EasyBeans User's guide

Florent BENOIT, ObjectWeb consortium

Table of Contents

Chapter 1. Introduction to EJB3

1.1. Overview

EJB3 is included in the next J2EE specification, JAVA EE 5. (http://java.sun.com/javaee/5/ [http://java.sun.com/javaee/5/])

The EJB3 specification is defined in JSR 220, which can be found at the following location: http://www.jcp.org/en/jsr/detail?id=220

The publication is published as three separate files:

1. The core
2. The persistence provider
3. The simplified specification, which contains new features

The EJB3 persistence provider is plugged into the EJB3 container. Available persistence providers are: Hibernate [http://www.hibernate.org], Speedo [http://speedo.objectweb.org] (An ObjectWeb product), etc.

1.2. The Advantage of EJB3

EJB 2.x was too complex. Developers were using additional tools to make it easier.

- XDoclet (Attribute oriented programming): http://xdoclet.sourceforge.net
- Hibernate for persistence: http://www.hibernate.org

The main focus for this specification is on Ease Of Development (EoD). One major way this has been simplified is by using metadata attribute annotations suppported by JDK 5.0.

Simplifying EJB development should produce a wider range of Java EE developers.

1.3. EJB2 vs EJB3: EoD

The deployment descriptors are no longer required; everything can be accomplished using metadata annotations.

The CMP (Container Managed Persistence) has been simplified; it is now more like Hibernate or JDO.

Programmatic defaults have been incorporated. For example, the transaction model is set to REQUIRED by default. The value needs to be set only if a specific value other than the default value is desired.

The use of checked exceptions is reduced; the RemoteException is no longer mandatory on each remote business method.

Inheritance is now allowed; therefore, beans can extend some of the base code.

The native SQL queries are supported as an EJB-QL (Query Language) enhancement.

1.4. New Features

1.4.1. Metadata Annotations

Metadata annotations is new. FOr example, to define a stateless session bean, the @Stateless annotation is declared on the bean class.

1.4.2. Business Interceptors

The new business interceptors allow the developer to intercept each business method of the bean. The parameters and the returned values can be changed. For example, an interceptor can be used to determine the time that a method takes to execute.

1.4.3. Lifecycle Interceptors

In addition to business interceptors, the EJB2 callbacks (such as the `ejbActivate()` method) are now defined using annotation. For the `ejbActivate()` method, this is done with the help of @PostActivate annotation. This annotation is set on a method that will be called by the container.

1.4.4. Dependency Injection

Dependency injection makes it possible to request that the container inject resources, instead of trying to get them. For example, with the EJB2 specification, in order to get an EJB, the following code was used:

```
try {
Object o = new InitialContext().lookup("java:comp/env/ejb/MyEJB");
myBean = PortableRemoteObject.narrow(o, MyInterface.clas);
} catch (NamingException e) {
....
}
```

With EJB3 this is done using only the following code:

```
@EJB private MyInterface myBean;
```

If the @EJB annotation is found in the class, the container will look up and inject an instance of the bean in the `myBean` variable.

1.4.5. Persistence

New features are linked to the persistence layer. For example, EJB3 entities are POJO (Plain Old Java Object). This means that they can be created by using the `new()` constructor: `new MyEntity();`

`Also` entities are managed by an EntityManager: entitymanager.persist(entity);

In addition, entities have callbacks available.

Chapter 2. Getting EasyBeans From the SVN Repository

Anyone can check out source code from the SVN server using the following command (for GUI SVN client use, configuration values are the same as for command line use):

svn checkout svn://svn.forge.objectweb.org/svnroot/easybeans easybeans

Chapter 3. Using the Examples

3.1. Compiling the Examples

3.1.1. Requirements

Before running the examples, be sure to follow the requirements for compiling and running these EasyBeans examples.

3.1.2. Compile

The ant tool is used to build the examples. To compile the examples, use the `build.xml` file that is located in the `examples` directory.

The command **`ant install_all_examples`** must be launched in the `examples` directory:

```
$ ant install_all_examples
Buildfile: build.xml

install_all_examples:

init:

compile:
[javac] Compiling 7 source files to /home/benoitf/workspace/easybeans/output/example-classes

install.persistence:

install.war:
[mkdir] Created dir: /home/benoitf/workspace/easybeans/webapps/stateless.war/WEB-INF/classes
[copy] Copying 6 files to
/home/benoitf/workspace/easybeans/webapps/stateless.war/WEB-INF/classes
[copy] Copying 1 file to /home/benoitf/workspace/easybeans/webapps/stateless.war/WEB-INF

install:
[copy] Copying 5 files to /home/benoitf/workspace/easybeans/ejb3s/stateless.jar
[jar] Building jar: /home/benoitf/workspace/easybeans/clients/client-stateless.jar

init:

compile:
[javac] Compiling 3 source files to /home/benoitf/workspace/easybeans/output/example-classes

install.persistence:

install.war:

install:
[copy] Copying 2 files to /home/benoitf/workspace/easybeans/ejb3s/stateful.jar
[jar] Building jar: /home/benoitf/workspace/easybeans/clients/client-stateful.jar

init:

compile:
[javac] Compiling 4 source files to /home/benoitf/workspace/easybeans/output/example-classes

install.persistence:
[mkdir] Created dir: /home/benoitf/workspace/easybeans/ejb3s/entitybean.jar/META-INF
[copy] Copying 1 file to /home/benoitf/workspace/easybeans/ejb3s/entitybean.jar/META-INF

install.war:

install:
[copy] Copying 4 files to /home/benoitf/workspace/easybeans/ejb3s/entitybean.jar
[jar] Building jar: /home/benoitf/workspace/easybeans/clients/client-entitybean.jar

init:

compile:
[javac] Compiling 3 source files to /home/benoitf/workspace/easybeans/output/example-classes

install.persistence:
```

```
install.war:

install:
[copy] Copying 2 files to /home/benoitf/workspace/easybeans/ejb3s/mdb.jar
[jar] Building jar: /home/benoitf/workspace/easybeans/clients/client-mdb.jar

init:

compile:
[javac] Compiling 5 source files to /home/benoitf/workspace/easybeans/output/example-classes

install.persistence:

install.war:

install:
[copy] Copying 4 files to /home/benoitf/workspace/easybeans/ejb3s/security.jar
[jar] Building jar: /home/benoitf/workspace/easybeans/clients/client-security.jar

BUILD SUCCESSFUL
Total time: 8 seconds
```

The examples are copied under the `ejb3s/` folder of the project and are available for the deployment.

Note

If the EasyBeans server is running, it will detect these new applications and deploy them automatically.

3.2. Running Examples

Each example has its own `build.xml` file; this allows each example to be run independently.

3.2.1. Stateless Session Bean

The `build.xml` file for this example is located in the `examples/statelessbean` folder.

3.2.1.1. Description

This example is a stateless session bean. It contains a `helloWorld()` method that displays text on the server side. Additionally, it demonstrates the use of EJB3 annotation, such as @Stateless.

The `trace()` method is annotated with @AroundInvoke EJB3 annotation. This method will be called at each call on a business method. The business methods are defined in the interface implemented by the SessionBean class.

The signature of the method annotated by @AroundInvoke when it is defined in the bean class, must follow this signature:

```
(private|protected|public) Object methodName(InvocationContext invocationContext)
   throws Exception;
```

Note

As a new feature of EJB3, the bean's interface does not need to extend the Remote interface.

3.2.1.2. Running the Server

If the server is not available, it must be run by following the steps described in Chapter 3, "Running the EasyBeans Server."

3.2.1.3. Deploying the Bean

The stateless session bean must be deployed. If the bean has been installed in the `ejb3s` folder, this is done automatically.

On the server side, the following output should display:

```
[java] INFO: Creating container for archive
/home/benoitf/workspace/easybeans/ejb3s/stateless.jar.
[java] INFO: Analyze elapsed during : 95 ms
[java] INFO: Binding bean XXX with interface XXX into registry with jndi name XXX
[java] INFO: Enhancement elapsed during : 105 ms
[java] INFO: Container started in : 274 ms
```

Once this information is displayed on the screen, the container is ready to receive client calls.

3.2.1.4. Running the Client

Once the container has been started, the client can be launched.

Run the client with the following ant command: **ant run.client**

If the client runs successfully, the following output is displayed:

```
[java] Calling helloWorld method...
[java] Add 1 + 2...
[java] Sum = '3'.
```

Note

In the client's code, the use of the PortableRemoteObject.narrow() call is no longer required.

3.2.2. Stateful Session Bean

The `build.xml` file for this example is located in the `examples/statefulbean` folder.

3.2.2.1. Description

This is an example of a stateful session bean using the `SessionSynchronization` interface.

It uses the @Stateful annotation and uses the default transaction model, which is REQUIRED.

3.2.2.2. Running the Server

If the server is not available, it must be run by following the steps described in Chapter 3, "Running the EasyBeans Server."

3.2.2.3. Deploying the Bean

The stateful session bean must be deployed. It is done automatically if the bean has been installed in the `ejb3s` folder.

On the server side, the following output should be seen:

```
[java] INFO: Creating container for archive
/home/benoitf/workspace/easybeans/ejb3s/stateful.jar.
[java] INFO: Analyze elapsed during : 89 ms
[java] INFO: Enhancement elapsed during : 76 ms
[java] INFO: Binding bean XXX with interface XXX into registry with jndi name XXX
[java] INFO: Container started in : 251 ms
```

Once this information is displayed on the screen, the container is ready to receive client calls.

3.2.2.4. Running the Client

Once the container has been started, the client can be launched.

Run the client with the following ant command: **ant run.client**

If the client runs successfully, the following output is displayed:

```
[java] Start a first transaction
[java] First request on the new bean
[java] Second request on the bean
[java] Commit the transaction
[java] Start a second transaction
[java] Buy 50 amount.
[java] Rollback the transaction
[java] after rollback, value = 30
[java] Request outside any transaction
[java] Check that value = 30
[java] ClientStateful OK. Exiting.
```

3.2.3. Entity Bean

The `build.xml` file for this example is located in the `examples/entitybean` folder.

3.2.3.1. Description

This is an example of an entity bean. It describes how to use the new Java Persistence Model of an EJB3 persistence provider. To access EJB3 entities that are POJO, a stateless session bean is used. It is a facade bean.

The Entity class is a POJO class annotated with @Entity. The entities class is managed by the persistence provider.

Currently, the persistence provider is supplied by the Hibernate product, but the ObjectWeb Speedo product should be available soon. Users will have the choice between providers.

This example uses the @Stateful annotation and uses the default transaction model, which is REQUIRED.

The example shows an entity bean using EJB3 Hibernate-prototype persistence provider.

3.2.3.2. Running the Server

If the server is not available, it must be run following the steps described in Chapter 3, "Running the EasyBeans Server."

3.2.3.3. Deploying the Bean

The entity bean must be deployed. It is done automatically if the bean has been installed in the `ejb3s` folder.

On the server side, the following output should be seen:

```
[java] INFO: Creating container for archive
/home/benoitf/workspace/easybeans/ejb3s/entitybean.jar.
[java] INFO: Analyze elapsed during : 95 ms
[java] INFO: Enhancement elapsed during : 102 ms
[java] INFO: No persistence provider was set, set to value
org.hibernate.ejb.HibernatePersistence.
[java] INFO: Hibernate 3.1.1
[java] INFO: Using provided datasource
[java] INFO: RDBMS: HSQL Database Engine, version: 1.8.0
[...]
[java] INFO: Binding bean XXX with interface XXX into registry with jndi name XXX
[java] INFO: Container started in : 2010 ms
```

Once this information is displayed on the screen, the container is ready to receive client calls.

3.2.3.4. Running the Client

Once the container has been started, the client can be launched.

The client is run with the following ant command: **ant run.client**

If the client runs successfully, the following output is displayed:

```
[java] Employee with id 1 = Florent
[java] Employee with id 2 = Whale
```

3.2.3.5. Properties for the persistence

These properties are defined in the `META-INF/persistence.xml` file.

3.2.3.5.1. JDBC Dialect

By default, the dialect used to communicate with the database is set to HSQL, as it is embedded in EasyBeans.

This dialect configuration is done with the following properties:

<property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect" /> <property name="toplink.target-database" value="HSQL"/>

3.2.3.5.2. Database (tables)

By default, the tables are created and the database is empty after loading the entity beans.

This configuration is done with the following properties:

<property name="hibernate.hbm2ddl.auto" value="create-drop"/> <property name="toplink.ddl-generation" value="drop-and-create-tables"/>

In order to keep data in the database, this property should be changed.

3.2.4. Security example

The `build.xml` file for this example is located in the `examples/security` folder.

3.2.4.1. Description

This example illustrates the use of different Java EE 5.0 annotations which are linked to the security part.

The annotations used by the example are:

- @DeclareRoles, which is used to declare the roles used by an EJB component
- @RolesAllowed, which lists the authorized roles in order to call a method
- @DenyAll, which denies the call to the method (for every role)
- @RunAs, which sets a new identity when calling other EJBs

3.2.4.2. Running the Server

If the server is not available, it must be run following the steps described in Chapter 3, "Running the EasyBeans Server."

3.2.4.3. Deploying the Bean

The security bean example must be deployed. It is done automatically if the bean has been installed in the `ejb3s` folder.

On the server side, the following output should display:

```
[java] 10/16/06 5:24:50 PM (I) ContainersMonitor.scanNewContainers : Creating container for
archive /home/benoitf/workspace/easybeans/ejb3s/security.jar.
[java] 10/16/06 5:24:50 PM (I) JContainer3.start : Analyze elapsed during : 124 ms
[java] 10/16/06 5:24:50 PM (I) JContainer3.start : Enhancement elapsed during : 99 ms
[java] 10/16/06 5:24:50 PM (I) JContainer3.start : Container started in : 363 ms
[java] 10/16/06 5:24:50 PM (I) ContainersMonitor.scanNewContainers : Creating container for
archive /home/benoitf/workspace/easybeans/ejb3s/mdb.jar.
[java] 10/16/06 5:24:50 PM (I) JContainer3.start : Analyze elapsed during : 4 ms
[java] 10/16/06 5:24:51 PM (I) JContainer3.start : Enhancement elapsed during : 287 ms
[java] 10/16/06 5:24:51 PM (I) JContainer3.start : Container started in : 544 ms
```

Once this information is displayed on the screen, the container is ready to receive client calls.

3.2.4.4. Running the Client

Once the container has been started, the client can be launched.

The client is run with the following ant command: **ant run.client**

If the client runs successfully, the following output is displayed on the client side:

```
run.client:
[java] Oct 16, 2006 5:27:03 PM
org.objectweb.carol.util.configuration.ConfigurationRepository init
[java] INFO: No protocols were defined for property 'carol.protocols', trying with default
protocol = 'jrmp'.
[java] Calling methods that everybody can call...
[java] Call a bean with run-as in order to have 'admin' role...
[java] Access denied as expected (method is denied)
```

The following output is displayed on the server side:

```
[java] someRolesAllowed() called
[java] -> Caller is 'Principal[EasyBeans/Anonymous]'.
[java] for run-as bean, caller is Caller is 'Principal[EasyBeans/Anonymous]
[java] onlyAdminAllowed() called
[java] -> Caller is 'Principal[admin]'.
[java] someRolesAllowed() called
[java] -> Caller is 'Principal[admin]'.
```

Chapter 4. Writing a HelloWorld Bean

4.1. Requirements

This example illustrates the basics of an EJB3 application, showing all the steps used to build and run the EJB.

The only addtional information required is to know how to run the server.

4.2. Writing Code for the Bean

The HelloWorld bean is divided into two parts: the business interface, and the class implementing this interface.

4.2.1. Writing the Interface

The interface declares only one method: `helloWorld()`

```
package org.objectweb.easybeans.tutorial.helloworld;

/**
* Interface of the HelloWorld example.
* @author Florent Benoit
*/
public interface HelloWorldInterface {

/**
* Hello world.
*/
void helloWorld();

}
```

Note

Even if this interface is used as a remote interface, it does not need to extend `java.rmi.Remote` interface.

4.2.2. Writing the Business Code

The following code implements the existing interface:

```
package org.objectweb.easybeans.tutorial.helloworld;

/**
* Business code for the HelloWorld interface.
* @author Florent Benoit
*/
public class HelloWorldBean implements HelloWorldInterface {

/**
* Hello world implementation.
*/
public void helloWorld() {
System.out.println("Hello world !");
}

}
```

Note

At this moment, the bean is not an EJB; this is only a class implementing an interface.

4.2.3. Defining the EJB Code as a Stateless Session Bean

Now that the EJB code has been written, it is time to define the EJB application.

This bean will be a stateless session bean, thus the class will be annotated with @Stateless annotation.

In addition, the interface must be a remote interface to be available for remote clients. This is done by using the @Remote annotation.

```
package org.objectweb.easybeans.tutorial.helloworld;


import javax.ejb.Remote;
import javax.ejb.Stateless;

/**
* Business code for the HelloWorld interface.
* @author Florent Benoit
*/
@Stateless
@Remote(HelloWorldInterface.class)
public class HelloWorldBean implements HelloWorldInterface {

/**
* Hello world implementation.
*/
public void helloWorld() {
System.out.println("Hello world !");
}

}
```

Note

If a class implements a single interface, this interface is defined as a local interface by default.

4.2.4. Packaging the Bean

The two classes (`HelloWorldInterface` and `HelloWorldBean`) must be compiled.

Then, a folder named `ejb3s/helloworld.jar/` must be created and classes placed in this folder. They will be deployed and loaded automatically.

4.3. Writing the Client Code

The client can access the business interface directly and can call the methods of the bean directly.

```
package org.objectweb.easybeans.tutorial.helloworld;

import javax.naming.Context;
import javax.naming.InitialContext;

/**
* Client of the helloworld bean.
* @author Florent Benoit
*/
public final class Client {

/**
* JNDI name of the bean.
*/
private static final String JNDI_NAME =
"org.objectweb.easybeans.tutorial.helloworld.HelloWorldBean"
+ "_" + HelloWorldInterface.class.getName() + "@Remote"

/**
* Utility class. No public constructor
```

```
*/
private Client() {
}

/**
* Main method.
* @param args the arguments (not required)
* @throws Exception if exception is found.
*/
public static void main(final String[] args) throws Exception {
Context initialContext = new InitialContext();

HelloWorldInterface businessItf =
(HelloWorldInterface) initialContext.lookup(JNDI_NAME);

System.out.println("Calling helloWorld method...");
businessItf.helloWorld();
}

}
```

Note

The client does not call the PortableRemoteObject.narrow() method. Also, no create() method is required.

4.4. Writing a First Business Method Interceptor

An interceptor can be defined in the bean class or in another class. In this example, it will be defined in the bean's class. A business interceptor is defined by using the @AroundInvoke annotation.

The following interceptor will print the name of the method that is invoked. Of course, this could be extended to perform more functions.

```
/**
* Dummy interceptor.
* @param invocationContext contains attributes of invocation
* @return method's invocation result
* @throws Exception if invocation fails
*/
@AroundInvoke
public Object intercept(final InvocationContext invocationContext) throws Exception {
System.out.println("Intercepting method '" + invocationContext.getMethod().getName()
+ "'.");
try {
return invocationContext.proceed();
} finally {
System.out.println("End of intercepting.");
}
}
```

Caution

Be sure to call the `proceed()` method on the invocationContext object; otherwise, the invocation is broken.

4.5. Writing a First Lifecycle Interceptor

The bean can be notified of certain lifecycle events: for example, when a bean is created or destroyed.

In the following example, a method of the bean will receive an event when an instance of the bean is built. This is done by using the @PostConstruct annotation.

Llifecycle interceptors of a bean may be defined in another class.

```
/**
* Notified of postconstruct event.
```

```
*/
@PostConstruct
public void notified() {
System.out.println("New instance of this bean");
}
```

Chapter 5. Configuration File

5.1. Introduction

EasyBeans is configured with the help of an easy-to-understand XML configuration file.

The following is an example of an EasyBeans XML configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<easybeans xmlns="http://org.objectweb.easybeans.server">
<!-- Define components that will be started at runtime -->
<components>
<!-- RMI/JRMP will be used as protocol layer -->
<rmi>
<protocol name="jrmp" port="1099" hostname="localhost" />
</rmi>

<!-- Start a transaction service -->
<tm />

<!-- Start a JMS provider -->
<jms port="16030" hostname="localhost" />

<!-- Creates an embedded HSQLDB database -->
<hsqldb port="9001" dbName="jdbc_1">
<user name="easybeans" password="easybeans" />
</hsqldb>

<!-- Creates a JDBC pool with jdbc_1 JNDI name -->
<jdbcpool jndiName="jdbc_1" username="easybeans"
password="easybeans"
url="jdbc:hsqldb:hsql://localhost:9001/jdbc_1"
driver="org.hsqldb.jdbcDriver" />

<!-- Start smartclient server with a link to the rmi component-->
<smart-server port="2503" rmi="#rmi" />
</components>
</easybeans>
```

By default, an `easybeans-default.xml` file is used. To change the default configuration, the user must provide a file named `easybeans.xml`, which is located at classloader/CLASSPATH.

5.2. Configuration

Each element defined inside the <components> element is a component.

Note that some elements are required only for the standalone mode. JMS, RMI, HSQL, and JDBC pools are configured through JOnAS server when EasyBeans runs inside JOnAS.

5.2.1. RMI Component

The RMI configuration is done using the <rmi> element.

To run EasyBeans with multiple protocols, the <protocol> element can be added more than once.

The hostname and port attributes are configurable.

Protocols could be "jrmp, jeremie, iiop, cmi". The default is jrmp.

Note

Some protocols may require libraries that are not packaged by default in EasyBeans.

5.2.2. Transaction Component

The Transaction Component is defined by the <tm> element.

A timeout attribute, which is the transaction timeout (in seconds), can be defined on this element. The default is 60 seconds.

The implementation provided by the JOTM [http://jotm.objectweb.org] objectweb project is the default implementation.

5.2.3. JMS Component

The JMS component is used for JMS Message Driven Beans. Attributes are the port number and the hostname.

The default implementation is the implementation provided by the JORAM [http://joram.objectweb.org] objectweb project.

5.2.4. HSQL Database

EasyBeans can run an embedded database. Available attributes are the port number and the database name. The <hsqldb> may be duplicated in order to run several HSQLDB instances.

Users are defined through the <user> element.

5.2.5. JDBC Pool

This component allows the JDBC datasource to be bound into JNDI. The jndi name used is provided by the jndiName attribute.

Required attributes are username, password, url and driver.

Optional attributes are poolMin, poolMax and pstmtMax. This component provides the option to set the minimum size of the pool, the maximum size, and the size of the prepared statement cache.

5.2.6. SmartServer Component

This component is used by the Smart JNDI factory on the client side. This allows the client to download missing classes. The client can be run without a big jar file that provides all the classes. Classes are loaded on demand.

Note

Refer to the Chapter titled, Smart JNDI Factory, for more information about this feature.

5.3. Advanced Configuration

This configuration file can be extended to create and set properties on other classes.

5.3.1. Mapping File

A mapping file named `easybeans-mapping.xml` provides the information that rmi is the Carol-Component, tm is the JOTM component, and jms is the Joram component. This file is located in the org.objectweb.easybeans.server package.

The following is an extract of the `easybeans-mapping.xml` file.

```
<?xml version="1.0" encoding="UTF-8"?>
<easybeans-mapping xmlns="http://easybeans.objectweb.org/xml/ns/mapping">

<class name="org.objectweb.easybeans.component.Components"
alias="components" />
```

```
<package name="org.objectweb.easybeans.component.carol">
<class name="CarolComponent" alias="rmi">
<attribute name="portNumber" alias="port" />
</class>
<class name="Protocol" alias="protocol" />
</package>

...

</easybeans-mapping>
```

Note

This mapping file is referenced by the easybeans configuration file using the XML namespace : xmlns="http://org.objectweb.easybeans.server".

Each element configured within this namespace will use the mapping done in the org.objectweb.easybeans.server package.

Users can define their own mapping by providing a file in a package. The name of the the file must be xxx-mapping.xml, where xxx is the name of the element that will use this namespace.

Example: For the element <easybeans xmlns="http://org.objectweb.easybeans.server">, the resource searched in the classloader is org/objectweb/easybeans/server/easybeans-mapping.xml.

5.3.2. Other Configuration Files

EasyBeans can be configured through other configuration files as it uses a POJO configuration. If done this way, it can be configured using the Spring Framework component or other frameworks/tools.

Chapter 6. Smart JNDI Factory

6.1. Introduction

The smart factory provided by EasyBeans is a factory that allows downloading of some classes from the server.

It is useful when developing heavy clients.

In order to run the clients, the developer must provide all the classes used to compile the client code and include a small jar file (less than 50kB) to add to the CLASSPATH.

Required libraries for running a client are:

- The client's code (used at compile time)
- The Interface of the Beans that are accessed (used at compile time)
- The Java EE API used by the client (used at compile time)
- The smart factory provided by the `ow_easybeans_component_smartclient.jar` jar file

6.2. Running the Client

The smart factory is configured through two properties.

6.2.1. Initial Context Factory

The first property is the InitialContextFactory name. The smart factory is named `org.objectweb.easybeans.component.smartclient.spi.SmartContextFactory.`

This property can be set as a System property in one of the following ways:

- by using -Djava.naming.factory.initial=org.objectweb.easybeans.component.smartclient.spi.SmartContextFactory
- by using System.setProperty(Context.INITIAL_CONTEXT_FACTORY, `org.objectweb.easybeans.component.smartclient.spi.SmartContextFactory`)

It can also be used as a parameter when creating an InitialContext:

```
Hashtable<String, Object> env = new Hashtable<String, Object>();
env.put(Context.INITIAL_CONTEXT_FACTORY,
org.objectweb.easybeans.component.smartclient.spi.SmartContextFactory);
return new InitialContext(env);
```

6.2.2. Provider URL

This property is used to provide the remote address and the remote port.

By default, this property is set (if not defined) to smart://localhost:2503

The port number must match the port defined in the EasyBeans configuration file.

This property can be set using:

```
Hashtable<String, Object> env = new Hashtable<String, Object>();
```

```
env.put(Context.INITIAL_CONTEXT_FACTORY,
org.objectweb.easybeans.component.smartclient.spi.SmartContextFactory);
env.put(Context.PROVIDER_URL, "smart://localhost:2503");
return new InitialContext(env);
```

6.3. Example

The following is the output on the client side when this factory is enabled:

```
[java] Oct 17, 2006 5:38:13 PM
org.objectweb.easybeans.component.smartclient.spi.SmartContextFactory getInitialContext
[java] INFO: Initializing Smart Factory with remote URL 'smart://localhost:2503'.
[java] Oct 17, 2006 5:38:13 PM
org.objectweb.easybeans.component.smartclient.spi.SmartContextFactory getInitialContext
[java] INFO: Got remote PROVIDER_URL 'rmi://localhost:1099'.

...
[java] Downloaded 'xxx' classes, 'xxx' resources for a total of 'xxx' bytes and it took
'xxx' ms.
```

The following is the output on the server side:

```
[java] 10/17/06 5:38:13 PM (I) SmartClientEndPointComponent.handleReadProviderURLRequest :
Provider URL asked by client : 'rmi://localhost:1099'.
```