LAURENT ETIEMBLE
The JBoss Project

JBoss IDE 1.2.0 Tutorial
JBoss IDE 1.2.0 : Tutorial
Table of Content

PREFACE ................................................................................................................................................................................................................................... 4
FORWARD .................................................................................................................................................................................................................. 4
ABOUT THE AUTHORS ................................................................................................................................................................................... 4
ACKNOWLEDGMENTS .................................................................................................................................................................................... 4

1. INTRODUCTION ........................................................................................................................................................................................................ 5

2. THE TUTORIAL .............................................................................................................................................................................................................. 6
INTRODUCTION .................................................................................................................................................................................................................. 6
REQUIREMENTS ................................................................................................................................................................................................................. 6
THE PROJECT ............................................................................................................................................................................................................... 7
THE EJB ............................................................................................................................................................................................................................ 9
GENERATION OF THE EJB RELATED FILES ................................................................................................................................................ 13
THE SERVLET AND THE WEB-APP ................................................................................................................................................................. 18
GENERATION OF THE SERVLET RELATED FILES ...................................................................................................................................... 24
THE J2EE APPLICATION .................................................................................................................................................................................................. 28
THE PACKAGING ......................................................................................................................................................................................................... 29
JBOSS CONFIGURATION AND LAUNCH ...................................................................................................................................................... 41
DEPLOYMENT ................................................................................................................................................................................................................. 42
DEBUGGING .................................................................................................................................................................................................................. 44
CONCLUSION ............................................................................................................................................................................................................... 48
Preface

Forward
The JBoss IDE started with an XDoclet plug-in for eclipse in the middle of 2002. Then Hans Dockter met Marc as he participated at a JBoss training in Mallorca and they talked about the possibility of developing a JBoss IDE.

About the Authors

Hans Dockter, is the lead architect of the JBoss IDE. Hans works as an independent consultant and lives in Berlin, Germany.

Laurent Etiemble, is working as a developer on JBoss IDE. Laurent works as a consultant and lives in Paris, France.

JBoss Project, headed by Marc Fleury, is composed of over 100 developers worldwide who are working to deliver a full range of J2EE tools, making JBoss the premier Enterprise Java application server for the Java 2 Enterprise Edition platform.

JBoss is an Open Source, standards-compliant, J2EE application server implemented in 100% Pure Java. The JBoss/Server and complement of products are delivered under a public license. With a huge amount of downloads per month, JBoss is the most downloaded J2EE based server in the industry.

Acknowledgments

I’d like to thank SG Enterprise Systems and Chris Bedford, author of a nice tutorial for JBoss IDE. This one is inspired from it.
1. Introduction

JBoss-IDE offers you:

- A very comfortable and sophisticated support for XDoclet.
- The debugging and monitoring of JBoss servers and the controlling of there life cycles.
- An easy way to configure the packaging layout of archives (packed or exploded)
- A simple way to deploy the packaged and/or exploded archive to a JBoss server

Some part of the J2EE development process is not yet covered. That is J2EE specific project management (templates and wiazrds). This has still to be done. Anyway, the main focus for the next version is on these topics.

The following tutorial is an overview of the capabilities of JBoss-IDE. It aims to demonstrate how to use it to develop and debug J2EE applications.
2. The Tutorial

Introduction

The goals of this tutorial are to demonstrate how it is simple to develop a J2EE application with Jboss-IDE/Eclipse. The sample application that will be built is a J2EE application with one session EJB and one Servlet, which computes the Fibonacci suite.

The tutorial is split into several parts:

- The project: this part shows how the project is prepared (source and build path)
- The EJB: this part shows how to write an EJB class with its XDoclet javadoc tags.
- Generation of EJB files: this part shows how to configure the XDoclet generation configuration to generate all the EJB related files
- The Servlet and the Web-App: this part shows how to write a Servlet class with its XDoclet javadoc tags.
- Generation of Servlet files: this part shows how to configure the XDoclet generation configuration to generate all the Web related files
- The J2EE application: this part shows how to create the missing files.
- Packaging: this part shows how to package the J2EE application
- JBoss configuration: this part shows how to define debug configuration to launch JBoss inside Eclipse.
- Deployment: this part shows how to deploy by copy the J2EE application
- Debugging: this part shows how to set up breakpoints to debug the deployed application.

Requirements

For this tutorial you need:

- Java Development Kit 1.3.0 or higher (a JDK is needed to launch JBoss 3.x)
- Eclipse 2.1.0 (from http://www.eclipse.org)
JBoss Server 3.0.x or 3.2.x
You also need to have some Eclipse basis about development and debugging. Refer to Eclipse website for further informations.

The Project
This part shows how the project is prepared. We will create source folder, import libraries and make the build path.

Create a new Java Project. Enter “Tutorial” as the name of the new project.
Create a source folder named “src”. The default output folder will be “bin”.

In the package explorer, the new project should look like this.

Copy the two following files from your JBoss distribution to the root of the project:

- javax.servlet.jar (taken from the server/default/lib of the JBoss distribution)
- jboss-j2ee.jar (taken from the client folder of the JBoss distribution)
We want to put the two jars in the project build path. Edit the project properties by right clicking on the project and select “Properties”. In the property page, select “Java Build Path”. Click on “Add jars...” and select the two libraries (javax.servlet.jar and jboss-j2ee.jar).

Save the changes and you should see something as above. Now, you have a project with all you need to develop a nice J2EE application!

**The EJB**

The next step is to create an EJB. For simplicity, it will be a stateless session bean, but others types are also easy to write.
Create a new Java Class. The package will be “tutorial.ejb” and the class name “FiboBean”. Click on “Add…” to add the “SessionBean” interface. Be sure that only “Constructors from superclass” and “Inherited abstract methods” are checked.

Click on “Finish”. The class is then created and you should have a project like this.

Open the FiboBean.java file. We need a compliant EJB, so add an “ejbCreate” method without parameters, as our EJB is stateless.

```java
public void ejbCreate() throws CreateException {
}
```

To be interesting, our EJB need a business method. The method will compute a Fibonacci suite.

```java
public double[] compute(int number) {
    if (number < 0) {
        throw new EJBException("Argument should be positive");
    }
    double[] suite = new double[number + 1];
    suite[0] = 0;
    if (number == 0) {
```
The next step is to insert XDoclet javadoc related tags for the EJB. Thanks to the code completion, it is an easy operation. In the Java editor go in the Javadoc class paragraph. Type “@ejb.” And press CTRL+Space. You should see the magic of completion in action.

Complete the attributes of the tag with the following values (press CTRL+Space for each attribute if you want the completion):

```
* @ejb.bean name = "Fibo"
* display-name = "Fibo EJB"
* description = "EJB that computes Fibonacci suite"
* view-type = "remote"
* jndi-name = "ejb/tutorial/Fibo"
```

The "ejbCreate" and the "compute" method need to be tagged. Add the following to the "ejbCreate" method:

```
/**
   * Default create method
   * @throws CreateException
   */
```

```java
public class FiboBean implements SessionBean {

```
And the following to the "compute" method:

```java
/**
 * @param number
 * @return
 * @ejb.interface-method view-type = "remote"
 */
public double[] compute(int number) {
```

After that, the file should look like this. Now, we are ready to run XDoclet on the file generate all the EJB stuff.

```java
package tutorial.ejb;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

/**
 * @author John Doe
 *
 * @ejb.bean  name = "Fibo"
 * display-name = "Fibo EJB"
 * description = "EJB that computes Fibonacci suite"
 * view-type = "remote"
 * jndi-name = "ejb/tutorial/Fibo"
 */
public class FiboBean implements SessionBean {

    public FiboBean() {
        super();
    }

    /**
     * Default create method
     * @throws CreateException
     * @ejb.create-method
     */
    public void ejbCreate() throws CreateException {
    }

    public void ejbActivate() throws EJBException, RemoteException {
    }
```
public void ejbPassivate() throws EJBException, RemoteException {
}

public void ejbRemove() throws EJBException, RemoteException {
}

public void setSessionContext(SessionContext arg0)
        throws EJBException, RemoteException {
}

/**
   * @param number
   * @return
   * @ejb.interface-method view-type = "remote"
   */
public double[] compute(int number) {
    if (number < 0) {
        throw new EJBException("Argument should be positive");
    }

double[] suite = new double[number + 1];
suite[0] = 0;

    if (number == 0) {
        return suite;
    }

    suite[1] = 1;

    for (int i = 2; i <= number; i++) {
        suite[i] = suite[i - 1] + suite[i - 2];
    }

    return suite;
}

Generation of the EJB related files

To generate the EJB related classes and descriptors, we need to make some XDoclet configuration. With JBoss IDE, you can define several XDoclet generation configurations that will be run against the project.
Edit the project properties by right clicking on the project and select “Properties”.

In the property page, select “XDoclet configurations”.

Right-click in the upper area to pop-up the menu and choose "Add". Type "EJB" in the dialog and click "Ok".

You have created a new generation configuration named "EJB".

Select the “EJB” configuration.

In the lower-left area, right-click to popup the menu and choose "Add Doclet".

A list of available doclets will appear. Choose "ejbdoclet" and click "Ok".

On the lower-right area, you see the properties of the "ejbdoclet". Set them to:

- "destDir" with "src" and check it
- "ejbSpec" with "2.0" and check it

Our configuration now contains an "ejbdoclet" that will produce files in "src" folder and for the EJB 2.0 specifications.
In the lower-left area, right-click on "ejbdoclet" to popup the menu and choose "Add ".

A list of available subtasks will appear. Choose "fileset" and click "Ok".

On the lower-right area, you see the properties of the "fileset ". Set them to:
- "dir" with "src" and ckeck it
- uncheck "excludes"
- "includes" with "**/*.Bean.java" and ckeck it

Our configuration now contains an "ejbdoclet" with a "fileset" based on "src" and that filters source files to only pick up the EJB ones.

Repeat the operation to add a new subtask "deploymentdescriptor". The property to set is "destDir" with "src/META-INF". Don't forget to check it.

We will generate the deployment descriptor in the "src/META-INF" folder to remain simple.
Repeat the operation to add a new subtask "jboss". The properties to set are "destDir" with "src/META-INF" and "Version" with "3.0". Don't forget to check them.

We will generate the jboss deployment descriptor in the "src/META-INF" folder and for the 3.0 version.

Repeat the operation to add a new subtask "packageSubstitution". The properties to set are "packages" with "ejb" and "substituteWith" with "interfaces". Don't forget to check them.

We will generate the EJB related classes in a different package (EJB lies in tutorial.ejb, interfaces will be generated in tutorial.interfaces).
Repeat the operation to add new subtasks "remoteinterface" and "homeinterface". No properties are to be set.

These subtasks are to generate the EJB Home and Remote interfaces.

Click "Ok" to save the generation configuration. This action will generate an Ant build file "xdoclet-build.xml" in the project. This file contains the generation configuration ready to be launch.

Right-click on the project and select "Run XDoclet". The XDoclet generation will display its output in the console. The output looks like this one.
After the generation, you should have a project like this. Note that a "tutorial.interfaces" package has been created with new classes inside. There is also a "META-INF" folder with the deployment descriptors (standard and jboss ones).

**The Servlet and the Web-App**

Having an EJB is not enough. We will write a servlet that access this EJB to perform the actual computation of the Fibonacci suite.

Create a new Java Class. The package will be “tutorial.web” and the class name “ComputeServlet”. Click on “Browse...” to add the “HTTPServlet” superclass. Be sure that only “Constructors from superclass” and “Inherited abstract methods” are checked.
Click on “Finish”. The class is then created and you should have a project like this.

Open the **ComputeServlet.java** file. To be interesting, our servlet some initialization and some processing code.

Right-click in the editor area and select "Source > Override/Implement Methods...". Check "init" and "doPost" methods and click "Ok". Two method stubs will be generated.

Add the two following private members.

```java
private FiboHome home;
private String value;
```
Define the "init" method like this. This code is responsible for the initialization of the EJB Home interface and the for grab of the environment entry.

```java
public void init() throws ServletException {
    try {
        Context context = new InitialContext();
        value = (String) context.lookup("java:/comp/env/Title");
        Object ref = context.lookup("java:/comp/env.ejb/Fibo");
        home = (FiboHome) PortableRemoteObject.narrow(ref, FiboHome.class);
    } catch (Exception e) {
        throw new ServletException("Lookup of java:/comp/env/ failed");
    }
}
```

Define the "doPost" method like this. The code will parse the request to get the limit parameter, create an instance of the EJB, performs computation, releases the instance and output the result as HTML.

```java
protected void doPost(
    HttpServletRequest request,
    HttpServletResponse response)
        throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();

out.println("<html><head><title>");
out.println(value);
out.println("</title></head>");
out.println("<body>");
out.println("<h1>");
out.println(value);
out.println("</h1>");

try {
    Fibo bean = home.create();
    int limit = 0;
    String value = request.getParameter("limit");
    if (value != null) {
        try {
            limit = Integer.parseInt(value);
        } catch (Exception e) {
        }
    }
    double[] result = bean.compute(limit);
    bean.remove();
    out.println("<p>");
    out.println("The ");
```
The next step is to insert XDoclet javadoc related tags for the Servlet. In the Java editor go in the Javadoc class paragraph. Type “@web.” And press CTRL+Space. You should see the magic of completion in action.

Complete the attributes of the tag with the following values (press CTRL+Space for each attribute if you want the completion):

```java
* @web.servlet name = "ComputeServlet"
* display-name = "Computation Servlet"
* description = "Servlet that compute Fibonacci suite"
*
* @web.servlet-mapping url-pattern = "/Compute"
```
public class ComputeServlet extends HttpServlet { 

After that, the file should look like this. Now, we are ready to run XDoclet on the file generate all the Web stuff.
public class ComputeServlet extends HttpServlet {
    private FiboHome home;
    private String value;

    public ComputeServlet() {
        super();
    }

    public void init() throws ServletException {
        try {
            Context context = new InitialContext();
            value = (String) context.lookup("java:/comp/env/Title");
            Object ref = context.lookup("java:/comp/env/ejb/Fibo");
            home = (FiboHome) PortableRemoteObject.narrow(ref,
                    FiboHome.class);
        } catch (Exception e) {
            throw new ServletException("Lookup of java:/comp/env/Title failed");
        }
    }

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();

            out.println("<html><head><title>");
            out.println(value);
            out.println("</title></head>");
            out.println("<body>");

            out.println("<hl>");
            out.println(value);
            out.println("</hl>");

            try {
                Fibo bean = home.create();
                int limit = 0;
                String value = request.getParameter("limit");
                if (value != null) {
                    try {
                        limit = Integer.parseInt(value);
                    } catch (Exception e) {
                    }
                }
            }
        }
double[] result = bean.compute(limit);
bean.remove();

out.println("<p>");
out.print("The ");
out.print(limit);
out.print(" first Fibonacci numbers ");

for (int i = 0; i < result.length; i++) {
    out.println("<br>");
    out.println(i);
    out.println(" : ");
    out.println(result[i]);
}

out.println("</p>");
} catch (Exception e) {
    out.println(e.getMessage());
e.printStackTrace(out);
} finally {
    out.println("</body"/></html>");
    out.close();
}
}

**Generation of the Servlet related files**

To generate the Web descriptors, we need to make some XDoclet configuration. Like EJB, we will define a generation configuration for the Web stuff.

Edit the project properties by right clicking on the project and select “Properties”.

In the property page, select “XDoclet configurations”.

Right-click in the upper area to pop-up the menu and choose "Add". Type "Web" in the dialog and click "Ok".

You have created a new generation configuration named "Web".
Select the “Web” configuration.

In the lower-left area, right-click to popup the menu and choose "Add Doclet".

A list of available doclets will appear. Choose "webdoclet" and click "Ok".

On the lower-right area, you see the properties of the "webdoclet". Set them to:

- "destDir" with "src/WEB-INF" and check it

Our configuration now contains an "webdoclet" that will produce files in "src/WEB-INF" folder.

In the lower-left area, right-click on "webdoclet" to popup the menu and choose "Add".

A list of available subtasks will appear. Choose "fileset" and click "Ok".

On the lower-right area, you see the properties of the "fileset". Set them to:

- "dir" with "src" and check it
- uncheck "excludes"
- "includes" with "**/*Servlet.java" and check it

Our configuration now contains an "webdoclet" with a "fileset" based on "src" and that filters source files to only pick up the Servlet ones.
Repeat the operation to add a new subtask "deploymentdescriptor". The property to set is "Servletspec" with "2.3". Don't forget to check it.

We will generate the deployment descriptor in the "src/WEB-INF" folder to remain simple (property inherit from "webdoclet").

Repeat the operation to add a new subtask "jbosswebxml". The property to set is "Version" with "3.0". Don't forget to check it.

We will generate the jboss deployment descriptor in the "src/WEB-INF" folder (property inherit from "webdoclet").

Click "Ok" to save the generation configuration. This action will update the Ant build file "xdoclet-build.xml" in the project. This file now contains the generation configurations for the EJB and for the Servlet.

Right-click on the project and select "Run XDoclet". The XDoclet generation will display its output in the console. The output looks like this one.
After the generation, you should have a project like this. Note that a "WEB-INF" folder has been created with the Web deployment descriptor (standard and jboss ones).

A Servlet is useless unless some parameters are passed. To pass these parameters to the Servlet, a HTML page with a form is needed.

Create a "docroot" folder under the root of the project. Create an empty file named "index.html". The "index.html" file is intended to be the default page for the Web application and contains a form that will be posted to the Servlet.

The content of "index.html" is the following:

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>
    Fibonacci Application
  </title>
</head>
```
The J2EE Application

This project is intended to be a complete J2EE application. We are going to create some additional files to have all the material needed to build it.

In the "src/META-INF" folder, create a file named "application.xml". The "application.xml" file is the J2EE application descriptor that will point to the EJB package and to the War package.
The content of "application.xml" is the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN" "http://java.sun.com/dtd/application_1_3.dtd">
<application>
  <display-name>Sum Application</display-name>
  <module>
    <ejb>FiboEJB.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>FiboWeb.war</web-uri>
      <context-root>/fibo</context-root>
    </web>
  </module>
</application>
```

**The Packaging**

JBoss-IDE provides an easy way to configure the packaging of various archive. There is no restriction of what can be packaged.

In this tutorial, four packaging configuration will be defined:

- One for the EJB Jar. It will contain the EJB classes and interfaces, as well as the deployment descriptors.
- One for the EJB client Jar. It will contain the EJB interfaces.
- One for the Web Application War. It will contain the Servlet class, the EJB client Jar, as well as the deployment descriptors.
- One for the J2EE application Ear. It will contain the EJB Jar and the Web Application War, as well as the deployment descriptor.

When launched, these four packaging configurations will create the J2EE application ready to be deployed.
Edit the project properties by right clicking on the project and select “Properties”.

In the property page, select “Packaging configurations”.

Right-click in the area to pop-up the menu and choose "Add Archive". Type "FiboEJB.jar" in the dialog and click "Ok".

You have created a new packaging configuration that will produce the “FiboEJB.jar” file.

We want to add the EJB classes and interfaces. Eclipse has generated the compiled classes into the src/bin folder (declared as the default output dir of the project).

Select the “FiboEJB.jar” item and right-click in the area to pop-up the menu and choose "Add Folder". A “Folder Selection” dialog appears.

This dialog allows to select which folder (local to workspace or in the file system) to include into the package, to specify include and exclude filters (A la Ant) and to set a prefix that will be append when building the package.

Click on “Project Folder”. A “Folder Chooser” dialog appears.
This dialog allows selecting which folder to include. This folder can be chosen among all the opened projects.
Select the “/Tutorial/bin” folder and click “Ok”.

The folder is now “/Tutorial/bin”.
As we only want the EJB classes and interfaces, specify the following as include filter:
“tutorial/ejb/*.class,tutorial/interfaces/*.class”
Click on “Ok”.
We now want to add the standard EJB deployment descriptor.
Select the “FiboEJB.jar” item and right-click in the area to pop-up the menu and choose "Add File". A “File Selection” dialog appears.

This dialog allows to select which file (local to workspace or in the file system) to include into the package and to set a prefix which will be append when building the package.
Click on “Project File”. A “File Chooser” dialog appears.

This dialog allows to select which file to include. This file can be choosen among all the opened projects.
Select the “/Tutorial/src/META-INF/ebj-jar.xml” folder and click “Ok”.

We now want to add the standard EJB deployment descriptor.
Select the “FiboEJB.jar” item and right-click in the area to pop-up the menu and choose "Add File". A “File Selection” dialog appears.
The file is now “/Tutorial/src/META-INF/ejb-jar.xml”. The “ejb-jar.xml” must be located under the “META-INF” of the EJB package. Set the prefix to “META-INF”. Click on “Ok”.

To add the specific EJB deployment descriptor, select the “FiboEJB.jar” item and right-click in the area to pop-up the menu and choose "Add File". The file to choose is “/Tutorial/src/META-INF/jboss.xml”. The “jboss.xml” must be located under the “META-INF” of the EJB package. Set the prefix to “META-INF”. Click on “Ok”.

The packaging configuration for the “FiboEJB.jar” is now completed.
Right-click in the area to pop-up the menu and choose "Add Archive". Type "FiboEJB-client.jar" in the dialog and click "Ok".

You have created a new packing configuration that will produce the “FiboEJB-client.jar” file.

Select the “FiboEJB-client.jar” item and right-click in the area to pop-up the menu and choose "Add Folder". A “Folder Selection” dialog appears.

Click on “Project Folder” and select the “/Tutorial/bin” folder from the “Folder Chooser” dialog.

As we only want the EJB interfaces, set the include filter to “tutorial/interfaces/*.class”.

Click on “Ok”.

The packaging configuration for the “FiboEJB-client.jar” is now completed.
Right-click in the area to pop-up the menu and choose "Add Archive". Type "FiboWeb.war" in the dialog and click "Ok".

You have created a new packaging configuration that will produce the “FiboWeb.war” file.

Select the “FiboWeb.war” item and right-click in the area to pop-up the menu and choose "Add Folder". A “Folder Selection” dialog appears.

Click on “Project Folder” and select the “/Tutorial/bin” folder from the “Folder Chooser” dialog.

As we only want the Servlet class, set the include filter to “tutorial/web/*.class”.

The classes must be located under the “WEB-INF/classes” of the War package. Set the prefix to “WEB-INF/classes”.

Click on “Ok”.

To add the standard Web deployment descriptor, select the “FiboWeb.war” item and right-click in the area to pop-up the menu and choose "Add File". A “File Selection” dialog appears.

The file to choose is “/Tutorial/src/WEB-INF/web.xml”.

The “web.xml” must be located under the “WEB-INF” of the War package. Set the prefix to “WEB-INF”.

Click on “Ok”.
To add the specific Web deployment descriptor, select the “FiboWeb.war” item and right-click in the area to pop-up the menu and choose "Add File". A “File Selection” dialog appears.

The file to choose is “/Tutorial/src/WEB-INF/jboss-web.xml”.

The “jboss-web.xml” must be located under the “WEB-INF” of the War package. Set the prefix to “WEB-INF”.

Click on “Ok”.

To add the EJB Client Jar, select the “FiboWeb.war” item and right-click in the area to pop-up the menu and choose "Add File". A “File Selection” dialog appears.

The file to choose is “/Tutorial/FiboEJB-client.jar”. But it doesn’t exist yet as the packaging has not been run. Instead of selecting it, go in the text field and type the name of the file “/Tutorial/FiboEJB-client.jar”. Even if the file doesn’t exist, it can be added to a packaging configuration.

The “FiboEJB-client.jar” must be located under the “WEB-INF/lib” of the War package. Set the prefix to “WEB-INF/lib”.

Click on “Ok”.
Select the “FiboWeb.war” item and right-click in the area to pop-up the menu and choose "Add Folder". A “Folder Selection” dialog appears.

Click on “Project Folder” and select the “/Tutorial/docroot” folder from the “Folder Chooser” dialog. This is the content of the Web Application.

Click on “Ok”.

The packaging configuration for the “FiboWeb.war” is now completed.
Right-click in the area to pop-up the menu and choose "Add Archive". Type "FiboApp.ear" in the dialog and click "Ok".

You have created a new packaging configuration that will produce the “FiboApp.ear” file.

To add the application deployment descriptor, select the “FiboApp.ear” item and right-click in the area to pop-up the menu and choose "Add File". A “File Selection” dialog appears.

The file to choose is “/Tutorial/src/META-INF/application.xml”.

The “application.xml” must be located under the “META-INF” of the Ear package. Set the prefix to “META-INF”. Click on “Ok”.

To add the EJB module, select the “FiboApp.ear” item and right-click in the area to pop-up the menu and choose "Add File". A “File Selection” dialog appears.

The file to choose is “/Tutorial/FiboEJB.jar”. But it doesn’t exist yet as the packaging has not been run. Instead of selecting it, go in the text field and type the name of the file “/Tutorial/FiboEJB.jar”. Even if the file doesn’t exist, it can be added to a packaging configuration. Click on “Ok”.

---

Page 38 of 49
To add the Webmodule, select the “FiboApp.ear” item and right-click in the area to pop-up the menu and choose "Add File". A “File Selection” dialog appears.

The file to choose is “/Tutorial/FiboWeb.war”. But it doesn’t exist yet as the packaging has not been run. Instead of selecting it, go in the text field and type the name of the file “/Tutorial/FiboWeb.war”. Even if the file doesn’t exist, it can be added to a packaging configuration.

Click on “Ok”.

The packaging configuration for the “FiboApp.ear” is now completed.
Click "Ok" to save the packaging configuration. This action will create the Ant build file "packaging-build.xml" in the project. This file now contains the packaging configurations for the full J2EE Application.

Right-click on the project and select "Run Packaging". The packaging will display its output in the console. The output looks like this one.
After the execution, you should have a project like this.

**JBoss Configuration and Launch**

Now, it is time to configure the JBoss server if not done.

Click on the debug shortcut and select "Debug..." to open the debug configurations.
The debug dialog allow to configure the available JBoss configurations that will be used for debugging.

Select the configuration you want to launch and click on "Debug" and you will see JBoss starting, the output is made in the console.

**Deployment**

The deployment within JBoss-IDE can be done in two ways:

- A file-system copy. It is like copy and paste the resource from a file explorer.
- A local deployment through the MainDeployer MBean (Experimental). The URL of the resource is sent to the MainDeployer Mbean, which deploys and watches it.

In addition, the deployment target is stored during the workbench session. This means that if you have deployed a package on a target, you can redeploy or undeploy it without specifying the target.

The Deployer plugin automatically creates file-system targets from the debug configurations. Other deployment target can be defined.
Select “Window > Preferences”. The workbench preferences appears.

Select “JBoss-IDE > Deployer” to display the defined deployment targets.

The upper area contains the file system targets build upon the debug configuration defined.

The lower area contains the user-defined deployment targets.

We assume that we want to deploy to a pre-defined JBoss instance and we don’t define custom deployment targets.

The deployment is fairly simple. Right click on the “FiboApp.ear” file and select the "Deployment" > "Deploy To..." item.

A dialog box appears with the list of the deployment targets. It contains both the default and the user-defined deployment targets.

Select the one you are interested in.

If the deployment is successful, a dialog should pop-up to tell it.
In the console view, you should see some deployment activity. The J2EE application is now deployed.

When a resource is deployed, a small decorator appears in the top-left corner of the icon.

**Debugging**

Prior to the debugging, we need to set some breakpoints inside the code.

Open the `FiboBean.java` file. Double click in left column to create a breakpoint.

In the example, the breakpoint is set in front of the test.
Open the `ComputeServlet.java` file. Double click in left column to create a breakpoint.

In the example, the breakpoint is set in front of the EJB creation.

Open a web browser and type `http://localhost:8080/fibo/`. The host/port can change if the web server listens on another host/port. You should a simple form like the one above. Enter a positive value in the field and press "Compute".

Switch to your Eclipse workbench. You should see that execution has been suspended on the first breakpoint (in the servlet). You can go step by step in the code or go on with execution.
Another suspension must occurred when hitting the second breakpoint (in the EJB). You can go step by step in the code or go on with execution.
After resuming the execution and back to the browser, the response should be like above.
Conclusion

This simple tutorial is intended to give an overview of what is possible with JBoss IDE. We hope that it will be useful for developers who want to go with it.