datasource configuration file**

**Procedure**

1. Set the database connection details in the MySQL datasource configuration file named `datasource-mysql.xml` in the `<TalendRuntimePath>/add-ons/datasources/dataservice` folder.

   ```xml
   <bean id="mysqlDataSource" class="com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource">
     <property name="url" value="jdbc:mysql://localhost:3306/root"/>
     <property name="user" value="root"/>
     <property name="password" value=""/>
   </bean>

   <bean id="dataSource" class="org.apache.commons.dbcp.datasources.SharedPoolDataSource" destroy-method="close">
     <property name="connectionPoolDataSource" ref="mysqlDataSource"/>
     <property name="maxActive" value="20"/>
     <property name="maxIdle" value="5"/>
     <property name="maxWait" value="-1"/>
   </bean>

   <service ref="dataSource" interface="javax.sql.DataSource">
     <service-properties>
       <entry key="osgi.jndi.service.name" value="jdbc/sample"/>
     </service-properties>
   </service>
   ```

   2. Copy the configuration file to the deploy folder within the Talend Runtime container directory. The datasource alias `jdbc/sample` will be used in the MySQL database components when configuring the Job in the Studio. You can also change it as needed.

**Creating a data integration Job**

In this section, a data integration Job is created to insert data into a MySQL database.

**Adding and linking components**

**Procedure**

1. From the palette, drag and drop a `tPrejob`, a `tMysqlConnection`, a `tRouteInput`, a `tMap`, a `tLogRow`, a `tMysqlOuput`, a `tPostjob` and a `tMysqlClose` component onto the design workspace. Link them together with the **Row > Main** connection.
2. Link the **tPrejob** component to the **tMysqlConnection** component using a **Trigger > On Component Ok** connection to build the pre-job.

3. Link the **tRouteInput** component to the **tMap** component using a **Row > Main** connection.

4. Connect the **tMap** component to the **tLogRow** component using the **Row > New Output (Main)** connection and name the output connection as **row2**.

5. Link the **tLogRow** component to the **tMysqlOutput** component using a **Row > Main** connection.

6. Link the **tPostjob** component to the **tMysqlClose** component using a **Trigger > On Component Ok** connection to build the post-job.

**Warning:**
The use of **tMysqlClose** in the post-job as shown above in the Route-Job example is not valid for Data Service operation Jobs containing **tESBProviderRequest**, **tESBProviderResponse**, **tESBProviderFault**, or **tRestRequest** components. In these kind of Jobs it is not recommended to use close connection components like **tMySQLClose**. Because closing datasource connection is implemented directly in the **tESBProvider* and **tRestRequest** components.

### Configuring the pre-job

In the pre-job, the component does not have any parameter or property to configure. It just initialize the connection to the MySQL database before the execution of the main Job. So all the required configurations are made in the **tMysqlConnection** component.

**Procedure**

1. Double-click **tMysqlConnection** to open its **Basic settings** view in the **Component** tab.
2. Set the database connection details in the relevant fields, including the host name or IP address of your database server, the listening port number, the database name, the user name and password for your database authentication. These connection parameters are used when you run the Route to call this Job in the Studio. If you want to deploy the Route and the Job in Talend Runtime, you can leave these parameters empty and go to the next step directly.

3. Select the Specify a data source alias check box and enter the data source alias, which is set in the datasource configuration file in Talend Runtime.

4. In the Advanced Settings tab, select the Auto Commit check box to commit any changes to the database automatically upon the transaction.

Configuring the main Job

Procedure

1. Double-click tRouteInput to open its Basic settings view in the Component tab.
2. Click […] next to Edit Schema. In the schema dialog box, click the plus button to add a new line of String type and name it body. Set the length of this column to 50. Click OK to close the dialog box.

![Schema of tRouteInput_1](image)

3. In the Simple Expression column for the body element, enter "${in.body}" to get the body of the input message from the Route.

4. Double-click tMap to open the tMap editor.

![TMap with Simple Expression](image)

5. In the lower right corner of the tMap editor, click [+ ] to add a column of String type in the output table and name it welcomeText. Set the length of this column to 50.

6. In the upper left corner of the editor, select the body column and drop it to the output table. In the Expression field of the output table, add + " Talend!" as the welcome text. When done, click OK to validate your changes, close the editor and propagate the changes to the next component.

7. The tLogRow component is used to monitor data processed and does not need any configuration.

8. Double-click tMysqlOuput to open its Basic settings view in the Component tab.
9. Select the **Use an existing connection** check box and select the **tMysqlConnection_1** in the **Component List** to reuse the connection details that you have already set. In the **Table** name field, enter **Hello** as the name of the table to be created. Select **Drop table if exists and create** in the **Action on table** list, and select **Insert** in the **Action on data list**. Click **Sync columns** to retrieve the schema set in the **tMap**.

### Configuring the post-job

In the post-job, the **tPostjob** component does not have any parameter or property to configure. It just closes the database connection.

#### Procedure

1. Double-click **tMysqlClose** to open its **Basic settings** view in the **Component** tab.

2. The **Component List** is already filled in with **tMysqlConnection_1**. Keep this setting and press **Ctrl + S** to save your Job.

### Creating a Route to send a message and call the Job

In this section, you will create a Route to send a message and call the Job.

For more information on how to create a Route, see the *How to create a Route* section of the Talend Studio User Guide at [https://help.talend.com](https://help.talend.com).
Dropping and linking the components

Procedure

1. Drag and drop a cTimer, a cSetHeader, and a cTalendJob from the Palette onto the design workspace.

![Diagram showing cTimer, cSetHeader, and cTalendJob components linked with Row > Route connection]

2. Link the components with the Row > Route connection as shown above.

Configuring the Mediation components

Procedure

1. Double-click the cTimer component to open its Basic settings view in the Component tab.

2. Set the values for the Period (1000), Repeat (1) and Delay (1000) fields as shown above to trigger a message exchange after a delay of 1000 milliseconds.

3. Double-click the cSetBody component to display its Basic settings view in the Component tab.

4. Select Constant in the Language list, and enter "Hello" in the Expression field.

5. Double-click the cTalendJob component to display its Basic settings view in the Component tab.
6. Select **Repository** to call a Job from the repository. In the **Repository Job** area, select **Use Selected Context**. Click [...] next to the **Job** field to open the **Assign Job** wizard. Select **Assign an existing Job to this cTalendJob component** and click **Next**.

7. In the **Job selection** view, select **tmysql** that we just created in the **Job designs** tree view and click **Finish**.
tmysql is now displayed in the Job field. By default, the latest version and the default context of it is selected.

8. Press Ctrl+S to save your Route.

**Executing the Route in the Studio**

To test the Route and the Job before deploying them into Talend Runtime, you can run the Route in the Studio first.

**Procedure**

Press F6 to execute the Route. The message exchange log is printed in the execution console of the Route.

**Results**

In the MySQL database, a one column table is created with the data *Hello Talend!*
Deploy the Route into Runtime

In this section, the Route just created is deployed in the Talend Runtime container.

Procedure

1. Right-click the Route you have created and select **Build Route** in the contextual menu.

2. In the **Build Route** wizard, click **Browse** to navigate to the directory where you want to store the Route and click **Finish**. In this use case, we export the Route directly to the deploy folder within the Talend Runtime container directory so that it starts to run immediately.
Results

When you run the Route by building it into the container/deploy folder, the `tMysqlConnection` component will look for the `sample/jdbc` datasource in the container. If the datasource exists, the connection parameters from this datasource will be used at runtime to establish the connection to the MySQL database.
Unit testing of Routes

Unit testing is an essential part of any professional software development process. This section will show you how Routes can be unit tested using the Apache Camel Testing framework.

Creating a Route

To show an example of Unit testing, we will create a Route first. This Route reads a file from the specific directory and filters the messages of the file.

Procedure

1. Drag and drop a cFile, a cConvertBodyTo, a cMessageFilter and a cMessagingEndpoint from the Palette onto the design workspace.

2. Label the components for better identification of their roles. Link the cMessageFilter component and the cMessagingEndpoint component using the Row > filter connection. Link the other components as shown above using the Row > Route connection.

3. Double-click the cFile component to open its Basic settings view in the Component tab and configure it as shown below to read files from a local file system.

4. Double-click the cConvertBodyTo component to open its Basic settings view in the Component tab and configure it as shown below to convert the message body of each file into String type.
5. Double-click the cMessageFilter component to open its **Basic settings** view in the **Component** tab and configure it as shown below to filter the message body based on the expression:

\[
\text{${body}} \text{ contains 'bar'}
\]

6. Double-click the cMessagingEndpoint component to open its **Basic settings** view in the **Component** tab and configure it as shown below to log the output of the Route.

7. In the **Run** view, click the **Run** button to launch the execution of your Route. The data that matches the filtering criterion is displayed in the console.

### Results

For more information on how to create a Route, see the *How to create a Route* section of the Talend Studio User Guide at [https://help.talend.com](https://help.talend.com).

For more information on the properties and usage of the Mediation components, see the Talend ESB Mediation Components documentation in [Talend Help Center](https://help.talend.com).
Publishing the Route to the Artifact repository

Procedure

1. Right-click the Route in the Repository tree view and select Publish in the context menu.
2. The Publish into Artifact Repository wizard opens. Keep the default settings and click Finish to publish the Route into the Artifact repository.

Results

Now your item is available as an artifact in the repository.

For more information on how to publish a Route, see the Publishing into the Artifact repository section of the Talend Studio User Guide at https://help.talend.com.

Creating a Maven project for unit test

Unit tests will be put in a separate project. We need to switch to the Java perspective and create a simple maven-based project.

For more information on how to switch to the Java perspective, see the GUI section of the Talend Studio User Guide at https://help.talend.com.

Procedure

1. Right-click the blank area in the Package Explorer and select New>Project... in the context menu.
2. The **New Project** wizard opens. Select **Maven Project** under the **Maven** node and click **Next**.

3. In the **Select project name and location** view, the **Use default Workspace location** option is selected. Keep it and click **Next**.
4. In the **Select an Archetype** view, the **GroupId** `org.apache.maven.archetypes` and **ArtifactId** `maven-archetype-quickstart` is selected. Keep the default settings and click **Next**.
5. In the **Specify Archetype parameters** view, enter `org.talend` in the **Group Id** field and `route-unit-test` in the **Artifact Id** field. Click **Finish**. The project appears in the **Package Explorer**.

![New Maven project dialog box](image-url)
6. Double-click the pom.xml file under the newly created project node to open it in the design workspace and edit it as shown below.

```xml
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.talend</groupId>
  <artifactId>route-unit-test</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-test</artifactId>
      <version>2.9.1</version>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-jdk14</artifactId>
      <version>1.6.1</version>
    </dependency>
    <dependency>
      <groupId>org.talend.camel</groupId>
      <artifactId>systemRoutines</artifactId>
      <version>1.0.0</version>
    </dependency>
    <dependency>
      <groupId>org.talend.camel</groupId>
      <artifactId>userRoutines</artifactId>
      <version>1.0.0</version>
    </dependency>
    <dependency>
      <groupId>org.example</groupId>
      <artifactId>SimpleRoute</artifactId>
      <version>0.2.0-SNAPSHOT</version>
      <type>jar</type>
    </dependency>
  </dependencies>
  <repositories>
    <repository>
      <id>repo-snapshot</id>
      <name>Snapshots bundles</name>
      <url>http://tadmin:tadmin@localhost:8082/archiva/repository/repo-snaps</url>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </repository>
  </repositories>
</project>
```

As is shown above, the Maven dependencies are Junit and Camel unit testing framework, Utility JAR files required for Talend ESB Route, and Route JAR files published from the Studio.

In this use case we will use the Route in Talend Artifact repository directly, so it needs to be added into the pom.xml file as a repository. The username and password is put in the repository URL for simplicity. You can also specify it in your ${Maven_HOME}/conf/settings.xml.

7. Save your pom.xml file.
Publishing utility JAR files to local Maven repository

Start a command line console to deploy the utility JAR files into your local Maven repository.

Procedure

1. Execute following command to deploy the system routines:
   ```shell
   mvn install:install-file -DgroupId=org.talend.camel -DartifactId=systemRoutines -Dversion=1.0.0 -Dfile=./src/ext/systemRoutines.jar -Dpackaging=jar -DgeneratePom=true
   ```

2. Execute following command to deploy the user routines:
   ```shell
   mvn install:install-file -DgroupId=org.talend.camel -DartifactId=userRoutines -Dversion=1.0.0 -Dfile=./src/ext/userRoutines.jar -Dpackaging=jar -DgeneratePom=true
   ```

Results

Now we have all the necessary dependencies to do unit tests.

Writing and running a Junit test

Procedure

1. Right-click the Maven project `route-unit-test` in the Package Explorer and select New>JUnit Test Case in the context menu.
2. The **JUnit Test Case** wizard opens. Select **JUnit 3 test** and enter **RouteTestSample** as the name of your test class. Click **Finish**.
3. The test class opens in the design workspace. Write the test as follows.

```java
package org.talend.test;
import java.util.Map;
import org.apache.camel.CamelContext;
import org.apache.camel.EndpointInject;
import org.apache.camel.Producer;
import org.apache.camel.ProducerTemplate;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.mock.MockEndpoint;
import org.apache.camel.test.junit4.CamelTestSupport;
import org.junit.Test;
import esbdemos.simpleroute_0_1.SimpleRoute;
public class RouteTestSample extends CamelTestSupport {
    @EndpointInject(uri = "mock:result")
    protected MockEndpoint resultEndpoint;
    @Produce(uri = "direct:start")
    protected ProducerTemplate template;
    private RouteBuilder builder;
    @Test
    public void testSendMatchingMessage() throws Exception {
        String expectedBody = "bar";
        resultEndpoint.expectedBodiesReceived(expectedBody);
        template.sendBody(expectedBody);
        resultEndpoint.assertIsSatisfied();
    }
    @Test
    public void testSendNotMatchingMessage() throws Exception {
        resultEndpoint.expectedMessageCount(0);
        template.sendBody("foo");
        resultEndpoint.assertIsSatisfied();
    }
    protected CamelContext createCamelContext() throws Exception {
        SimpleRoute route = new SimpleRoute();
        // Setup the test aliases
        Map<String, String> uriProperties = route.getUriMap();
        uriProperties.put("start", "direct:start");
        uriProperties.put("end", "mock:result");
        route.loadCustomUriMap(uriProperties);
        // Build the route
        builder = route.Route(false);
        return builder.getContext();
    }
    protected RouteBuilder createRouteBuilder() {
        return builder;
    }
}
```

As is shown above, we are going to use the Camel Test framework to test the Route, which requires to use special mock endpoints to extract results during the unit testing. But the Route we have created does not use such mock endpoints. It uses the usual file and log components. As a solution for that, we will replace the file and log components with the direct:start and mock:result components. In lines 49 to 52, you can see a Map<String, String> which maps the component name to it's URI, and replaces values for start and end. By doing so we are able to mock the components to test the Route in isolation.

4. Click the button in the toolbar to run the test.
The test is successful.