PEtALS-SE-Transcoder

This document explain how to install, configure and use the petals-se-transcoder JBI component.

PEtALS Team
Anne-Marie Barthe <anne-marie.barthe@ebmwebsourcing.com>

- October 2007 -

Table of Contents

List of Figures

List of Tables

PEtALS-SE-TRANSCODER

The current component version (1.0) is based on the PEtALS Component Development Kit (also CDK) 3.0.

The Transcoder component is a JBI service engine. It works on data stored into a database (HSQLDB only for the current version of the component). It allows to :

- find a specific value in a table, helped with a key-value couple (translate operation)
- update a key-value couple
- create a key-value couple
- delete a key-value couple

The query parameters (table and column names, reference value) are given under an XML form (it will be detailled in the following chapters).

The connection parameters are defined into the jbi.xml file of the Transcoder Service Unit.

Prerequisiteries

You need to have a database (HSQLDB, MySQL, etc.), with at least a table (two columns minimum) filled with key-value couples.

You also need to start the SampleClient to use this component.

If you want to build the Transcoder from sources, you also need to use Maven (http://maven.apache.org).

Chapter 1. Features

When the Transcoder component starts, it registers and activates an endpoint in the JBI context.

By default, the endpoint name is generated using the container name like 'TranscoderEndpoint(INTERNAL):petals-se-transcoder, container1'. The service QName is '{http://petals.objectweb.org/}transcoderService'.

Four operations are provided by the endpoint :

- **translate** : searches the translation of a key-value couple for an other key, and returns the searched value
- **create** : creates a key-value couple
- **update** : updates a key-value couple
- **delete** : delete a key-value couple

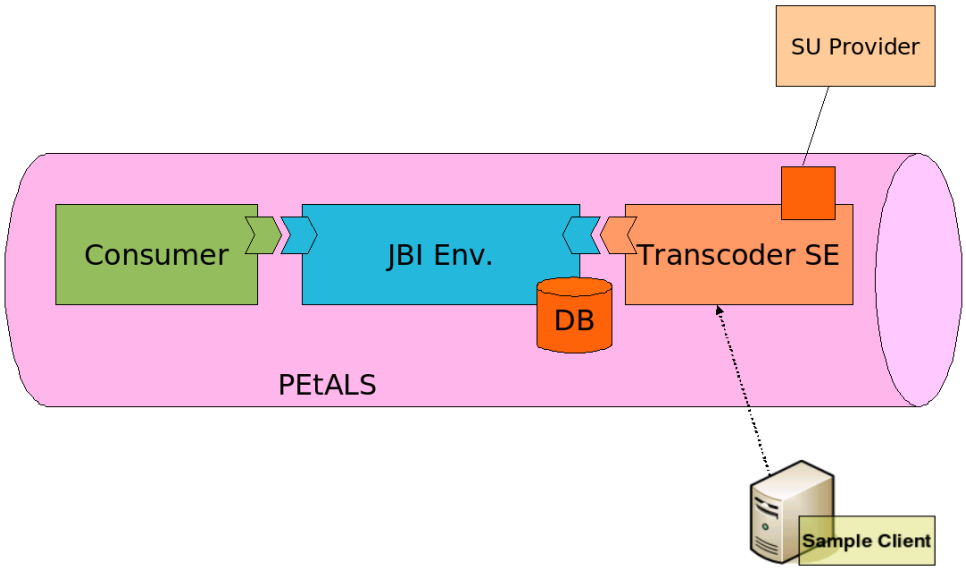
Chapter 2. Service Configuration

Since the Transcoder component is based on the CDK framework, you can easily deploy service units which will activate new JBI endpoints. This can be helpful since the component automatically activates randomly generated endpoint names at component startup.

2.1. Query the HSQLDB database

PROVIDE SERVICE : Expose query operations to consumers

Figure 2.1. Provides Transcoder service



2.1.1. Service Unit descriptor

```
<jbi:jbi xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:petals="http://petals.ow2.org/extensions"
xmlns:tns="http://petals.ow2.org/"
xmlns:jbi="http://java.sun.com/xml/ns/jbi" version="1.0">
<jbi:services binding-component="false">
<jbi:provides interface-name="tns:transcoderPT"
service-name="tns:transcoderService"
endpoint-name="TranscoderEndpoint">

<!-- URL of specific contract service if exists -->
<petals:wsdl>transcoder.wsdl</petals:wsdl>

<!-- Specific parameters for this service unit -->
<petals:params>
<petals:param name="driverClass">JDBC Driver</petals:param>
<petals:param name="urlDB">Address of the database</petals:param>
<petals:param name="usernameDB">Your user login</petals:param>
<petals:param name="passwordDB">Your user password</petals:param>
</petals:params>
</jbi:provides>
</jbi:services>
</jbi:jbi>
```

Table 2.1. Service Unit attributes to provide services

Attribute	Description	Default	Required
provides	Name of the JBI service that will be activated to expose Transcoder into the JBI environment. interface (qname), service (qname) and endpoint (string) name are required.		Yes
driverClass	The JDBC driver you will use to access to your database (ex : org.hsqldb.jdbcDriver)		Yes
urlDB	The address of the database to connect to. It can be relative to the root directory for a local database (ex: 'home/user/Database/test')		Yes
usernameDB	The username for user authentification ('sa' is default **for an HSQLDB database**)		Yes
passwordDB	The password for user authentification (" is default **for an HSQLDB database**)		Yes

Table 2.2. Advanced configuration of Service-Unit (provides elements)

Attribute	Description	Default	Required
wsdl	Path to a wsdl file describing services and operations offered by an endpoint activated by the SU. This extension is only usable with provides fields. The path can be a URL "http" or "file", or relative to the root directory of the SU archive. Ex : "file://home/user/test.wsdl" or "/WSDL/test.wsdl". If no wsdl path is specified, a simplified description will automatically be written by the CF.		

2.1.2. Service Unit content

The Transcoder Service Unit has to contain the following elements, packaged in an archive file :

- The META-INF/jbi.xml descriptor file, as described above
- An optional WSDL file, describing the related service

```
Service-unit.zip
  + META-INF
       - jbi.xml (as defined above)
  + service.wsdl (optional)
```

2.1.3. Usage

Once a provides node is configures, you can start to query the database via the Transcoder Component. You just have to send the queries to endpoints activated by the service unit deployments (containing jbi.xml with provides nodes).

Caution

Only InOut (for translate operation) and InOnly (for update/delete/create operations) exchange pattern are allowed.

Chapter 3. Samples

This section describes how to install (and use) the different components and service assemblies in order to test the Transcoder Service Engine component.

For each case, you must have installed (and run) :

- The Transcoder Service Engine component (download here)
- The Sample Client Service Engine component (download here)
- The sa-transcoder-provide Service Assembly (download here). This service assembly contains the su-transcoder-provide Service Unit which provides an endpoint

Caution

Configure the JBI Descriptor of the su-transcoder-provide Service Unit before installing it, to etablish a JDBC connection to the right database, with the right account.

- A database, with at least one two-column table, filled with key-values couples.

  For the following examples, we will use this table "Department", from the database "citycode" (the unique key is "DPTMTCODE"). This HSQL database is available into the resources directory of the sample SU component.

DPTMTCODE	DPTMTNAME	WEATHERCODE	CITY
01	Ain	FRXX0017	Bourg-en-Bresse
02	Aisne	FRXX0206	Laon
03	Allier	22135	Moulins
04	Alpes de Haute-Provence	FRXX0211	Digne
05	Hautes-Alpes	11752	Gap
06	Alpes-Maritimes	FRXX0073	Nice
07	Ardeche	FRXX0208	Privas
08	Ardennes	FRXX0205	Charleville Mezieres
09	Ariege	FRXX0204	Foix
10	Aube	FRXX0101	Troyes
...	...	...	...
85	Vendee	FRXX0158	La-Roche-Sur-Yon
86	Vienne	FRXX0170	Poitiers
87	Haute-Vienne	FRXX0119	Limoges
88	Vosges	FRXX0152	Epinal
89	Yonne	FRXX0008	Auxerre
90	Territoire-de-Belfort	FRXX0012	Belfort
91	Essonne	FRXX0201	Evry
92	Hauts-de-Seine	FRXX0202	Nanterre
93	Seine-Saint-Denis	FRXX0015	Bobigny
94	Val-de-Marne	FRXX0284	Creteil
95	Val-d Oise	26342	Pontoise

For our usecase, this type of table is used to translate the id (i.e. dptmtcode, here) of a department or the name of its main city into its weather code (for example, for a weather forecast webservice) to get the weather forecast for any french department.

The **xml string** to set into the **IN field** of the SampleClient is always like this :

```
<tra>
  <table-name>Name of the table</table-name>
  <column-one-name>Column name of the first key-value couple</column-one-name>
  <column-one-value>Value of the first key-value couple</column-one-value>
  <column-two-name>Column name of the second key-value couple</column-two-name>
  <column-two-value>Value of the secondd key-value couple</column-two-value>
</tra>
```

3.1. Translate operation

The name of this operation is `translate`. The right pattern to use is `InOut`.

The aim of this operation is to retrieve a value (`columnTwoValue`) thanks to a defined key-value couple (`columnOneName` and `columnOneValue`) and the name of the second key (`columnTwoName`). The query is like this : "`SELECT columnTwoName FROM tableName WHERE columnOneName='columnOneValue';`"

For our example, we want to retrieve the weathercode of a department thanks to the name of its main city . Our SQL query is like the following : "SELECT weathercode FROM department WHERE city='Nice';". It will return "FRXX0073" as a result.

So, our **XML query string** looks like :

```
<tra>
  <table-name>department</table-name>
  <column-one-name>city</column-one-name>
  <column-one-value>Nice</column-one-value>
  <column-two-name>weathercode</column-two-name>
  <column-two-value></column-two-value>
</tra>
```

Caution

The `<column-two-value></column-two-value>`tag is empty.

The **result** is returned in the **OUT field** of the SampleClient :

```
<result>FRXX0171</result>
```

3.2. Update operation

The name of this operation is `update`. The right pattern to use is `InOnly`.

The aim of this operation is to update a key-value couple (`columnOneName and columnOneValue`) thanks to an other key-value couple (`columnTwoName` and `columnTwoValue`). The query is like this : "`UPDATE tableName SET columnTwoName='columnTwoValue' WHERE columnOneName='columnOneValue';`"

For our example, we want to update the weathercode of a department. Our SQL query is like the following : "UPDATE department SET weathercode='FRXX0075' WHERE id='06';".

So, our **XML query string** looks like :

```
<tra>
  <table-name>department</table-name>
  <column-one-name>dptmtcode</column-one-name>
  <column-one-value>06</column-one-value>
  <column-two-name>weathercode</column-two-name>
```

```
  <column-two-value>FRXX0075</column-two-value>
</tra>
```

A **DONE STATUS** is returned in the SampleClient's **OUT field**.

3.3. Create operation

The name of this operation is `create`. The right pattern to use is `InOnly`.

The aim of this operation is to create a key-value couple. Two cases exist : the one where the unique key is automatically generated, the other where the key is generated by the user.

3.3.1. Unique key automatically generated

The query is like this : "INSERT INTO tableName (columnOneName) VALUES ('columnOneValue') ;"If the deparment id was automatically generated, the query would be like this : "INSERT INTO department (dptmtname) VALUES ('Guadeloupe');" (the weathercode can be add by using the update operation).So, our XML query string would be :

```
<tra>
  <table-name>department</table-name>
  <column-one-name>dptmtname</column-one-name>
  <column-one-value>Guadeloupe</column-one-value>
  <column-two-name></column-two-name>
  <column-two-value></column-two-value>
</tra>
```

Warning

In this case, the <column-two-name></column-two-name> and <column-two-value></column-two-value> tags are empty !

A **DONE STATUS** would be returned in the SampleClient's **OUT field**.

3.3.2. Unique key manually generated

The query is like this : "INSERT INTO `tableName` (`columnOneName`, `columnTwoName`) VALUES (`'columnOneValue'`,`'columnTwoValue'`) ;"

For our example, the SQL query is like this : "INSERT INTO `department` (`dptmtcode`, dptmtname) VALUES (`'97'`, `'Guadeloupe'`);".

So, our **XML query string** looks like :

```
<tra>
  <table-name>department</table-name>
  <column-one-name>dptmtcode</column-one-name>
  <column-one-value>97</column-one-value>
  <column-two-name>dptmtname</column-two-name>
  <column-two-value>Guadeloupe</column-two-value>
</tra>
```

Tip

You can also use this XML pattern to add directly two values if the unique key is automatically generated.

A **DONE STATUS** is returned in the SampleClient's **OUT field**.

3.4. Delete operation

The name of this operation is `delete`. The right pattern to use is `InOnly`.

The aim of this operation is to delete data associated to a key-value couple (`columnOneName and columnOneValue`). The query is like this : "`DELETE FROM tableName WHERE columnOneName='columnOneValue';`"

For our example, we want to delete the data linked to the department which id number (i.e. dptmtcode) is '06'. Our SQL query is like the following : "DELETE FROM department WHERE dptmtcode='06';".

So, our **XML query string** looks like :

```
<tra>
  <table-name>department</table-name>
  <column-one-name>dptmtcode</column-one-name>
  <column-one-value>06</column-one-value>
  <column-two-name></column-two-name>
  <column-two-value></column-two-value>
</tra>
```

Caution

The `<column-two-name></column-two-name>` and `<column-two-value></column-two-value>` tags are empty !

A **DONE STATUS** is returned in the SampleClient's **OUT field**.