This page last changed on Mar 13, 2011 by patrik_nordwall.

Pure EJB3 Tutorial

Sculptor provides an EJB3 target implementation without any Spring dependencies. This tutorial describes how to setup projects, explore it with junit tests and finally deploy and try in JBoss.

Setup Projects

In this first part we will setup the project structure for maven and eclipse.
It consists of the following projects:

- helloworld-parent - Only a maven project for building the other parts.
- helloworld - Business tier. EJB project containing the services and domain objects.
- helloworld-ear - EAR package of the deployable application.

1. Create helloworld-parent project with the following command (one line).

mvn archetype:generate -DarchetypeGroupId=org.fornax.cartridges -DarchetypeArtifactId=fornax-cartridges-sculptor-archetype-parent -DarchetypeVersion=2.0.0 -DarchetypeRepository=http://fornax-platform.org/nexus/content/repositories/public

Fill in groupId and artifactId:

```
Define value for groupId: : org.helloworld
Define value for artifactId: : helloworld-parent
Define value for version:  1.0-SNAPSHOT: :
Define value for package:  org.helloworld: :
```

2. Create helloworld project with the following command. It is only the archetypeArtifactId that differs from previous command.

mvn archetype:generate -DarchetypeGroupId=org.fornax.cartridges -DarchetypeArtifactId=fornax-cartridges-sculptor-archetype-pure-ejb3 -DarchetypeVersion=2.0.0 -DarchetypeRepository=http://fornax-platform.org/nexus/content/repositories/public

Fill in same groupId as previous, but use `helloworld` as artifactId.

3. Create helloworld-ear project with the following command. It is only the archetypeArtifactId that differs from previous command.

mvn archetype:generate -DarchetypeGroupId=org.fornax.cartridges -DarchetypeArtifactId=fornax-cartridges-sculptor-archetype-ear -DarchetypeVersion=2.0.0 -DarchetypeRepository=http://fornax-platform.org/nexus/content/repositories/public

Fill in same groupId as previous, but use `helloworld-ear` as artifactId.

4. Edit the created pom.xml in helloworld-ear directory. Remove the depencency to helloworld-web, which is located at the end of the pom file.

5. Edit the pom.xml in the helloworld-parent directory. Remove the web module and add the ear module. The modules section should look like this:

```
<modules>
    <module>../helloworld</module>
    <!-- <module>../helloworld-web</module> -->
    <module>../helloworld-ear</module>
</modules>
```

6. Back to the command prompt. Change directory to `helloworld-parent`. Run `mvn install`.

7. Run `mvn eclipse:eclipse` to create Eclipse projects.

8. Open Eclipse and import the projects.

Ordinary CRUD Service

In this part we will create and Entity with CRUD operations. We will explore it with JUnit testing using OpenEJB.

1. Modify the file named `model.btdesign` in the folder `src/main/resources/`. Use the following:

```
Application Helloworld {
    basePackage=org.helloworld

    Module milkyway {

        Entity Planet {
            scaffold
            String name key;
            String message;
         }

    }

}
```

2. Run `mvn clean install` to generate code and build. The JUnit test will fail.

3. Now it is your job to complete the failing JUnit test - PlanetServiceTest.
HSQLDB is used as in memory database when running JUnit. Add test data in `src/test/resources/dbunit/PlanetServiceTest.xml`

```
<?xml version="1.0" encoding="UTF-8"?>

<dataset>
  <PLANET id="1" name="Earth" message="Hello from Earth"
    LASTUPDATED="2006-12-08" LASTUPDATEDBY="dbunit" version="1" />
  <PLANET id="2" name="Mars" message="Hello from Mars"
    LASTUPDATED="2006-12-08" LASTUPDATEDBY="dbunit" version="1" />
</dataset>
```

When running the test the application is deployed in the embedded OpenEJB container. All that is taken care of by the `AbstractOpenEJBDbUnitTest` base class.

Messaging

Let us add a Message Driven Bean into the mix.

1. Add the following consumer to the same module as the `Planet` entity, i.e. in `model.btdesign`.

```
    Consumer PlanetConsumer {
        queueName=queue/addPlanet

    }
```

2. Generate by running: `mvn -Dfornax.generator.force.execution=true -o -npu generate-sources`

3. This time I provide the Junit test and your job is to develop the solution. Change the PlanetConsumerTest so that it looks like this:

```
@Test
public void testConsume() throws Exception {
    int countBefore = countRowsInTable(Planet.class);
    String message = "Jupiter";
    Destination replyTo = sendMessage(queue, message);
    waitForReply(replyTo);
    int countAfter = countRowsInTable(Planet.class);
    assertEquals(countBefore + 1, countAfter);
}
```

4. As I said, it is up to you to make this test green. The starting point of your coding is in `PlanetConsumerBean`. You need the `PlanetRepository`, which is injected in this way in `model.btdesign`:

```
    Entity Planet {
        scaffold
        String name key;
        String message;

        Repository PlanetRepository {
        }
    }

    Consumer PlanetConsumer {
        inject @PlanetRepository
        queueName=queue/addPlanet

    }
```

Remote and Local Interfaces

By default both local and remote interfaces are generated for each Service. You can adjust that with the hints "notRemote" and "notLocal".

```
Service PlanetFacade {
    hint="notLocal"
    sayHello => InternalPlanetService.sayHello;
}
```

```
    Service InternalPlanetService {
        hint="notRemote"
        String sayHello;
    }
```

Web Service

The third ingredient in this tutorial will be a web service, which we later will try in JBoss.

1. Define the web service in `model.btdesign`:

```
        Service PlanetWebService {
            webservice
            List<@PlanetDto> getAllPlanets;
        }

        DataTransferObject PlanetDto {
            String name required
        }
```

Note that when working with web services we must use Data Transfer Objects as parameters and return types.

Attributes in DTOs are by default optional, i.e. elements will be skipped if the value is null. It is possible to define `required` as done above to indicate minOccurs="1" in WSDL. It is also possible to use `nullable`, which means that `xs:nil` is sent when the value of the attribute is null.

2. Please complete the failing test, `PlanetWebServiceTest`. You will need to add some code in `PlanetWebServiceBean`. Once again you will need to inject the repository. (inject keyword must be placed after webservice keyword)

Deploy in JBoss

Well done! Let us try it for real in JBoss.

1. Install JBoss, as described here. Please replace hibernate jar files as described there.

2. By default HSQLDB is used as in memory database, i.e. the data will be wiped away after each JBoss shutdown. You might want to replace that with a persistent database as described here, but that can wait.

Let us start with HSQLDB. You need to add a mbean datasource in JBoss (server/default/deploy/Helloworld-ds.xml). You can copy the generated file from src/generated/resources/dbschema/Helloworld-ds.xml.

3. You also need to add the invalid message queue to a JBoss configuration file. Add the following to `server\default\deploy\jms\jbossmq-destinations-service.xml`.

```
  <mbean code="org.jboss.mq.server.jmx.Queue"
    name="jboss.mq.destination:service=Queue,name=helloworld.invalidMessageQueue">
   <depends optional-attribute-name="DestinationManager">jboss.mq:service=DestinationManager</
depends>
  </mbean>
```

4. Run `mvn clean install` from the `helloworld-parent` project.

5. Deploy the ear to JBoss. I recommend that you use the hot deployment script as described here. First you must remove all references to the non existing web project in `antbuild-ear.xml` files located in `helloworld-parent` and `helloworld-ear` projects.

Test It

To try the web service I recommend that you install the free SoapUI tool.

The initial WSDL url is: http://localhost:8080/myapp/PlanetWebService/WebDelegateEndPoint?wsdl

When you run the getAllPlanets request the response is empty, yes the database is empty.

Let us send in a message to store a new planet.

To send a message to JBoss you can use a main class like this.

```
/**
 * Simple main class to send a message to JBoss @ localhost.
 * To be able to run this class you have to add the following two jars before other jars
 * in the classpath:
 * <ul>
 * <li>jboss-5.1.0.GA/client/jbossall-client.jar</li>
 * <li>jboss-5.1.0.GA/client/log4j.jar</li>
 * </ul>
 */
public class SimpleSend {

    public static void main(String[] args) {

        String message = "Earth";
        if (args.length > 0) {
            message = args[0];
        }
        String queueName = "queue/addPlanet";

        QueueConnection queueConnection = null;

        try {

            // InitialContext for jboss
            Properties properties = new Properties();
            properties.put(Context.INITIAL_CONTEXT_FACTORY,
                    "org.jnp.interfaces.NamingContextFactory");
            properties.put(Context.PROVIDER_URL, "jnp://localhost:1099");
            InitialContext jndiContext = new InitialContext(properties);

            // lookup queue
            QueueConnectionFactory queueConnectionFactory =
                (QueueConnectionFactory) jndiContext.lookup("ConnectionFactory");
            Queue testQueue = (Queue) jndiContext.lookup(queueName);

            queueConnection = queueConnectionFactory.createQueueConnection();
            QueueSession queueSession = queueConnection.createQueueSession(false,
                    Session.AUTO_ACKNOWLEDGE);
            QueueSender queueSender = queueSession.createSender(testQueue);
            TextMessage textMessage = queueSession.createTextMessage();

            textMessage.setText(message);
            queueSender.send(textMessage);
            System.out.println("Message sent");

        } catch (NamingException nameEx) {
            System.out.println("Naming error: " + nameEx);
        } catch (javax.jms.JMSException jmsEx) {
            System.out.println("JMS Exception: " + jmsEx.toString());
        } finally {
            if (queueConnection != null) {
                try {
                    queueConnection.close();
                } catch (javax.jms.JMSException jmse) {
                    // ignore
                }
            }
        }
    }
}
```

To be able to run `SimpleSend` you have to add the following two jars before other jars in the classpath of the run confiuguration for SimpleSend:

- jboss-5.1.0.GA/client/jbossall-client.jar
- jboss-5.1.0.GA/client/log4j.jar

Run `SimpleSend`.

Open SoapUI again an execute the `getAllPlanets` request. Voilà! The planet is retrieved.

👈	2	3	4	5	6	7	8	9	👉