PEtALS-BC-XQuare

This document explain how to install and configure the petals-bc-xquare JBI component.

PEtALS Team
Marc Dutoo <marc.dutoo@openwide.fr>
Guillaume Decarnin <guillaume.decarnin@openwide.fr>
Roland Naudin <roland.naudin@ebmwebsourcing.com>
Adrien Louis <adrien.louis@ebmwebsourcing.com>

- November 2007 -

Table of Contents

Preface

This processor builds upon the XQuare Bridge library (see web site, tutorial and reference documentation at http://xquare.objectweb.org/ in order to allow users to interact with databases through its four operations : `insert`, `query`, `storedQuery` and `sql`.

Chapter 1. Operations

1.1. insert

The `insert` operation lets the user provide in the content of the incoming JBI message an XML document defining data to insert or update in the target database record(s). This XMl document must match the XQuare mapping (mapping XML file and insertion XML schema file). Note that updating existing data or adding new records are both possible. Look at the XQuare Bridge reference documentation for more details.

The content of the response is (if no fault because of an internal exception) the single tag `<success/>`.

1.2. query

The `query` operation lets the user provide in the content of the incoming JBI message an XQuery statement wrapped within the `query` top tag. This XQuery must be XQuare compliant and consistent with the configured database and target mapping service.

The response is an XML document whose top tag is `<results>` and whose elements are specified by the XQuery's results formatting part. Note that modules (akin to XML / SQL views, defined in *.mod files provided in the service unit's deployment directory) can be imported and used in such queries. Look at the XQuery reference documentation for more details.

```
<query>
      for $pfsId in collection("routing")/routing[login/text()="b"]
      return <result>{$pfsId}</result>
</query>
```

1.3. storedQuery

The `storedQuery` operation lets the user provide as the content of an incoming JBI message the id of the stored parameterized query to be executed and the value of its parameters. Concretely it must consist in the `storedQuery` top tag with the `id` attribute and zero or more `param` sub-tags with the `name` attribute and the parameter value in the text section. Those names must be consistent with the parameters defined within the XQuery code of the stored query.

The processing of the stored query is then similar than the `query` operation.

Exemple of an incoming message:

```
<storedQuery id="1">
    <param name="userid">142</param>
</storedQuery>
```

1.4. sql

The "`sql`" operation allow to execute SQL statements. Concretely it must consist in the "`sql`" top tag with the sql statement(s). Each SQL statement must be separated by a `;` character. The last statement can be a query.

```
<sql>delete from users where userid ='142'</sql>
```

1.5. Configuration

The target database is configured through the SU extensions provided by a given service unit and uses the following properties:

- `url` : the JDBC URL of the targeted database. Required for accessing the database through JDBC. Ex. `jdbc:mysql://localhost:3306/myDB`

- `driver` : the driver to use for jdbc access. The driver library can be added in the ServiceUnit deployed
- `user` : the user required by the JDBC connection. Required for accessing the database through JDBC. Ex. `petals_samples`
- `password` : the password required by the JDBC connection. Required for accessing the database through JDBC. Ex. `petals_samples`

Additional properties for configuring XQuare Bridge:

- `mapping` : the name of the XQuare mapping file to be found in the service unit deployment directory Required. Ex. `auction.map`
- (not a property) The schema definition of insertion XML documents must be provided in the service unit's deployment directory, with the same prefix than the corresponding mapping file (ex. auction.xsd)

Configuring stored queries:

- `storedQuery`(multiple) a Xquery with parameters. The first storedQuery will be referenced as 1, the second as 2,... when they are called

```
<xquare:storedQuery>
    <![CDATA[
    declare variable $userid as xs:string external
    for $user in collection("users")/users[userid = $userid]
    return $user
    ]]>
 </xquare:storedQuery>
```

Additional properties for configuring DataSource pool:

- `maxActive` : The maximum number of active connections that can be allocated from this pool at the same time, or negative for no limit.
- `maxIdle` : The maximum number of connections that can remain idle in the pool, without extra ones being released, or negative for no limit.
- `minIdle` : The minimum number of connections that can remain idle in the pool, without extra ones being created, or zero to create none.
- `maxWait` : The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception, or -1 to wait indefinitely.
- `timeBetweenEvictionRunsMillis` : The number of milliseconds to sleep between runs of the idle object evictor thread. When non-positive, no idle object evictor thread will be run.

Chapter 2. Database polling - Not effective yet

The XQuare External listener "listens" to new data in any XQuareBC service unit's target database and builds upon the XQuery capabilities of the companion XQuareBCJBIProcessor in order to get them, format them as XML and send it as a message to the recipient service of the Consumes. The "listening" part is done by regularly looking for "new data" as the results of a configured SQL query.

WARNING: polling is not a best practice to integrate databases. Databases are usually application-driven, and those applications (or their users) should drive the "new data" lookup process. This implementation is provided as fitting the most generic needs.

2.1. How it works

This default implementation identifies "new data" by a given field that must be incremental - that is, be different and always increased in newly inserted records. The usual case for it is a record identifier or primary key, but it can work with other kind of fields as well, like timestamps. Concretely, it executes the configured "previousDelimiter" to get the old "current data delimiting value", then it executes the configured "curentDelimiter" to get the new "current data delimiting value" as available in the actual data. If they are different, the configured XQuery is then executed through the XQuareBCJBIProcessor and its results formatted as an XML document and sent as a JBI message to the configured JBI service. Finally the "previous data delimiting value" is updated to the new one.

Notes:

This requires a new table to be added to the target database, that will contain a single record with a single field being the last seen "maximum data delimiting value". This table may start empty, in which case the "`initDataDelimiter`" will be used the first time.

Before executing the XQuery, in its code the `##PREVIOUS_DATA_DELIMITER##` and `##CURRENT_DATA_DELIMITER##` expressions are replaced by the values found in the preceding executions of "previousDelimiter" and "currentDelimiter".

Many such listeners may be deployed in different service units to listen to different kinds of "new data" in the same database.

Since it uses the XQuareBCJBIProcessor's "query" capabilities, look at its documentation for more details on how it behaves.

2.2. Configuration

The datasource to the target database and the XQuareBCJBIProcessor's XQuery capabilities are configured through the XQuareBCJBIProcessor (look at its javadoc for details). The configuration specific to this listener is configured through the same `jbi.xml` file provided by a given service unit, and uses the following additional properties :

- `operation` : This service's operation. Ex. `sayHello`
- `xquare.dataListener.polling.period` : The period of the polling. Optional (default value : 1000 ms).
- `xquare.verifyTargetedService` : whether to verify the existence of the recipient service and operation (search the WSDL of the service). Possible values are `true` and `false`. Default value is `false`.
- `xquare.dataListener.delimiterStrategy` : The class name of the object who detects new data. Two strategies are released with XQuare BC: `org.ow2.petals.bc.xquare.consume.FileDelimiterNewDataStrategy` and `org.ow2.petals.bc.xquare.consume.DatabaseDelimiterNewDataStrategy`. A custom class may be used. The class must extend `org.ow2.petals.bc.xquare.consume.DelimiterNewDataStrategy` and be present in the Petals classpath.
- `xquare.dataListener.currentDelimiter` : An SQL query that must return the single value of the current data delimiter. Ex.:

```
select max(bid_date) from bids
```

- `xquare.dataListener.initialDelimiter` : The new data delimiter initial value if none in its own table and field. Ex. 1970-01-01

- `xquare.dataListener.xquery` : An XQuery that must return and format the current new data. Note that the ##PREVIOUS_DATA_DELIMITER## and ##NEW_DATA_DELIMITER## expressions are replaced within the SQL code before its execution. Ex.:

  for $i in collection("bids")/bids

  where $i/bid_date > xs:date("##PREVIOUS_DATA_DELIMITER##") and $i/bid_date <= xs:date("##CURRENT_DATA_DELIMITER##") return

  <bid>{ $i/bid_date }</bid>

Specific configuration for DatabaseDelimiterNewDataStrategy:

- `xquare.dataListener.previousDelimiter` : An SQL query that must return the single value of the saved, previous data delimiter. Ex.:

```
select max(bid_date) from bids_xqbc
```

- `xquare.dataListener.updateDelimiters` : An SQL insert that must update the value of the "new data" delimiter from "previous" to "current". Note that the ##CURRENT_DATA_DELIMITER## expression is replaced within the SQL code before its execution. Ex.:

```
delete from bids_xqbc;

insert into bids_xqbc (bid_date) values ("##NEW_DATA_DELIMITER##")
```

Specific configuration for FileDelimiterNewDataStrategy:

- `xquare.datalistener.delimiterFolder` (optional): The folder used to store the previousDelimiter. The default value is `{petals-folder}/xquare-bc-polling-files`.

- `xquare.datalistener.delimiterFilename` (optional): The filename used to store the previousDelimiter. The default value is `ServiceName.properties`.