SDN Contribution

Integrating a Third-Party Java Data Objects Implementation into SAP NetWeaver 2004s – Exemplified Through intelliBO

Applies to:

This how-to guide applies to SAP NetWeaver Application Server 2004. It is also applicable to SAP NetWeaver 2004s.

Summary

This tutorial shows how to integrate Signsoft's third-party Java DataObjects implementation, intelliBO, into SAP NetWeaver 2004.

Author: Markus Küfer

Company: SAP AG

Created on: March 2006

Author Bio

Markus Küfer has a long record in the Java area, especially when it comes to object-relational mapping and the integration of such systems into the J2EE world. He served as SAP's representative in the JDO Java Specification Request expert group (JSR 243) and contributed to the design of the upcoming SAP EJB 3.0 implementation. He now imparts knowledge as a member of Platform Ecosystem's Market Development Engineering team.

Table of Contents

Introduction

Java Data Objects (JDO) is a Java standard that defines transparent object persistence. Persistent data, typically stored in relational databases, is manipulated through its object representation by plain Java classes thus allowing the application programmer to focus on the application domain model.

There is a SAP JDO implementation available that comes with SAP NetWeaver Application Server (SAP NetWeaver AS) 6.40. All SAP applications using JDO rely on this implementation of the JDO 1.01 standard. The SAP JDO implementation offers unique features that transcend the JDO standard, such as locking and the exploitation of the OpenSQL infrastructure with the Java dictionary, and also features like table buffering and statement caching.

Nevertheless, customers or ISVs may rely on vendor extensions made by other JDO providers that they consider indispensable. To satisfy their needs, third-party JDO implementations have to be utilized with SAP NetWeaver AS. Signsoft's award-winning intelliBO is one of the most popular commercial JDO implementations available.

This how-to guide deals with the integration of the Intellibo JDO runtime into SAP NetWeaver AS and related aspects. It is neither an introduction to JDO, nor a tutorial on the extensions or design time of intelliBO. After you read this guide, you should be able to customize the SAP NetWeaver AS/intelliBO integration to your needs, deploy it, and also adapt and run an existing JDO application, or use intelliBO to write a JDO application within SAP NetWeaver Development Studio from scratch.

Prerequisites

Systems, Installed Applications, and Authorizations

- ☐ SAP NetWeaver AS – Java 6.40
- ☐ Signsoft intelliBO 4.0 download and valid license key

Knowledge

- ☐ Basic to sound JDO knowledge

Preparations

After you downloaded the intelliBO jar file, start the installation by:

```
java –jar <path>\ibo40.jar
```

The only dialogue will ask you to specifiy the location you want intelliBO to be installed at.

Having done this, you have to configure intelliBO:

Specify the Java Path

You will find a file named `shared.config` located in the `bin` directory of your intelliBO installation, which is `C:\Signsoft\intelliBO4.0\bin` by default for a Windows installation if you adhered to the proposed value during installation

There, adapt the path to your Java executable, which might look something like:

shared.config

```
java-path = C:\j2sdk1.4.2_09\bin\java.exe
vm-param  = -Xverify:none
```

License

The next step is to activate your license. This can be done in several ways. It is very convenient to use the following command line:

```
C:\Signsoft\intelliBO4.0\bin>ibolic.exe -nogui -add <your license key string>
```

In the following chapters, you will learn to integrate the intelliBO JDO implementation into the SAP J2EE Engine as well as how to use it in a sample application, including table creation.

Creating an intelliBO Resource Adapter

The intelliBO resource adapter integration enables the enlistment of connections, i.e. PersistenceManagers into the application server's JTA transactions as well as the pooling of PersistenceManager instances.

intelliBO integrates with the SAP J2EE servers via the Java Connector Architecture (JCA) standard. Therefore, it is packed and deployed to the server as a .rar file implementing a JCA-compliant resource adapter. intelliBO comes with no ready-to-run resource adapter, but with an ant script that allows building the resource adapter.

intelliBO also integrates server-specific using the so-called native integration that directly interacts with the server's TransactionManager in a non-standardized, proprietary way.This is intended for servers that do not offer complete JCA support. As the SAP J2EE server is fully compliant to the J2EE 1.3 standard, there is no need for such an integration mode.

The intelliBO installation ant scripts are targeting multiple platforms and settings; the script is quite generic and complex but has to be adopted anyway. This how-to guide therefore privides you with a small script that will exactly suit our needs. Simply extract the attached `intellibo-howto.zip` file and open the `build.xml` that is contained int the top-level folder to adjust those parameters:

- intellibo.dir

  Set the value to the intelliBO installation directory, `C:/Signsoft/intelliBO4.0`, for example.

- license.dir

  Set the value to the folder that contains the `license.xml` file, `C:/Signsoft/intelliBO4.0/lib`, for example.

Connection Management and Transaction Support

For this tutorial, the resource apdapter's `transaction-support` attribute is set to `LocalTransaction`. This is specified in the `ra.xml` deployment descriptor required by the JCA specification. The `ra.xml` can be found at: **<**`intellibo-howto.zip`**>/**`rar/META-INF`.

Note that currently the XA transaction-support (transaction-support: `XATransaction`*) does not properly work for the integration of intelliBO into the SAP J2EE server due to different interpretations of the JCA standard. Therefore, we will use the local transaction-support throughout this tutorial.*

To persist PersistenceCapable instances, intelliBO uses a relational datastore. intelliBO therefore needs to acquire database connections. intelliBO can do this twofold: Either using the servers's JDBC connection pool or using its internal JDBC connection management. In this tutorial, we let intelliBO control the underlying JDBC connections. In general, in a managed server environment it is reasonable to have one central (JDBC) connection pooling for effective resource management. But: Only one resource that supports merely local transactions is allowed to participate in a global (JTA) transaction. As the intelliBO resource adapter is declared to support local transactions only, we would either have intelliBO using a XA datasource additionally deployed on the server. But due to the different interpretation of the JCA standard, this is a critical thing: The intelliBO JDO implementation might rely on a certain behaviour which is not guaranteed by the server's JCA implementation for XA resources. Therefore, we let intelliBO manage its JDBC connections itself:

We decare intelliBO to use the plain SAPDB driver without any XA features or OpenSQL features, because intelliBO is generating platform specific SQL statements which might not be executable on SAP's OpenSQL database abstraction layer (being a subset of SQL 92 only).

`ibo-config.xml` is the descriptor that defines the way intelliBO accquires JDBC connections. `ibo-config` is found at:

<`intellibo-howto.zip`**>**`\workspace\lib`.

We simply reuse the J2EE engine's schema, which is MDE in this case. Simply adopt the underlined parameters (database URL, username, password) to your schema[1].

Part of ibo-config.xml

```
      <node name="resource-entry">

            <entry key="type" value="jdbc" />

            <entry key="name" value="SJDO_DEFAULT" />

            <entry key="driver-name"
value="com.sap.dbtech.jdbc.DriverSapDB" />

            <entry key="url"
value="jdbc:sapdb://localhost/MDE?spaceoption=true&amp;unicode=yes" />

            <entry key="username" value="SAPMDEDB" />
```

[1] In case the database vendor differs, change `driver-name` and `dbsupport-name` accordingly, too.

```
            <entry key="password" value="abc123" />

            <entry key="dbsupport-name"
value="com.signsoft.ibo.dbsupport.sapdb.SAPDBDatabaseInfo" />

            <entry key="transactional" value="true" />

            <node name="pool-configuration">

                  <entry key="initial-size" value="1"/>

                  <entry key="maximum-size" value="5"/>

            </node>

      </node>
```

Logging

Besides the declaration of the relational store, `ibo-config.xml` is also the place to force intelliBO to tell a bit about its internal workflow and state. This might help you if you run into errors or to understand how intelliBO actually persists objects to the database, if you want to go into details. Here, you can switch on and off the logging for different domains and define where to log to. This is how a log declaration could look like:

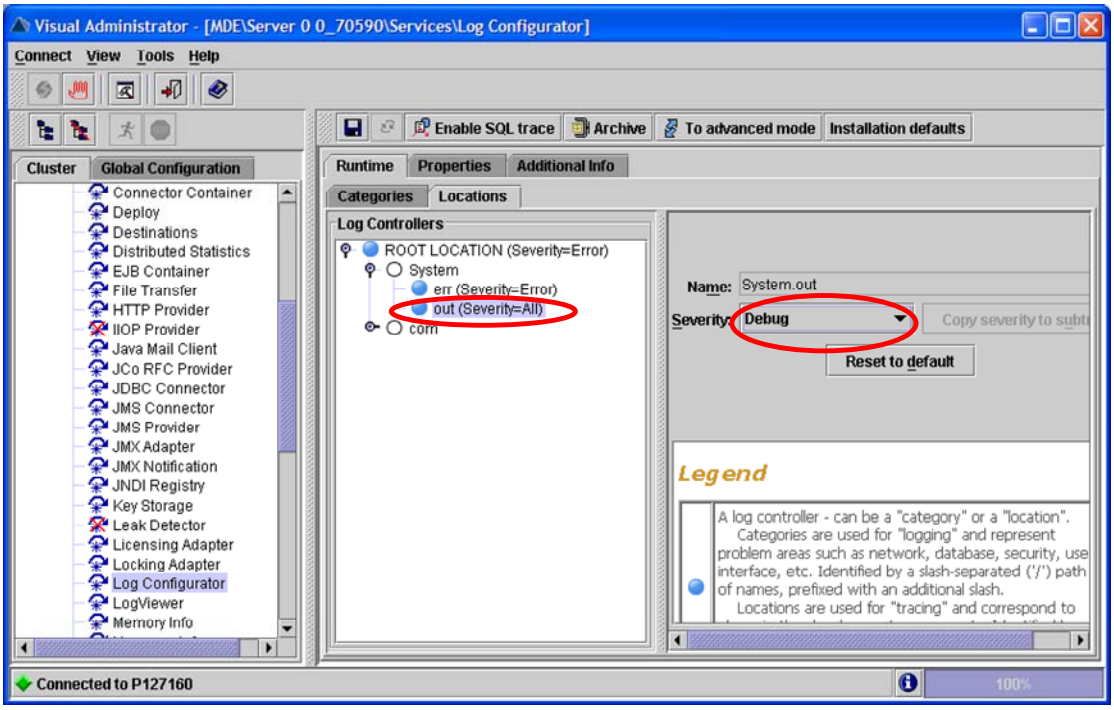
Part of ibo-config.xml

```
      <logging enabled="true" >

            <device type="sysout" />

            <domain name="IBO.SQL" />

            <domain name="IBO.CORE" />

      </logging>
```

This will cause the domains IBO.SQL and IBO.CORE to be logged to the default output stream.

If you log to sysout, i.e. the “standard” output stream, you have to make sure you to switch on the logging for the standard output stream in the Visual Administrator, too[2]:

[2] Please not that this is not intended to be used in productive mode.



Additional JDO Properties

`ibo-config.xml` is also the place to additionally overwrite the default values of JDO properties that are delared in the JDO specification. For example, you may use optimistic transactions (our demo application will use this feature).

Part of ibo-config.xml

```
<node name="environment">
      <entry key="optimistic" value="true" />
      <entry key="non-transactional-read" value="true" />
      <entry key="non-transactional-write" value="false" />
</node>
```

Building the intelliBO Resource Adapter

Now that we adjusted the intelliBO descriptor files, the intelliBO resource adapter can finally be build.

The already introduced `build.xml`[3] comes with three targets:

- `ra` builds the intelliBO resource adapter for the SAP Web Application Server.
- `ra.clean` deletes all the libraries that where created running the `ra` target
- `ra.all` will first run `ra.clean` and then `ra`. This is the ant script's default target.

Open a DOS console in:

`<intellibo-howto.zip extraction location>``\workspace\`

and type **`ant`**. The resulting `ibo.rar` will be found at this path, too.

With the SAP Web Application Server, .rar files cannot be directly deployed via the Visual Admin. To deploy the resource adapter, start your server and open the telnet console: Open a DOS console and type: **`telnet localhost <telnetPort>`**[4] Then log in with the Administrator password you would also use for the Visual Admin, type **`jump 0`**, press return and then **`add deploy`**, followed by another return. The actual command to deploy the resource adapter is: **`deploy <intellibo-howto.zip extraction location>\workspace\ibo.rar`**.

Now that the intelliBO Resorce adapter is up and running on SAP NetWeaver, let's test it with a sample application.

Using the intelliBO Resource Adapter: The Sample Application

Additionally to providing a ready-to-run EAR file, this how-to guide comes with sample NWDS projects that demonstrate how to use intelliBO within the NWDS at design-time. This includes:

- to create table schemas,
- to verify the consistency of JDO persistence metadata and Java class
- and to enhance the Java classes to persistence-capable classes.

Several sample applications come with the intelliBO installation, one of them deals with the J2EE server integration. We use this sample for this demonstration. It consists of a simple remote stateless session bean named "PersonManagerBean" with two persistence-capable classes, "Address" and "Person", whereas there is a unidirectional 1:1 relationship from Person to Address. The sample uses intelliBO's primary key generation and does only specify the JDO metadata, i.e. information on which fields are persistent. The mapping metadata, i.e. information on how a certain attribute of a persistence-capable class is mapped to the database, is not specified and therefore done by intelliBO transparently. A JSP page allows creating, updating and deleting entries in a list of persons. For your convenience, the sample application used comes as a complete SAP NetWeaver Developer Studio (NWDS) project thereby allowing to build tables, enhance classes, and deploy the application right out of the NWDS.

Sample application consists of three NWDS projects:

- *intelliBO*: The EJB project containing the session bean and the persistence-capables.
- *intelliBOWEB*: The Web project that contains a single JSP.

---

[3] Note that there's another build.xml for the sample application that is described in the next chapter.

[4] The telnet port is 50008 usually.

- *intelliBOEAR*: The Enterprise Archive project to assemble the EAR file.

Importing the Sample Projects

The prerequisite to import the sample projects is that you extracted the complete how-to guide as described.

Import these projects into your NWDS using *File* → *Import…* → *Multiple Existing Projects into Workspace*. For the *Base Folder* entry, browse to **`<intellibo-howto.zip extraction location>`** or enter it in the text field. In *Options*, select *Search for Projects in all Subfolders*. Clicking *Finish* will then result in the three projects being imported.

Configuring the intelliBO Design Time Support

The intelliBO design time support is provided via a separate ANT script that comes with the EJB project. You will find this `build.xml` on top-level of the *intelliBO* project. It is done in a way that allows for reuse in other projects as well.

Property	Description
`intellibo.dir`	The Signsoft installation folder
`ide.dir`	Location of the SAP NetWeaver Developer Studio
`message.level`	Severity level of intelliBO tools' messages

The other properties do not have to be changed as long as you stick to the imported projects' structure. Once you use the script for your own projects, you may want to adapt:

Property	Meaning
`add.jars`	Additional JARs needed for enhancement like ejb2.0.jar.
`src.dir`	Base folder of packages containing JDO metadata
`bin.dir`	Base folder containing the .class files
`tables.dest.dir`	Target folder for the generated datab base schemas

Creating Tables

The table descriptions necessary are also created via the ANT script. To generate database platform-specific table descriptions, the `db-config.xml` in the `tables` subfolder of the *intelliBO* project is used to describe the database support (*supportBase*) required. In our sample, we use SAPDB:

Part of db-config.xml

```
<!-- can also be used by the verifier to check against the db!-->
<node name="db-config">
  <entry key="default" value="samples"/>
  <node name="connection">
    <entry key="name" value="samples"/>
```

```
    <entry key="description" value="Signsoft intelliBO samples"/>

    <entry key="driver" value="com.sap.dbtech.jdbc.DriverSapDB"/>

    <entry key="url"
value="jdbc:sapdb://localhost/MDE?spaceoption=true&amp;unicode=yes"/>

    <node name="properties">

      <entry key="user" value="SAPMDEDB"/>

      <entry key="password" value="abc123"/>

    </node>

    <entry key="supportBase" value="dbsupport_sapdb"/>

    <entry key="supportDriver"
value="com.signsoft.ibo.dbsupport.sapdb.SAPDBDatabaseInfo"/>

  </node>

</node>
```

Just in case you wonder why the complete set of database connect parameters is given: db-config.xml is also used to verify the consistency of the database against the JDO mapping metadata. For us there's no need to check against the database, as we leave both the creation of the JDO mapping metadata and the database schema creation up to intelliBO. But you might take advantage of it in your own NWDI/intelliBO projects. If so, adapt the underlined properties to your need and adapt the examples.verify target in the build.xml file accordingly.

To run the schema generation, use the Package Explorer. In the *intelliBO* project, navigate to `tables/build.xml` and right-click it. Select *Run Ant…* In the list of build targets, flag *examples.schemagen* and press *Run*.

The generated scripts can be found in the tables folder underneath the *intelliBO* project.

For example, here is the create.sql's content:

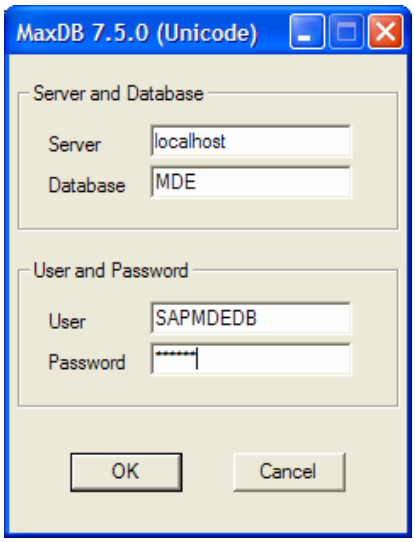
create.sql

```
CREATE TABLE TADDRESS (FID INTEGER, FSTREET VARCHAR(200), FCITY
VARCHAR(200), PRIMARY KEY (FID));

CREATE TABLE TPERSON (FID INTEGER, FAGE INTEGER, FNAME VARCHAR(200),
FTADDRESS_FID INTEGER, PRIMARY KEY (FID));

CREATE TABLE TSEQUENCE (FNAME VARCHAR(200), FKEY INTEGER);



INSERT INTO TSEQUENCE (FNAME, FKEY) VALUES ('default', 100);
```

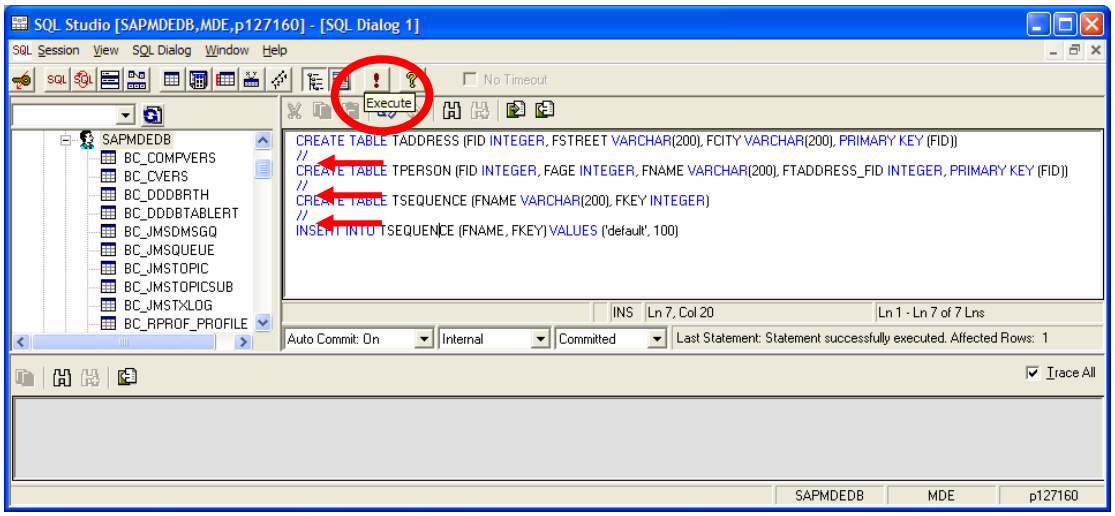
For SAPDB, you have to take the following steps to create tables with the scripts generated:

Make sure your database is up and running. If you have the NetWeaver J2EE server running and you stick to our model to use the J2EE server's database to hold the application's tables, you can directly open SAPDB's SQL Studio (*start → All Programms → MySQL MaxDB → SQL Studio*).

To connect to the database, press the upper left yellow key icon and enter your connection data data:



Once connected to the database, copy the content of create.sql into the SQL dialog window, replace the semicolons at the end of each statement with a comment in a new line and press the exclamation mark icon to execute the tables' creation:



The same procedure can be applied to drop those tables using the `drop.sql` script generated along with the `create.sql` file.

Automating the Enhancement of JDO Classes: One-Step Compile and Enhance

As mentioned above, it is possible to seamlessly integrate intelliBO's verification and enhancement mechanisms into the NWDS. We will now set up a feature to automatically verify and enhance the classes on every incremental or full build, as you wish. We use NWDS's underlying Eclipse features to add external tools (like Ant) to Eclipse's internal build order.

In the sample NWDS intelliBO project, this intelliBO integration is already done for your convenience. In case you want to try it on your own projects, these are the steps to take:

Right-click the *intelliBO* project, select *Properties*. Now select *External Tools Builder* in the right navigation list and click the *New…* button. A window titled *Choose configuration type* pops up. Select the *Ant Build* here and press the *OK* button. This will result in a subsequent pop-up. Here, we have to describe the external Ant build:

- *Main* tab: Fill in a name, like *intelliboVerifierAndEnhancer*. *Location* tells the `build.xml`'s location. Browse for the *intelliBO* project and select the `build.xml`. You may also want to flag *Run tool in background* or *Capture output*.
- *Targets* tab: Flag the *build* target.[5]
- *Build Options* tab: Select *Full builds* or *Incremental builds*.

Once you are through those tabs, click *OK*.

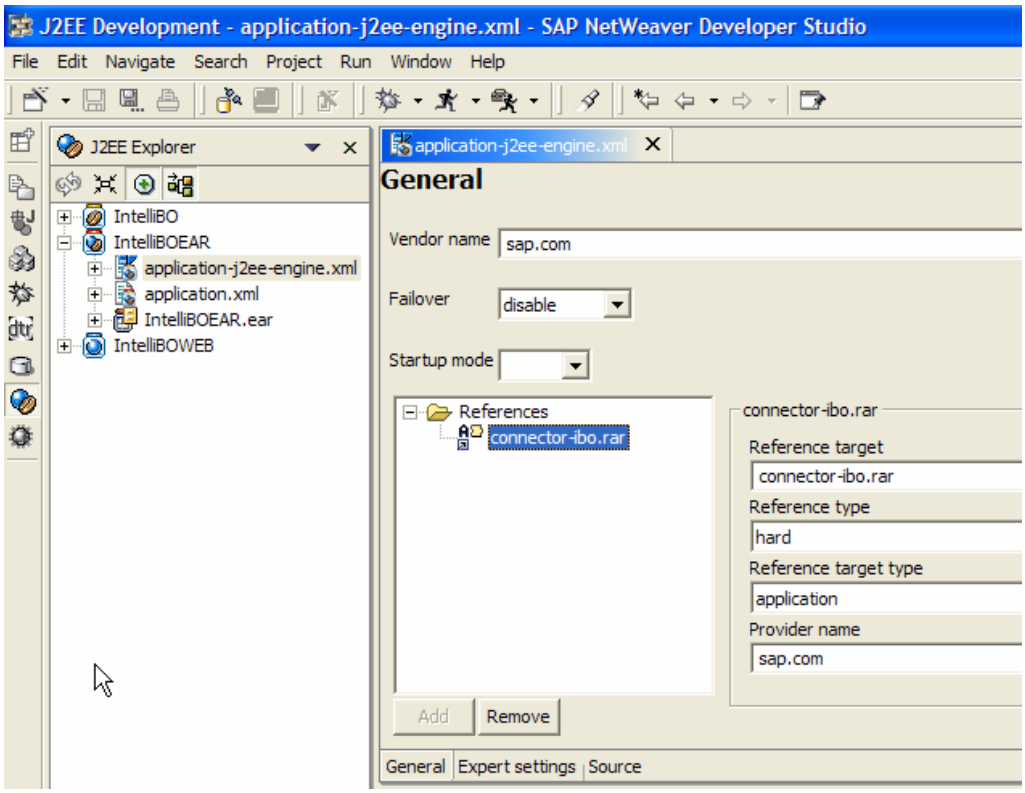
Saving Java files will now automatically trigger the JDO verification and enhancement.

Declaring the Application's Runtime Dependencies

As we integrated intelliBO JDO into the server's resource management system via JCA, there is no need to include any JDO or intelliBO specific classes. But we have to declare that our application will make use of the intelliBO resource adapter. Therefore, we have to declare a reference in the `application-j2ee-engine.xml`:

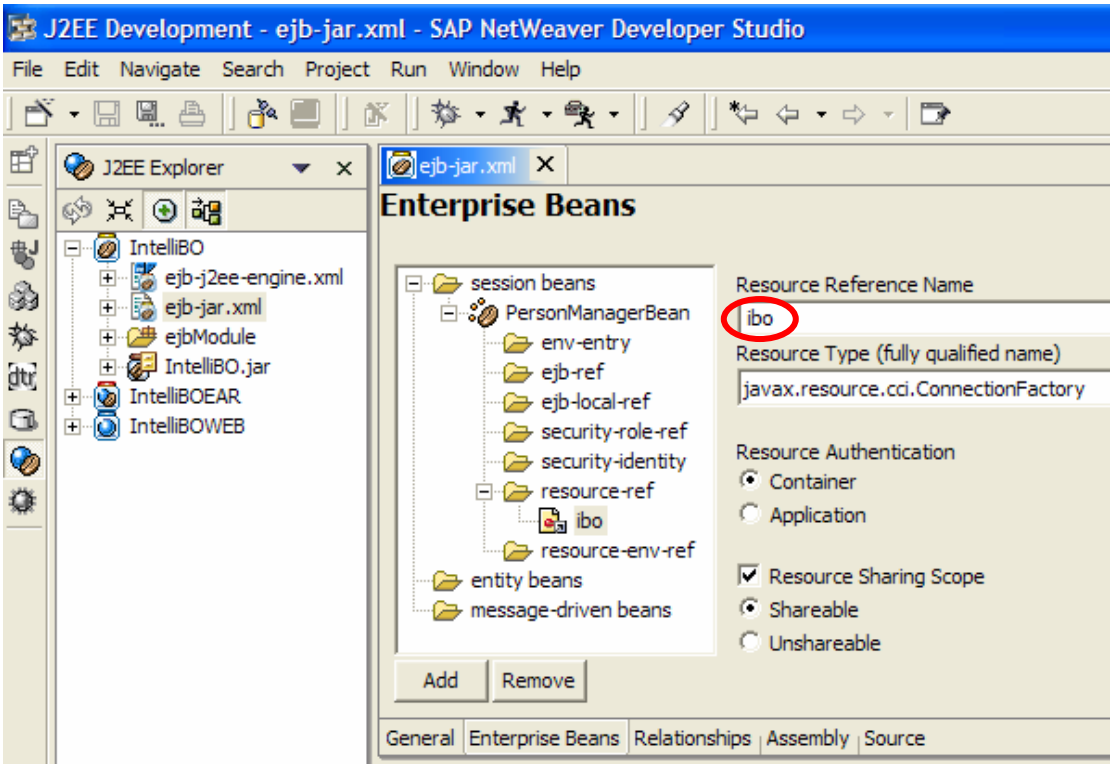
- Switch to the *J2EE Explorer* view.
- Open the `application-j2ee-engine.xml` descriptor that is found in the *intelliBOEAR* project.
- Select the *References* folder and click *Add*.
- Click *Create new*.
- Declare the reference as follows:

[5] The *build* target comprises all the other implied sub targets necessary.



The PersistenceManagerFactory's JNDI Lookup

To acquire a PersistenceManagerFactory, a JNDI lookup is being done. To allow for this, a resource-reference has to be declared in the ejb-jar.xml:



This resource reference name is used in the application's lookup, too. Here's the excerpt of `PersonManagerBean.java` (the session bean) that declares the JNDI lookup string for the `PersistenceManagerFactory`:

Part of PersonManagerBean.java

```
  private void getPmf()
  {
    Context initialContext;
    try
    {
      initialContext = new InitialContext();
      this.pmf = (PersistenceManagerFactory)
initialContext.lookup("java:comp/env/ibo");
```

The resource reference name is determined by the resource apdapter deployed. `connector-j2ee-engine.xml`, one of the resource adaptors deployment descriptors declares the JNDI name. `connector-j2ee-engine.xml` can be found at:

`<intellibo-howto.zip extraction location>/``rar/META-INF`.

Build, Deploy and Run

Now that our application is "JDO"-ed by means of intelliBO, let's package the application into an ear.

For this step, there is no derivation from the usual way of doing that in the NWDS:

1. In the *J2EE Explorer* view, select the Enterprise Application project *intelliBOEAR*.
2. From the context menu, choose *Build EAR File*.

The resulting EAR file is *intelliBOEAR.ear*

Assuming the J2EE Engine has been started, deploy the EAR file:

1. In the *J2EE Explorer or J2EE DC Explorer*, select the *intelliBOEAR.ear* EAR file.
2. From the context menu, choose *Deploy to J2EE Engine*.

Finally, this is the URL to test our sample application using the intelliBO JDO implementation once the application is deployed: URL: http://localhost:50000/intelliBOWEB/index.jsp[6]

[6] Keep in mind to adjust the URL to your system, the port might differ, for example.

Copyright