Jetty12

How to use BTM as the transaction manager in Jetty 6.x

These instructions have been verified against BTM 1.1.

> **Jetty 6 JNDI support**
>
> Before you can get the transaction manager and datasources via JNDI ENC (URLs starting with `java:comp/`), you first need to configure Jetty with support for it.
> See Jetty's JNDI documentation page.

Contents

- Step 1: Copy the BTM jars
- Step 2: Configure BTM as the transaction manager
- Step 3: Configure DataSources that are transaction aware
- Step 4: Configure datasources references in your web.xml

Step 1: Copy the BTM jars

Copy the following jars from the BTM distribution to the jetty6 `lib/` directory:

- btm-1.1.jar
- geronimo-spec-jta-1.0.1B-rc4.jar
- slf4j-jdk14.jar (*or any other one available here*)

Step 2: Configure BTM as the transaction manager

Copy the following into your jetty config file:

```
<Call name="getConfiguration"
class="bitronix.tm.TransactionManagerService
s">
  <Set name="serverId">jetty-btm-node0</Set>
  <Set
name="logPart1Filename"><SystemProperty
name="jetty.home" default="."
/>/work/btm1.tlog</Set>
  <Set
name="logPart2Filename"><SystemProperty
name="jetty.home" default="."
/>/work/btm2.tlog</Set>
</Call>
<New
class="org.mortbay.jetty.plus.naming.Transac
tion">
  <Arg>
    <Call name="getTransactionManager"
class="bitronix.tm.TransactionManagerService
s" />
  </Arg>
</New>
```

Hint

This will make the transaction manager available under this JNDI URL: **java:comp/UserTransaction**.

Step 3: Configure DataSources that are transaction aware

The easiest way to do this is to use the `DataSource` that ship with BTM.

> **ordering of XML elements**
>
> It is recommended that the datasources definitions appear before the transaction manager's one in Jetty's XML configuration. If you don't, recovery will be run on *all known datasources each time a new one is created* instead of just once at transaction manager's startup.
> If this is problematic for you, please let us know by voting for BTM-4.

Here's an example of using BTM with a `DataSource` that implements `javax.sql.XADataSource`:

```
<New id="mydatasource"
class="org.mortbay.jetty.plus.naming.Resourc
e">
  <Arg>jdbc/mydatasource</Arg>
  <Arg>
    <New
class="bitronix.tm.resource.jdbc.PoolingData
Source">
      <Set
name="className">org.apache.derby.jdbc.Embed
dedXADataSource</Set>
      <Set
name="uniqueName">mydatasource</Set>
      <Set name="minPoolSize">0</Set>
      <Set name="maxPoolSize">5</Set>
      <Get name="driverProperties">
        <Put
name="databaseName">testdb</Put>
        <Put
name="createDatabase">create</Put>
      </Get>
      <Call name="init" />
    </New>
  </Arg>
</New>
```

The `bitronix.tm.resource.jdbc.PoolingDataSource` implements `javax.sql.DataSource` and interacts with the `javax.sql.XADataSource` provided in this instance by Derby.

> ✅ **Hint**
>
> This datasource will be available under this JNDI URL: **java:comp/env/jdbc/mydatasource**.

If your database vendor does not provide an `XADataSource`, you can use BTM's `bitronix.tm.resource.jdbc.lrc.LrcXADataSource` as the `XADataSource` to allow your database connections to be controlled by the transaction manager:

```
<New id="example-nonxads"
class="org.mortbay.jetty.plus.naming.Resourc
e">
  <Arg>jdbc/exampleNonXADS</Arg>
  <Arg>
    <New
class="bitronix.tm.resource.jdbc.PoolingData
Source">
      <Set
name="className">bitronix.tm.resource.jdbc.l
rc.LrcXADataSource</Set>
      <Set
name="uniqueName">exampleNonXADS</Set>
      <Set name="minPoolSize">0</Set>
      <Set name="maxPoolSize">5</Set>
      <Get name="driverProperties">
        <Put
name="driverClassName">org.apache.derby.jdbc
.EmbeddedDriver</Put>
        <Put
name="url">jdbc:derby::example;create=true</
Put>
      </Get>
      <Call name="init" />
    </New>
  </Arg>
</New>
```

> ✅ **Hint**
>
> This datasource will be available under this JNDI URL: **java:comp/env/jdbc/exampleNonXADS**.

Again, we've used Derby as an example, but as the `LrcXADataSource` uses only the class name and url of a `java.sql.Driver`, you can use it with any database providing a JDBC driver.

Step 4: Configure datasources references in your web.xml

Before your code can access configured datasources via JNDI ENC URLs, you need to declare resource references in your `web.xml`:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <resource-env-ref>

<resource-env-ref-name>jdbc/mydatasource</re
source-env-ref-name>

<resource-env-ref-type>javax.sql.DataSource<
/resource-env-ref-type>
  </resource-env-ref>

  <resource-env-ref>

<resource-env-ref-name>jdbc/exampleNonXADS</
resource-env-ref-name>

<resource-env-ref-type>javax.sql.DataSource<
/resource-env-ref-type>
  </resource-env-ref>
</web-app>
```

Now you can perform JNDI lookups on those URLs to access the configured datasources:

```
DataSource exampleNonXADS =
ctx.lookup("java:comp/env/jdbc/exampleNonXAD
S");
DataSource mydatasource =
ctx.lookup("java:comp/env/jdbc/mydatasource"
);
```