EasyBeans Developer's guide

Florent BENOIT, ObjectWeb consortium

Table of Contents

Chapter 1. Building EasyBeans From Source.

1.1. Requirements

1.1.1. JDK

A JDK 5.0 is required to build EasyBeans. Make sure that the JDK used to build EasyBeans is compliant with the new 5.0 features.

1.1.2. ANT

The ant tool is used with `build.xml` files to build EasyBeans. This tool is available at http://ant.apache.org.

1.1.3. TestNG

The test suite of EasyBeans uses the TestNG tool. This tool is available at http://www.testng.org.

1.1.4. Clover

The test suite of EasyBeans uses Clover, which is a code-coverage, analysis tool. Cenqua has granted licenses to open source projects. Refer to http://www.cenqua.com/ for more about Clover.

1.2. Optional Requirements

1.2.1. Eclipse

The EasyBeans project provides .project and .classpath for Eclipse 3.1 or greater. A project is ready to use once the source has been imported using the Eclipse tool. Eclipse tool is available at http://www.eclipse.org.

1.2.2. Eclipse Plugins

1.2.2.1. Checkstyle Plugin

The eclipse-checkstyle plugin is used to check the javadoc of Easybeans project. A warning will print if the EasyBeans coding convention is not used. This plugin is available at http://eclipse-cs.sourceforge.net.

1.2.2.2. AnyEdit Plugin

As part of the EasyBeans coding convention, the use of tabulation characters is disallowed. Files should contain only spaces. The AnyEdit plugin allows tabs to be converted to spaces when saving the file. Also, trailing spaces can be removed automatically.

This plugin is available at http://andrei.gmxhome.de/anyedit/.

1.2.2.3. Asm Plugin

EasyBeans uses bytecode enhancement. This is done using the ObjectWeb ASM project. [http://asm.objectweb.org] ASM provides a plugin that allows the ASM code of a given class to be obtained. The plugin is available at http://asm.objectweb.org/eclipse/index.html.

1.2.2.4. TestNG Plugin

The EasyBeans test suite uses TestNG. A plugin is available for Eclipse: http://testng.org/doc/eclipse.html.

1.3. Compiling EasyBeans

To compile EasyBeans, launch the command **ant compile** in the root directory of the project (named easybeans by default) being launched.

Note

The command **ant -p** can be used to list the targets that are available.

Once the command has been run successfully, an `output` folder is created with a subfolder `classes`. This `classes` folder contains the generated classes.

ant clean is used to clean the generated classes.

1.4. EasyBeans ant Targets

1.4.1. JavaDoc

Javadoc of EasyBeans can be generated by using **ant javadoc**

The resulting documentation will be available in the `output/dist/javadoc` folder.

1.4.2. Binary (.jar) Output

Binary jar files are built using **ant jar**.

The generated jar files are located in the `output/dist` folder.

1.4.3. Binary (.rar) Output

rar files are generated (one for JOnAS application server) using **ant rar**.

The rar file is located in the `output/dist` folder.

1.4.4. Binary Distribution (default target)

The javadoc and binary jar/rar files are built using **ant dist**.

The jar/rar files are located in the `output/dist` folder.

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. Running EasyBeans server.

3.1. Requirements

Review the requirements discussed in Chapter 1, "Building EasyBeans from Source."

3.2. Running

The `build.xml` file located in the project root will be used to launch the EasyBeans server. This file is contained in the source distribution.

Use the following command: **ant run.server**

The EasyBeans server will be launched and the following output will be printed:

```
Buildfile: /home/benoitf/workspace/easybeans/build.xml
init:
[mkdir] Created dir: /home/benoitf/workspace/easybeans/output
[mkdir] Created dir: /home/benoitf/workspace/easybeans/output/manifest
[mkdir] Created dir: /home/benoitf/workspace/easybeans/output/dist
[mkdir] Created dir: /home/benoitf/workspace/easybeans/output/classes
[mkdir] Created dir: /home/benoitf/workspace/easybeans/output/dist/javadoc
compile:
[javac] Compiling 246 source files to /home/benoitf/workspace/easybeans/output/classes
[copy] Copying 9 files to /home/benoitf/workspace/easybeans/output/classes
run.server:
[java] Mar 14, 2006 4:53:47 PM org.objectweb.easybeans.log.CommonsLoggerImpl warn
[java] WARNING: No directory was configured, take the default value of
/home/benoitf/workspace/easybeans/ejb3s.
[java] Mar 14, 2006 4:53:47 PM org.objectweb.easybeans.log.CommonsLoggerImpl warn
[java] WARNING: Directory /home/benoitf/workspace/easybeans/ejb3s created.
[java] Mar 14, 2006 4:53:47 PM org.objectweb.carol.util.configuration.TraceCarol infoCarol
[java] INFO: Name service for jrmp is started on port 1099
[java] Mar 14, 2006 4:53:47 PM org.objectweb.easybeans.log.CommonsLoggerImpl info
[java] INFO: List of all MBeans descriptors
[java] Mar 14, 2006 4:53:47 PM org.objectweb.easybeans.log.CommonsLoggerImpl info
[java] INFO: Found managedBean EJB3Deployer.
[java] Mar 14, 2006 4:53:47 PM org.objectweb.easybeans.log.CommonsLoggerImpl info
[java] INFO: End of list of all MBeans descriptors
[java] Mar 14, 2006 4:53:48 PM org.objectweb.jotm.Current <init>
[java] INFO: JOTM 2.0.11
[java] Mar 14, 2006 4:53:48 PM org.objectweb.easybeans.log.CommonsLoggerImpl info
[java] INFO: Startup was done in '684' ms.
[java] Mar 14, 2006 4:53:48 PM org.objectweb.easybeans.log.CommonsLoggerImpl info
[java] INFO: '0' containers have been created.
[java] Mar 14, 2006 4:53:48 PM org.objectweb.easybeans.log.CommonsLoggerImpl info
[java] INFO: Waiting requests...
```

EasyBeans is now launched and it is ready to handle EJBs.

Chapter 4. Using the Examples

4.1. Compiling the Examples

4.1.1. Requirements

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

4.1.2. 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:

```
Buildfile: /home/benoitf/workspace/easybeans/examples/build.xml
install_all_examples:
init:
[mkdir] Created dir: /home/benoitf/workspace/easybeans/output/dist/clients
[mkdir] Created dir: /home/benoitf/workspace/easybeans/output/dist/ejbjars
[mkdir] Created dir: /home/benoitf/workspace/easybeans/clients
compile:
[javac] Compiling 5 source files to /home/benoitf/workspace/easybeans/output/classes
install.persistence:
install:
[copy] Copying 4 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/classes
install.persistence:
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/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:
[copy] Copying 4 files to /home/benoitf/workspace/easybeans/ejb3s/entitybean.jar
[jar] Building jar: /home/benoitf/workspace/easybeans/clients/client-entitybean.jar
BUILD SUCCESSFUL
Total time: 4 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.

4.2. Running Examples

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

4.2.1. Stateless Session Bean

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

4.2.1.1. 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 the EJB3, the bean's interface does not need to extend the Remote interface.

4.2.1.2. 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."

4.2.1.3. 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 be seen:

```
[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.

4.2.1.4. 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.

4.2.2. Stateful Session Bean

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

4.2.2.1. 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.

4.2.2.2. 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."

4.2.2.3. 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.

4.2.2.4. 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.
```

4.2.3. Entity Bean

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

4.2.3.1. 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.

4.2.3.2. Running the Server

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

4.2.3.3. 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.

4.2.3.4. 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
```

Chapter 5. EasyBeans Code Convention

Contributions should follow the EasyBeans code convention. A good document to begin with is Java code convention [http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html]. Other conventions are also listed in this document.

In addition, EasyBeans uses tools to check the compliance: the checkstyle plugin [http://checkstyle.sourceforge.net/] and the eclipse checkstyle plugin [http://eclipse-cs.sourceforge.net/]. The configuration settings are available on EasyBeans SVN [http://forge.objectweb.org/plugins/scmsvn/index.php?group_id=236].

5.1. File Organization

5.1.1. Header

All files should have a header that contains the LGPL and the date.

If a file is modified, the modification year should be appended to the existing year, which is the year it was initially created. For example, if the create date is '1999' or '2004' it should be edited to '1999-2006' or '2004-2006', respectively.

Also, the tag $Id: code_convention.xml 314 2006-04-04 09:39:43Z pinheirg $ should be added. The following is a header example:

```
/**
* EasyBeans
* Copyright (C) 2006 Bull S.A.S.
* Contact: easybeans@objectweb.org
*
* This library is free software; you can redistribute it and/or
* modify it under the terms of the GNU Lesser General Public
* License as published by the Free Software Foundation; either
* version 2.1 of the License, or any later version.
*
* This library is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
* Lesser General Public License for more details.
*
* You should have received a copy of the GNU Lesser General Public
* License along with this library; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307
* USA
*
* --------------------------------------------------------------------------
* $Id: code_convention.xml 825 2006-07-06 22:45:26Z kburgess $
* --------------------------------------------------------------------------
*/
```

5.1.2. Imports

Imports should reference a valid class name, instead of using wildcard imports. Wildcard imports are not authorized.

For example, if the classes List and ArrayList are used, the imports should not be as follows:

```
import java.util.*;
```

The imports should have each class as follow:

```
import java.util.List;
import java.util.ArrayList;
```

The classes should not have an unused import.

Note

The Eclipse IDE provides facilities to do this job. There is the option Organize Imports (**Shift+Ctrl+O**) in the menu Source that correctly inserts the imports and removes the unused imports. However, this option does not work well with 'import static'.

5.1.3. Class and Interface Declarations

The class and interface names should begin with an uppercase letter. Also, each class and interface has an @author tag in the comment. For example:

```
/**
* This is an example that shows a class/interface declaration.
* @author Gisele Pinheiro Souza
* @author Eduardo Studzinski Estima de Castro
*/
public class ClassExample implements InterfaceExample{
}
```

5.2. Indentation / WhiteSpace

5.2.1. Indentation

The space character is used instead of the tab character. The number of spaces for an indent is *4 spaces*.

Wrapping a single source line into multiple lines should follow the Java code convention [http://java.sun.com/docs/codeconv/html/CodeConventions.doc3.html#248].

5.2.2. WhiteSpace

Any trailing spaces should be removed. Eclipse provides a plugin that removes the trailing spaces and converts the tab into spaces. The plugin is AnyEdit [http://andrei.gmxhome.de/anyedit/].

Use whitespaces in for() loop, while(), when concatenating strings. One space should be added before the operator and another after the operator. For example, the correct syntax is:

```
for (int i = 0; i < arTest.length; i++) {
String strResult = "The element " + i + " has the value " + arTest[i];
}
```

The following code does not adhere to the convention:

```
for (int i = 0; i< arTest.length; i++) {
String strResult = "The element "+ i+" has the value "+arTest[i];
}
```

5.3. JavaDoc Comments

All methods and attributes (including protected and private) must have a comment. The parameters, the exceptions thrown, and the method return should have a comment in the method comment. For example:

```
/**
* This is an example that is used in the EasyBeans Code Convention.
*/
```

```
private int intValue;

/**
* This is an example method to show a class comment.
* @param a an example of parameter.
* @param b other example of parameter.
* @return the method result.
* @throws Exception the exception thrown by the method.
*/
public int add(final int a, final int b) throws Exception {
return a + b;
}
```

5.4. Statements

5.4.1. If/else

Braces must be used in the if/else blocks, even if there is a single statement. To illustrate:

```
if (true) {
doThis();
}
```

The following is not allowed:

```
if (true)
doThis();
```

The position of the braces should be the same as in the first example. The following format is incorrect:

```
if (true)
{
test1();
test2();
}
```

5.4.2. Try/catch

All exceptions require a statement; no silent catching is allowed. For example:

```
try {
doThis();
} catch (Exception e) {
// should not occur
}
```

A logger can be used:

```
try {
doThis();
} catch (Exception e) {
logger.logDebug("Exception while doing .....", e);
}
```

5.4.3. Inline Conditionals

Inline conditionals are not allowed. The following code is incorrect:

```
b = isOk() ? true : false;
```

The correct way to write this is as follows:

```
if (isOk()) {
b = true;
} else {
b = false;
}
```

5.4.4. Naming Conventions

5.4.4.1. Static Final Attributes

Declarations are static final, not final static. This is a JLS recommendation.

5.4.4.2. Constants

Constants should be static and final, and should adhere to the following:

```
'^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$'
```

5.4.4.3. No Magic Numbers, Use Constants

Constants must be used in the code and magic number must be avoided. For example, the following is not allowed:

```
private int myAttribute = 5;
```

The correct format is:

```
/**
* Default value
*/
private static final int DEFAULT_VALUE = 5;

/**
* This attribute is initialized with the default value
*/
private int myAttribute = DEFAULT_VALUE;
```

5.4.4.4. Attribute Name

The attribute name should not have an underscore (_). The _ is valid for constants that are in uppercase.

Use pValue and mValue instead of p_Value and m_Value.

The pattern for attribute name is:

```
'^[a-z][a-zA-Z0-9]*$'
```

Chapter 6. Contributing to EasyBeans

6.1. Mailing Lists

Developers wanting to contribute information about EasyBeans can share their thoughts via the easybeans mailing list.

The steps necessary for subscribing to the list are described at the following url :http://www.objectweb.org/wws/info/easybeans

6.2. Ideas for Contributing

There are many ways to contribute to easybeans. New ideas are also welcome.

The following is a list of some of the ways to make contributions:

- Documentation: Improve or add to the existing documentation, create new chapters, translate, etc.
- Code: Some glue could be added so that EasyBeans could be integrated in other servers.
- Tests: Add new tests to the current test suite.