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. So check that the JDK used to build EasyBeans is compliant with new 5.0 features

1.1.2. ANT

To build EasyBeans, the ant tool is used with `build.xml` files. This tool is available at http://ant.apache.org

1.1.3. TestNG

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

1.1.4. Clover

The test suite of EasyBeans uses Clover that is a code coverage analysis tool. Cenqua has granted licenses to open source projects. More about Clover at http://www.cenqua.com/

1.2. Optional requirements

1.2.1. Eclipse

EasyBeans project provides .project and .classpath for Eclipse 3.1 or greater. You can import the sources in Eclipse tool and the project will be ready to go. 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. It prints warning 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 convetion, the use of tabulations characters is disallowed. Files should contain only spaces. The AnyEdit plugin allows to convert tabs into spaces when saving the file. Also, trailing spaces could 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 by using the ObjectWeb ASM project. [http://asm.objectweb.org]ASM provides a plugin allowing to get the ASM code of a given class. The plugin is available at http://asm.objectweb.org/eclipse/index.html

1.2.2.4. TestNG plugin

The test suite of EasyBeans is using TestNG. There is a plugin available for Eclipse : http://testng.org/doc/eclipse.html

1.3. Compiling EasyBeans

In the root of the project (named easybeans by default), the command **ant compile** needs to be launched.

Note

The command **ant -p** could 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 last 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. Binaries (.jar) output

By using **ant jar**, binaries jar files are built.

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

1.4.3. Binaries (.rar) output

By using **ant rar**, rar files will be generated (one for JOnAS application server).

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

1.4.4. Binary distribution (default target)

By using **ant dist**, the javadoc will be built and binaries jar/rar files too.

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

Chapter 2. Getting EasyBeans from SVN repository

Anyone can checkout source code from the SVN server. To do this, the following command should be used (For GUI SVN client use, configure it appropriately):

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

Chapter 3. Running EasyBeans server.

3.1. Requirements

First, all the requirements used in building chapter need to be checked.

3.2. Running

The `build.xml` file at which is at the root of the project will be used. This file is found in the source distribution.

The following command needs to be used : **ant run.server**

Then, 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...
```

Now, EasyBeans is launched and it is ready to handle EJBs.

Chapter 4. Using the examples

4.1. Compiling the examples

4.1.1. Requirements

Before trying to run the examples, the requirements in order to compile and run EasyBeans have to be followed.

4.1.2. Compile

The ant tool is used to build the examples. This time, the `build.xml` file that is used is located in the `examples` directory.

The command **`ant install_all_examples`** needs to 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 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 in order to be independent from each other.

4.2.1. Stateless session bean

The `build.xml` file to use is in `examples/statelessbean` folder.

4.2.1.1. Description

This example is a stateless session bean. It contains an `helloWorld()` method which displays a text on the server side. Also, it demonstrates the use of EJB3 annotation like @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, needs to follow this signature :

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

Note

As a new feature of the EJB3, the bean's interface doesn't need to extend anymore the Remote interface.

4.2.1.2. Run the server

If the server is not available, it needs to be run by following the steps described in this guide.

4.2.1.3. Deploying the bean

The stateless session bean needs to be deployed. It is done automatically if the bean has been installed in `ejb3s` folder.

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
```

If these informations are on the screen, it means that the container is ready to receive client calls.

4.2.1.4. Run the client

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

Note

In the client's code, the use of PortableRemoteObject.narrow() call is not required anymore.

4.2.2. Stateful session bean

The `build.xml` file to use is in `examples/statefulbean` folder.

4.2.2.1. Description

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

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

4.2.2.2. Run the server

If the server is not available, it needs to be run by following the steps described in this guide.

4.2.2.3. Deploying the bean

The stateful session bean needs to be deployed. It is done automatically if the bean has been installed in `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
```

If these informations are on the screen, it means that the container is ready to receive client calls.

4.2.2.4. Run the client

As 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] 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 to use is in `examples/entitybean` folder.

4.2.3.1. Description

This example is an entity bean.

It describes how to use the new Java Persistence Model of an EJB3 persistence provider. To access EJB3 entities which 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 are managed by the persistence provider.

Currently the persistence provider is provided by the Hibernate product, but ObjectWeb Speedo product should be used sooner. Users will have the choice between providers.

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

This example shows the use of an entity bean and using EJB3 persistence provider which is in this prototype Hibernate. In a next version, the ObjectWeb Speedo product will provide an EJB3 persistence provider, so users will have the choice between these providers.

4.2.3.2. Run the server

If the server is not available, it needs to be run by following the steps described in this guide.

4.2.3.3. Deploying the bean

The entity bean needs to be deployed. It is done automatically if the bean has been installed in `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
```

If these informations are on the screen, it means that the container is ready to receive client calls.

4.2.3.4. Run the client

As 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

The contributions should follow the EasyBeans code convention. An good document to get started is Java code convention [http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html]. Besides, there is others conventions that are listed in this document.

In addition, the 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. Files Organization

5.1.1. Header

All files should have the header which contains the LGPL and the date.

If a file is modified, the modification year should be appended in the existing year. For example, if there was '1999' or '2004' put '1999-2006' or '2004-2006', respectively.

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

```
/**
* 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 314 2006-04-04 09:39:43Z pinheirg $
* --------------------------------------------------------------------------
*/
```

5.1.2. Imports

The imports should reference a correct class name, instead of the wildcard imports. The wildcard imports is not authorized.

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

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

But the imports should have each class as follow:

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

The classes should not have a 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 inserts correctly 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 name should begin with an uppercase. Also, each class and interface has a @author tag in the comment. For example:

```
/**
* This is an example that shows how is 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 is used instead of the tab character. The number of spaces for an indent is *4 spaces*.

The wrapping lines follow the Java code convention in this part [http://java.sun.com/docs/codeconv/html/CodeConventions.doc3.html#248].

5.2.2. WhiteSpace

The trailing spaces should be removed. There is an eclipse plug in that removes the trailing spaces and converts the tab into spaces. The plug in 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 other after the operator. For example, the correct is:

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

The follow code does not follow 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. In a method comment the parameters, the exceptions thrown and the method return should have a 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 how is 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

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

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

This is not allowed:

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

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

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

5.4.2. Try/catch

All exception required a statement, no silent catching like :

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

You can use a logger :

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

5.4.3. Inline Conditionals

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

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

The correct way to write is:

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

5.4.4. Naming conventions

5.4.4.1. Static final attributes

The declarations is static final and is not final static, this is a JLS recommendation.

5.4.4.2. Constants

The constants should be static and final, and should follow the pattern:

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

5.4.4.3. No magic number, use constants

The constants must to be used avoiding magic numbers in the code. For example, it 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 a _. The _ is valid for constants that are in uppercase.

You can use pValue, mValue instead of p_Value, m_Value.

The pattern for attribute name is:

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

Chapter 6. Contributing to EasyBeans

6.1. Mailing-lists

First, developers wanting to contribute on EasyBeans could share their thoughts on the easybeans mailing list.

In order to subscribe to the list, steps are described at the following url :http://www.objectweb.org/wws/info/easybeans

6.2. Ideas for contributing

There are several ways to contribute to easybeans and then there a lot of ideas.

Here is a list of things than can be done :

- Documentation : improving the existing documentation, creating new chapters, translating, etc.
- Code : Some glue could be added so that EasyBeans could be integrated in other server.
- Tests : New tests can be added to the current test suite.