Configuring DataSources

Configuring DataSources in openejb.xml

The **<Resource>** element is used to configure a **javax.sql.DataSource**. It is also used to configure other resources like Timers, Topics, Queues. We will see some examples of using <Resource> to configure a DataSource.

The <Resource> element is designed after @Resource annotation and has similar attributes.

For example, this annotation in your bean:

```
@Resource(name = "myDerbyDatasource", type = javax.sql.DataSource.class)
```

Would map to a Resource declared in your openejb.xml as follows:

```
<Resource id="myDerbyDatasource" type="javax.sql.DataSource">
 . . . .
<Resource>
```

Note that in the xml element, the *type* value of *javax.sql.DataSource* can abbreviated to just *DataSource* as follows:

```
<Resource id="myDerbyDatasource" type="DataSource">
 . . . .
<Resource>
```

See Containers and Resources for a complete list of supported DataSource properties. See DataSource Password Encryption for information on specifying non-plain-text database passwords in your openejb.xml file. See Common DataSource Configurations for a list of the commonly used databases and their driver configurations.

You may also need data partitioning per customer or depending on any other business criteria. That's also an available feature. See Dynamic Datasource for more details.

JNDI names for configured DataSources

Example 1

```
<Resource id="Default JDBC Database" type="DataSource">
   . . . . . .
</Resource>
```

The global jndi name would be **java:openejb/Resource/Default JDBC Database**

Example 2

```
<Resource id="Derby Database"  type="DataSource">
  . . . . . .
</Resource>
```

The global jndi name would be **java:openejb/Resource/Derby Database**

Obtaining a DataSource

DataSource references in your ejb should get automatically mapped to the Resource you declare. The shortest and easiest rule is that **if your reference name matches a Resource in your openejb.xml, that's the one you get**. Essentially, the rules for mapping are as follows.

1. Name Attribute Match - @Resource with a name attribute matching the resource name gets that resource injected

2. Injected Name Match - variable name matching the resource name gets that resource injected
3. No Match - nothing matches a resource name, so the first resource available gets injected

There are various ways one could obtain a DataSource now. Lets take an example of Derby.

With a Resource declaration in your openejb.xml like this:

```
<Resource id="myDerbyDatabase"  type="DataSource">
  . . . . .
</Resource>
```

There are several possible ways to refer to it, as follows.

BY matching variable name to resource name

```
@Stateless
public class FooBean {
    @Resource DataSource myDerbyDatabase;
}
```

OR BY matching name

```
@Stateless
public class FooBean {
    @Resource(name="myDerbyDatabase")
    DataSource dataSource;
}
```

OR BY JNDI lookup

```
@Resource(name="myDerbyDatabase", type=javax.sql.DataSource.class)
@Stateless
public class FooBean {

    public void setSessionContext(SessionContext sessionContext) {
        DataSource dataSource = (DataSource) sessionContext.lookup("myDerbyDatabase");
    }

    public void someOtherMethod() throws Exception {
        InitialContext initialContext = new InitialContext();
        DataSource dataSource = (DataSource) initialContext.lookup("java:comp/env/myDerbyDatabase");
    }
}
```

OR

```
<resource-ref>
   <res-ref-name>myDerbyDatabase</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
 </resource-ref>
```

OR

```
<resource-ref>
   <res-ref-name>jdbc/myDerbyDatabase</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
 </resource-ref>
```

OR

```
<resource-ref>
   <res-ref-name>someOtherName</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <mapped-name>myDerbyDatabase</mapped-name>
 </resource-ref>
```