Jetty

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

These instructions have been verified against BTM 2.0.1.

> **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 or have a look at the appendix to get instructions to quickly enable this in the default Jetty config.

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
- Appendix: Configure Jetty's JNDI ENC

Step 1: Copy the BTM jars

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

- btm-2.0.1.jar
- geronimo-jta_1.0.1B_spec-1.0.1.jar
- slf4j-api-1.6.0.jar
- btm-jetty6-lifecycle-2.0.1.jar

Also copy the following jars in the jetty6 `lib/ext/` directory:

- slf4j-log4j12-1.6.0.jar (*available from here*)
- log4j-1.2.15.jar (*download it from here*)

You will also need to copy your JDBC driver's JAR file in that same folder. In this example, we've used Derby 10.3.2.1 so we copied `derby.jar` in Jetty's `lib/ext/` directory.

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>

<Call name="addLifeCycle">
  <Arg>
    <New
class="bitronix.tm.integration.jetty6.BTMLif
eCycle"/>
  </Arg>
</Call>

<New
class="org.mortbay.jetty.plus.naming.Transac
tion">
  <Arg>
    <Call name="getTransactionManager"
class="bitronix.tm.TransactionManagerService
s" />
  </Arg>
</New>
```

Do not forget to create the `work/` directory in Jetty's home folder !

> ✅ **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.

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">../work/db1</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:../work/db2;create=tru
e</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 = (DataSource)
ctx.lookup("java:comp/env/jdbc/exampleNonXAD
S");
DataSource mydatasource = (DataSource)
ctx.lookup("java:comp/env/jdbc/mydatasource"
);
```

and you can do JNDI lookups on this URL to access the transaction manager:

```
UserTransaction ut = (UserTransaction)
ctx.lookup("java:comp/UserTransaction");
```

Appendix: Configure Jetty's JNDI ENC

Here is a quick guide to get the JNDI ENC (ie: JNDI URLs starting with `java:comp/`) configured in Jetty's default `jetty.xml`.

Search for this XML snippet:

```
<Call name="addLifeCycle">
  <Arg>
    <New
class="org.mortbay.jetty.deployer.WebAppDepl
oyer">
      <Set name="contexts"><Ref
id="Contexts"/></Set>
      <Set name="webAppDir"><SystemProperty
name="jetty.home"
default="."/>/webapps</Set>
      <Set
name="parentLoaderPriority">false</Set>
      <Set name="extract">true</Set>
      <Set
name="allowDuplicates">false</Set>
      <Set
name="defaultsDescriptor"><SystemProperty
name="jetty.home"
default="."/>/etc/webdefault.xml</Set>
    </New>
  </Arg>
</Call>
```

and replace it with this one:

```
<Array id="plusConfig"
type="java.lang.String">

<Item>org.mortbay.jetty.webapp.WebInfConfigu
ration</Item>
```

```
<Item>org.mortbay.jetty.plus.webapp.EnvConfi
guration</Item>

<Item>org.mortbay.jetty.plus.webapp.Configur
ation</Item>

<Item>org.mortbay.jetty.webapp.JettyWebXmlCo
nfiguration</Item>

<Item>org.mortbay.jetty.webapp.TagLibConfigu
ration</Item>
</Array>

<Call name="addLifeCycle">
  <Arg>
    <New
class="org.mortbay.jetty.deployer.WebAppDepl
oyer">
      <Set name="contexts"><Ref
id="Contexts"/></Set>
      <Set name="webAppDir"><SystemProperty
name="jetty.home"
default="."/>/webapps</Set>
      <Set
name="parentLoaderPriority">false</Set>
      <Set name="extract">true</Set>
      <Set
name="allowDuplicates">false</Set>
      <Set
name="defaultsDescriptor"><SystemProperty
```

```
name="jetty.home"
default="."/>/etc/webdefault.xml</Set>
      <Set name="configurationClasses"><Ref
id="plusConfig"/></Set>
```

```
    </New>
  </Arg>
</Call>
```

Now all web applications deployed from the `webapps/` folder will be able to lookup objects from JNDI using `java:comp/`-like names.