3-2006

The JBoss Integration Plug-in for IntelliJ IDEA, Part 3

Douglas A. Lyon
Fairfield University, dlyon@fairfield.edu

Martin Fuhrer

Thomas Rowland

Published Citation

Douglas A. Lyon, Martin Fuhrer, Thomas Rowland, “The JBoss Integration Plug-in for IntelliJ IDEA, Part 3”, Journal of Object Technology, Volume 5, no. 2 (March 2006), pp. 13-26

JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2006**

Vol. 5, No. 2, March-April 2006

The JBoss Integration Plug-in for IntelliJ IDEA, Part 3.

Douglas Lyon, Fairfield University, Fairfield CT, U.S.A.
Martin Fuhrer, President of Furher Engineering AG, Biel, Switzerland
Thomas Rowland, Pitney Bowes, Shelton CT, U.S.A.

In nature,
animals without a spine
have the hardest shells.
- Anon

Abstract

This paper is the third in a series of papers that describe a new plug-in for enabling the integration of the IntelliJ IDEA IDE with the JBoss application server. The JBoss plug-in was first conceived and implemented by Martin Fuhrer at Fuhrer Engineering.

Part 1 discussed how to download and install the new JBoss plug-in, allowing the JBoss application server to integrate into the IntelliJ IDEA IDE development environment. It then demonstrated how to create a project with EJBs and web modules.

Part 2 discussed how to create a session bean in our project. The session bean contained the implementation for the functionality that we wish to expose to the client.

This paper continues to build upon our project by describing how to add a servlet for accessing the EJB methods implemented previously, and then how to create an application module for deployment to the JBoss application server.

1 CREATING A SERVLET

This section describes how to create a servlet that will make use of the EJB that was created in part 2 of this paper. One of the critical elements will be setting up the execution environment of the servlet in order to make the EJB available. This is accomplished by declaring a reference in the web module's deployment descriptor to the EJB's home interface. Once again, IntelliJ IDE wizards provide a GUI for the synthesis of the needed resources. During runtime, the servlet will use JNDI to look up the interface and create an object that can be used to invoke the EJB methods.

Cite this column as follows: Douglas Lyon, Martin Fuhrer and Thomas Rowland: "The JBoss Integration Plug-in for IntelliJ IDEA, Part 3. ", in *Journal of Object Technology*, vol. 5, no. 2, March-April 2006, pp. 13-26 http://www.jot.fm/issues/issue_2006_03/column2

1.1. Creating a Servlet

Right-click (or control-click, for the Mac) on the web module in the project JTree and select the *New:Servlet* menu item, as shown in Figure 1.1.

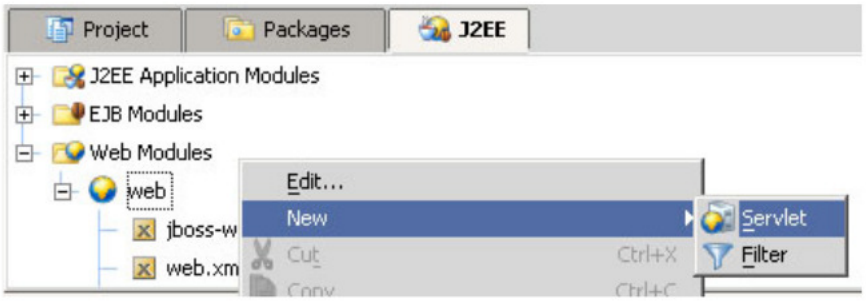

Figure 1.1 Creating a new servlet

In the *New Servlet* dialog, enter the servlet name and package, as shown in Figure 1.2, and select *OK*.

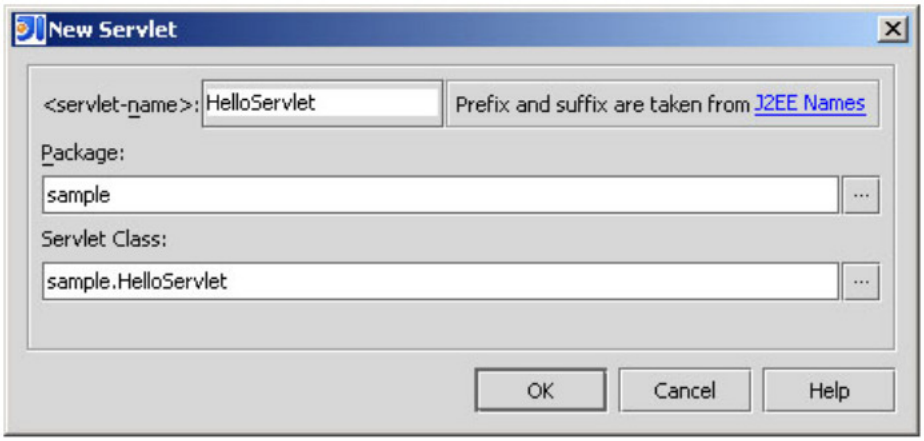

Figure 1.2 The New Servlet dialog

We now have an empty servlet. The *Servlet* dialog will be displayed, which you may close.

1.2. Mapping an EJB Reference

The *HelloServlet* has the role of providing a GUI for the EJB. The servlet locates the bean by its logical (reference) name *ejb/hello* and not by the real JNDI name. To accomplish this we first create an EJB reference, and then map the reference to the EJB's real JNDI name.

Close the *HelloServlet* class and open the *Web Module Properties* by right-clicking on the web module in the project JTree and selecting the *Edit* menu item.

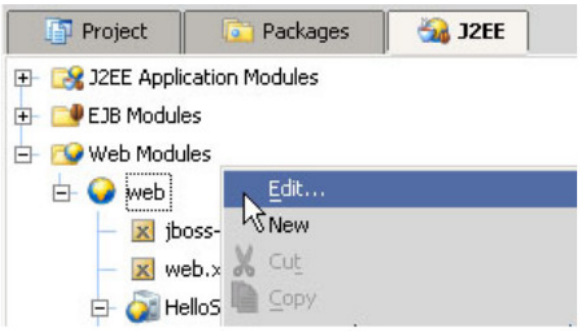

Figure 1.3 Accessing the web module properties in the Project JTree

Click the "+" sign below the *Ejb References Configured* label in the web properties dialog, as shown in Figure 1.4.

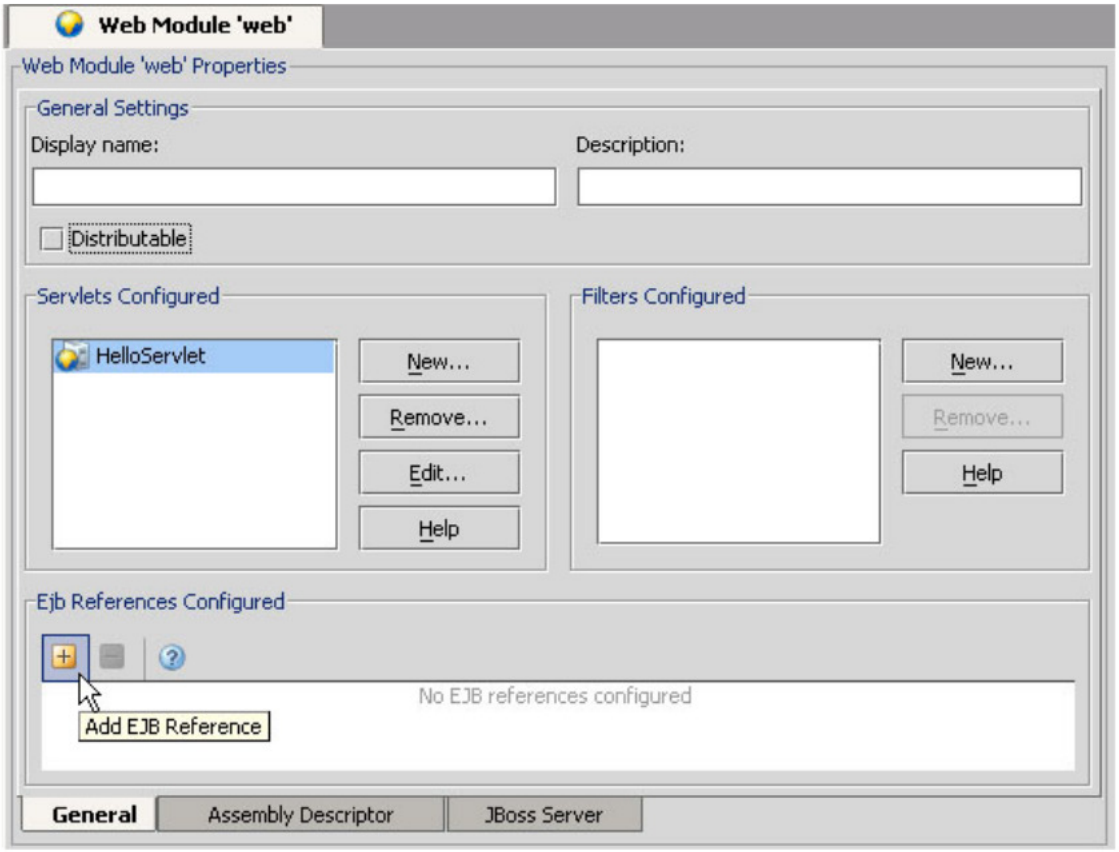

Figure 1.4 The Web Properties dialog

This will bring up the *Create New EJB Reference* dialog. Select the home interface, *HelloHome* (or *LocalHelloHome* in the case of a local EJB) and enter the logical name of

the bean, as it is used in the servlet (recall that *ejb/hello* was used in the servlet when making the call to JNDI lookup). Also note that cross module links will not work in the deployed environment. As a result, you should delete the entry that appears in the EJB reference's *Link* field. The dialog is shown in Figure 1.5 for the case of the remote EJB and Figure 1.6 for the case of the local EJB.

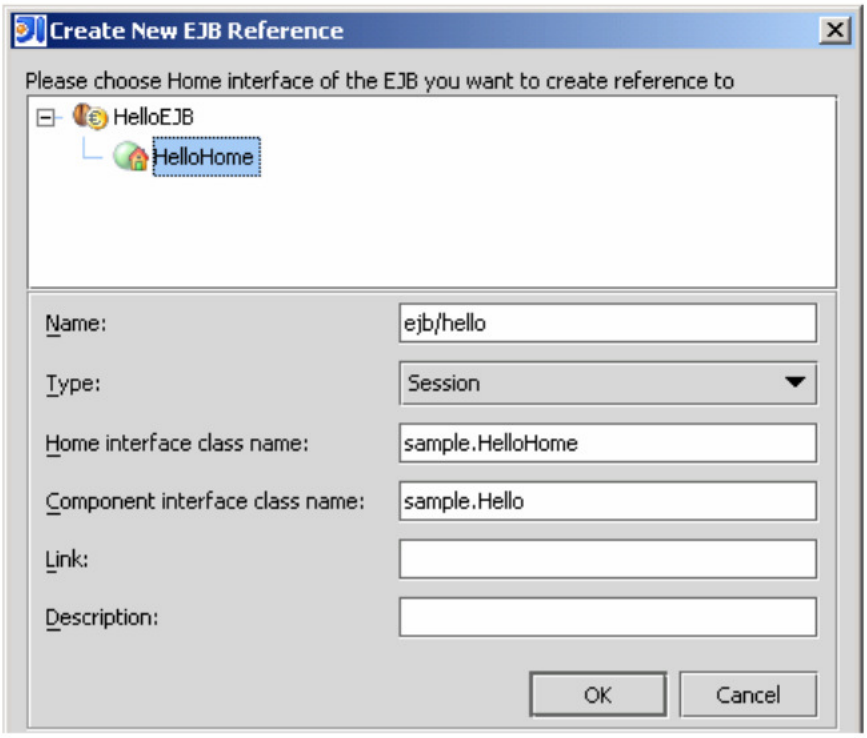

Figure 1.5 Setting the EJB Reference properties for a remote EJB

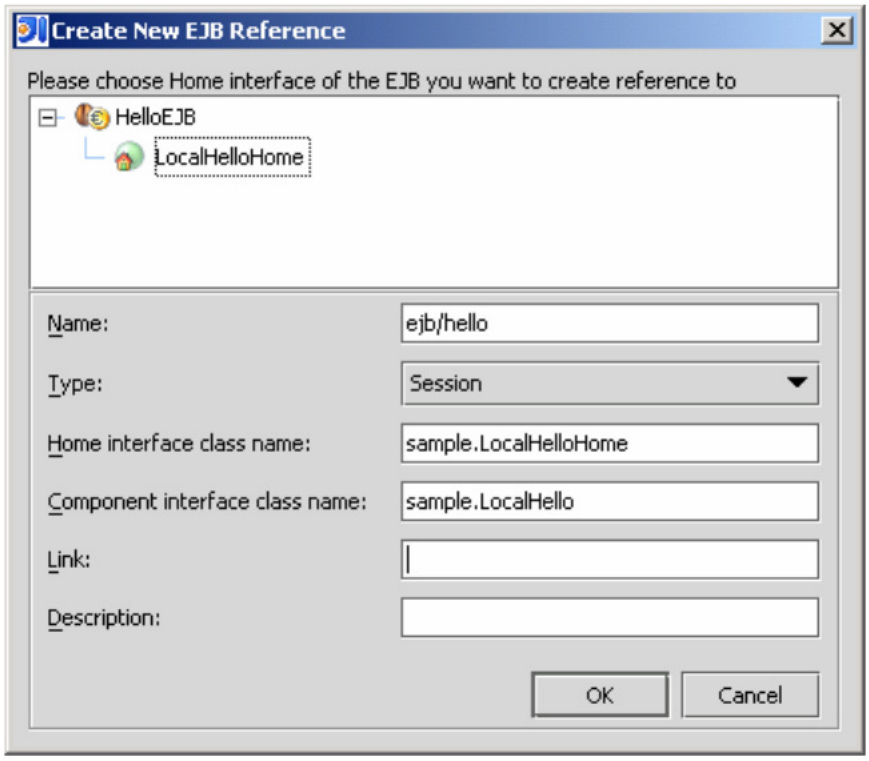

Figure 1.6 Setting the EJB Reference properties for a local EJB

Select *OK* to save and close the dialog. The *Web Module Properties* dialog now shows the newly created EJB reference. Figure 1.7 shows an EJB reference to the remote EJB.

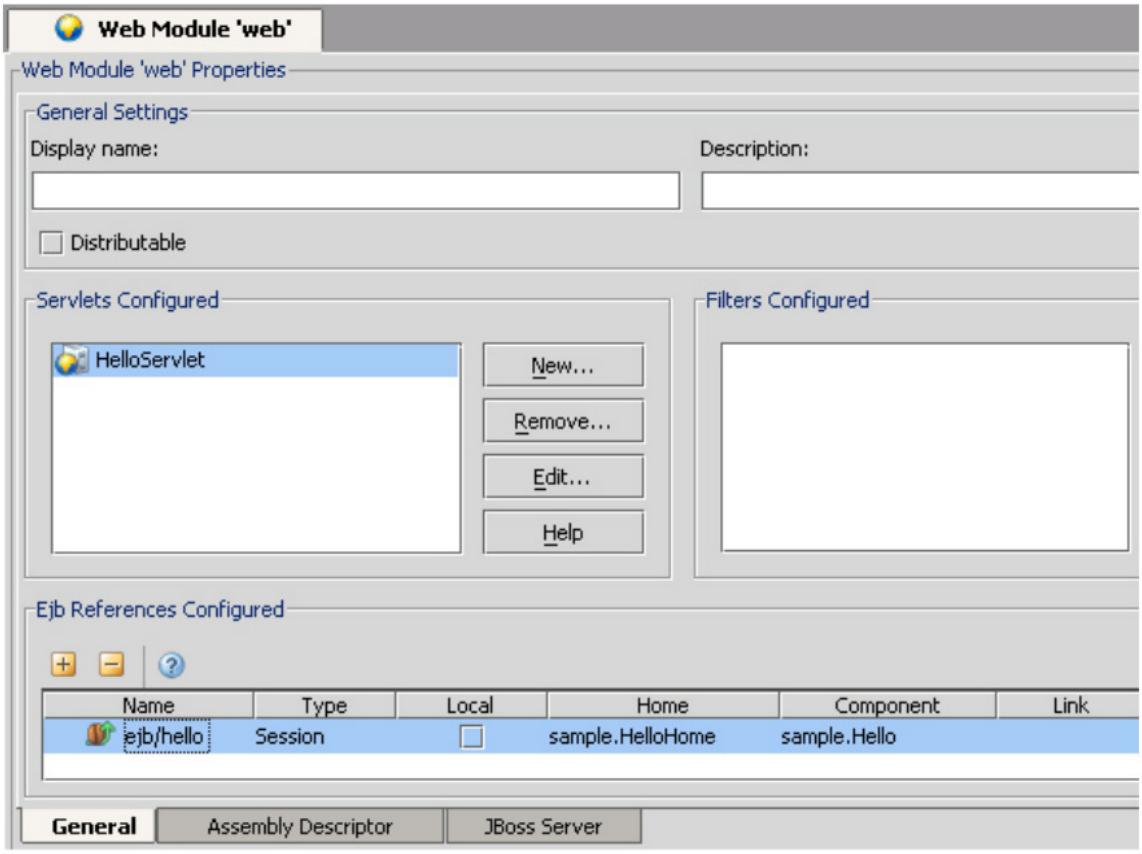

Figure 1.7 Web Module Properties dialog showing a reference to a remote EJB

If you are referencing the local EJB then the *Local* checkbox will be checked. The reference is shown in Figure 1.8.

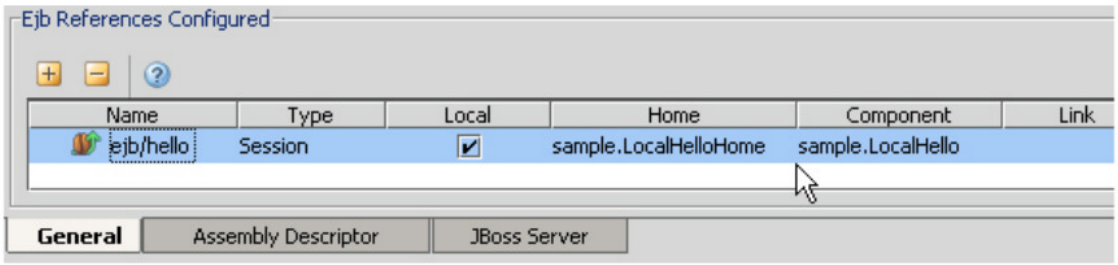

Figure 1.8 Web Module Properties dialog showing a reference to a local EJB

Select the *JBoss Server* tab and map the *Reference Name* of the EJB to the *JNDI Name* (recall that *hello* was specified as the JNDI name when creating the EJB), as shown in Figure 1.9.

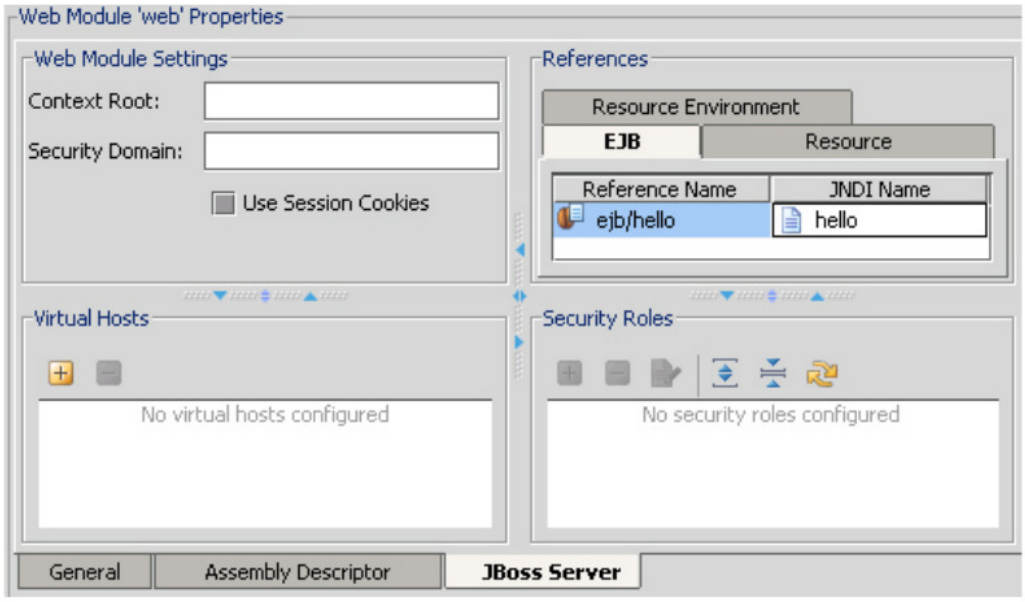

Figure 1.9 Mapping the EJB Reference Name to its JNDI Name

1.3. Mapping a Friendly URL for the Servlet

We can define a URL pattern to access the servlet. To do so we map the URL pattern to the servlet name, as shown below.

Select the *Assembly Descriptor* tab. Select the "+" icon under *Servlet Mappings* to reveal a *URL Pattern*, and enter in a URL pattern, as shown in Figure 1.10.

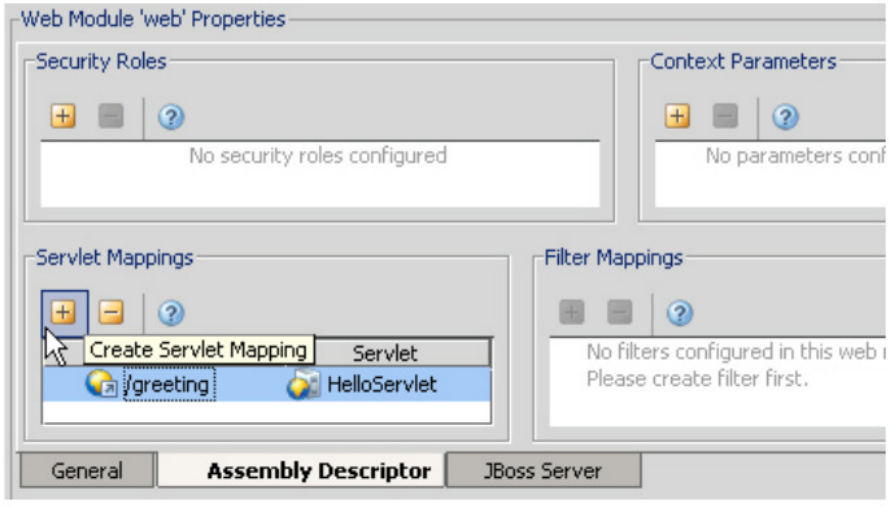

Figure 1.10 Mapping of a friendly URL to the Servlet

1.4. Modifying the Servlet Class to Access the EJB Methods

Recall that we added the methods `hello` and `getDate` to our session bean. Now we want our servlet to act as a client for invoking these methods, so we need to add this functionality to the servlet class. Remember that we showed how to create a local bean as well as how to create a remote bean. The only differences between the two in our servlet code will be the names of the bean interfaces.

Open the *HelloServlet* class from the project *JTree*.

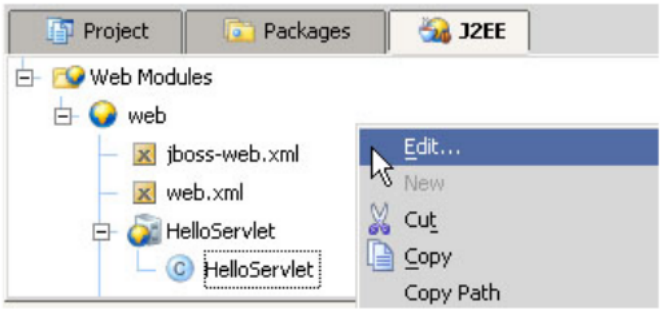

Figure 1.11 Opening the HelloServlet from the project JTree

Modify the *HelloServlet* code, as shown in Figures 1.12 and 1.13.

```
public class HelloServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
            throws ServletException, IOException {
        process(response);

    }

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {
        process(response);

    }

    private void process(HttpServletResponse response)
            throws IOException {
        ServletOutputStream out = response.getOutputStream();
        try {
            synthesizeOutput(out);
        } catch (Exception e) {
            out.println("<html><body>"
            + e.getMessage() + "</body></html>");
        }
    }

    private void synthesizeOutput(ServletOutputStream out)
```

```
            throws NamingException, CreateException, IOException
        {
        HelloHome home = (HelloHome) new
          InitialContext().lookup("java:comp/env/ejb/hello");
        Hello hello = home.create();
        out.println("<html><body><h2>" + hello.sayHello()
                    + " - " + hello.getDate()
                    + "</h2></body></html>");
    }
}
```

Figure 1.12 An implementation for accessing the remote *HelloServlet* EJB

Consider the local bean implementation. The code will be the same as for the remote bean implementation except that the JNDI lookup needs to reference the *LocalHello* component interface and the *LocalHelloHome* home interface.

For a servlet that implements a local interface, the `synthesizeOutput` method will then look like that shown in Figure 1.13.

```
    private void synthesizeOutput(ServletOutputStream out)
            throws NamingException, CreateException, IOException
        {
        LocalHelloHome home = (LocalHelloHome) new
          InitialContext().lookup("java:comp/env/ejb/hello");
        LocalHello hello = home.create();
        out.println("<html><body><h2>" + hello.sayHello()
                    + " - " + hello.getDate()
                    + "</h2></body></html>");
    }
```

Figure 1.13 A `SynthesizeOutput` method implementation for accessing the local *HelloServlet* EJB

Now that your servlet is complete, you are ready to create an application module.

2 CREATING AN APPLICATION MODULE

This section describes how to create an Enterprise Application Archive (i.e., an EAR file). An EAR file is an archive containing EJBs, resource adapters, web modules, and possibly other application modules [Chan]. The EAR file will serve as a container for our EJB module (ejb.jar, containing our session bean and EJB deployment descriptor files) and our web module (web.war, containing our servlet and web application deployment descriptor files) that were created in previous section. The EAR file encapsulates the entire J2EE application which will then be deployed from the IntelliJ environment to the JBoss application server.

In order to create an application module, you select *File:Settings* to open the *Settings* dialog box and click on the *Paths* icon, as shown in Figure 2.1.

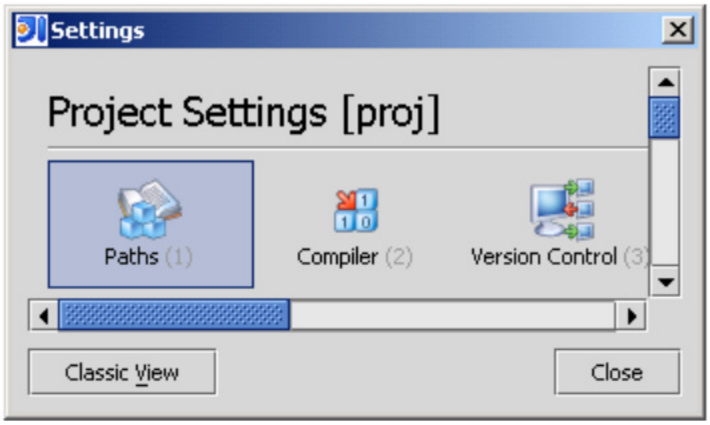

Figure 2.1 The Paths Icon

The *Paths* dialog box is displayed, as shown in Figure 2.2.

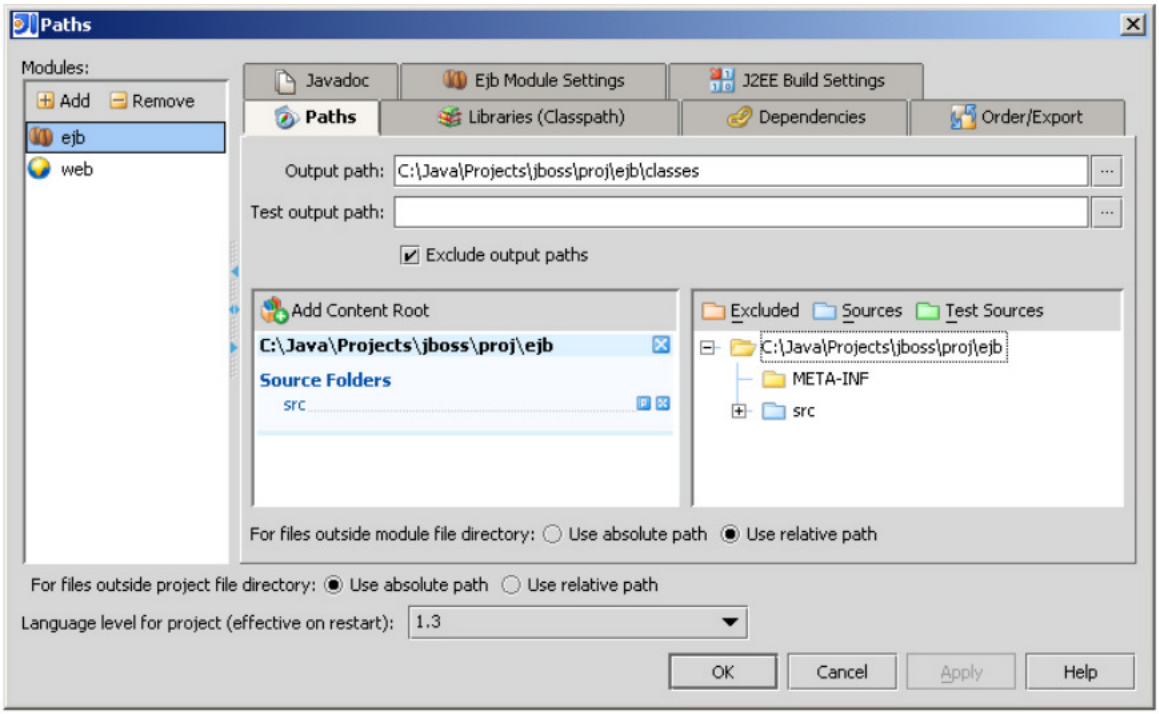

Figure 2.2 The Paths dialog

Select the "+" sign under *Modules* in order to bring up the *Add Module* dialog box, as shown in Figure 2.3.

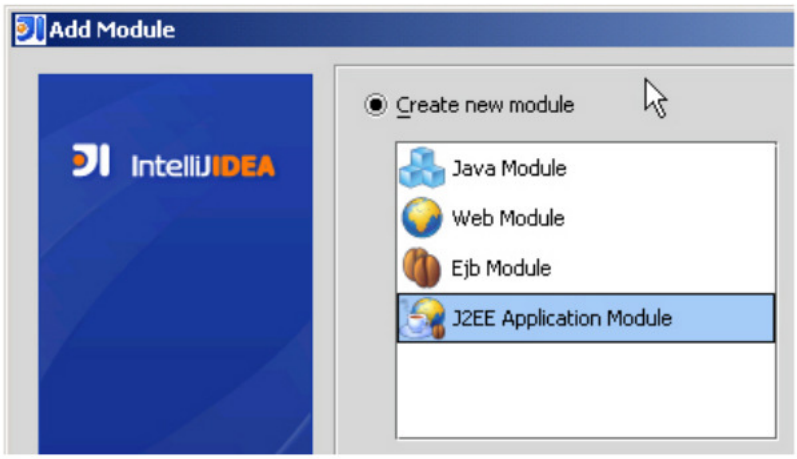

Figure 2.3 The Add Module dialog

Select *J2EE Application Module* and click the *Next* button. Enter the module name into the text field, as shown in Figure 2.4.

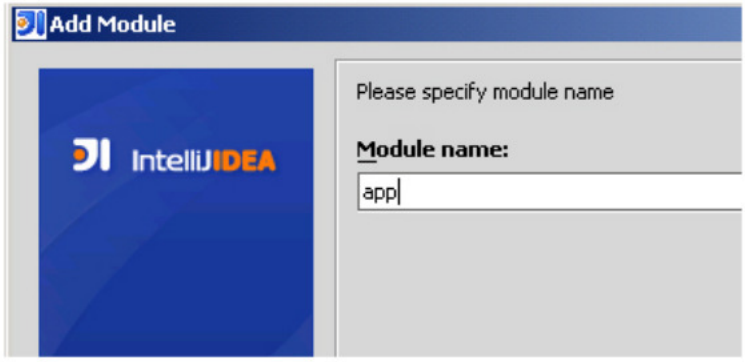

Figure 2.4 Entering the Application Module Name

Select *Next* through the next set of screens until you come to the point where you must specify the J2EE modules to include in your application. Set the *Packaging Method* to *Include Module in Build* for both for the ejb and web modules. Edit the *Web Module Context Root* so that the web module is named *sample* as shown in Figure 2.5.

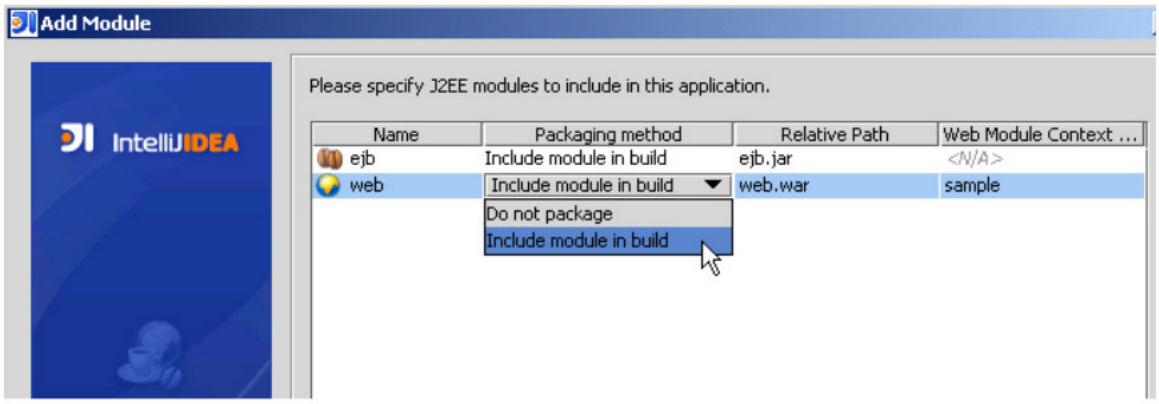

Figure 2.5 Setting the J2EE Modules for inclusion in the Application Module

Select Finish, and the J2EE Application Module Settings will reflect the update in the Modules and Libraries to Package section.

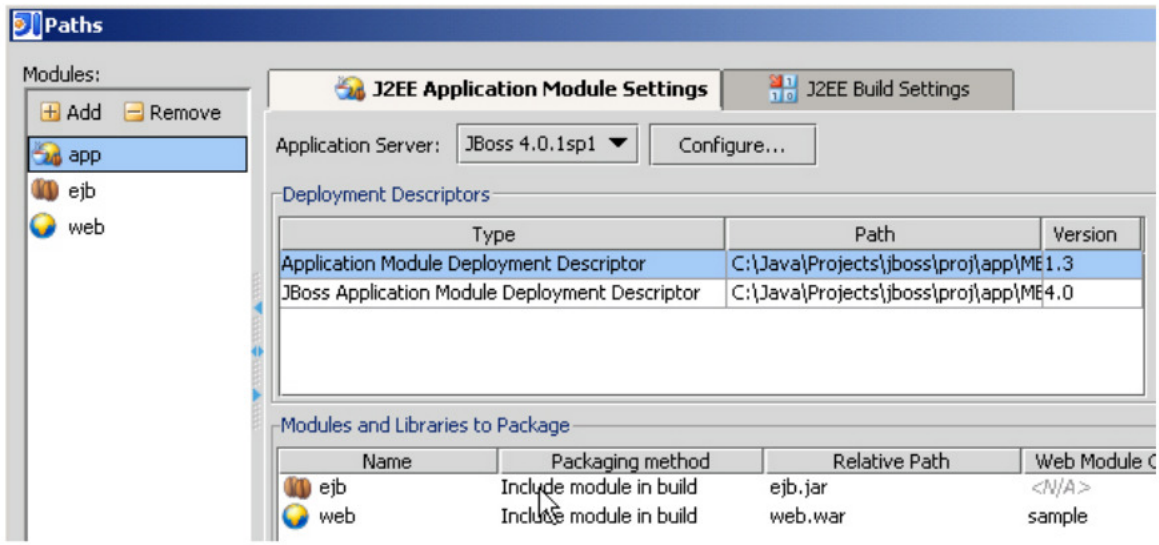

Figure 2.6 The J2EE Application Modules Settings dialog showing the ejb and web modules being included

Click on the *J2EE Build Settings* tab. Select the *Create application archive file* checkbox, as shown in Figure 2.7.

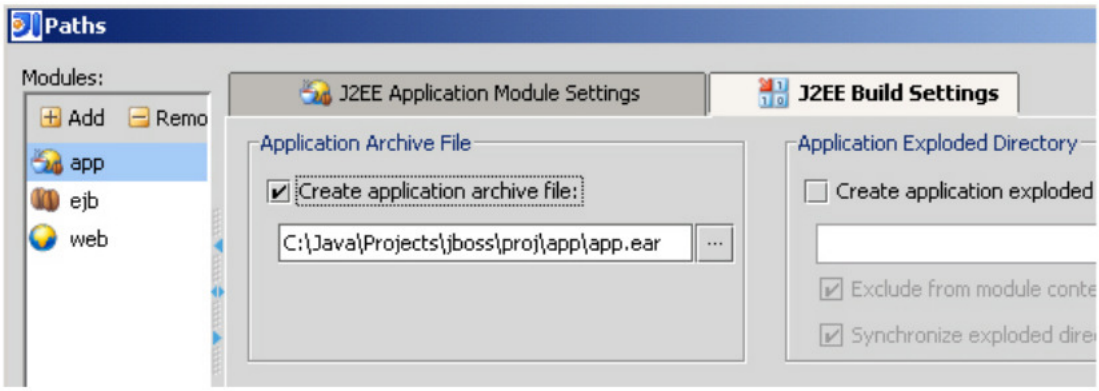

Figure 2.7 Create an Application Archive File.

Select *OK*, and close the *Project Settings* dialog. When you are finished, a new application module can be seen in the project window *JTree*, containing the EJB and Web modules, as shown in Figure 2.8.

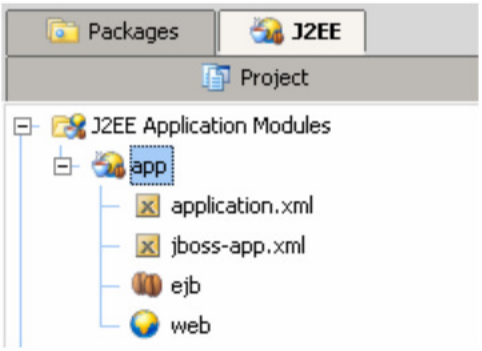

Figure 2.8 A new Application Module containing an EJB module and a Web module

3 CONCLUSION

This paper discussed how to create a servlet that will act as an interface for invoking EJB methods, and then using the data retrieved to produce an output to a web page. To accomplish this we declared a reference to the EJB, which allowed our servlet to use JNDI to look up the bean's home interface. This mapping resides in the web deployment descriptor (web.xml). We also needed to map the EJB's reference name to the real JNDI name, so that we could use the reference name in the JNDI lookup. This mapping resides in the JBoss-specific web deployment descriptor (jboss-web.xml). Luckily, we did not have to modify these XML files directly, as IntelliJ provided a GUI for us. We also saw how to map a user friendly URL into the created servlet.

Finally, this paper discussed the creation of a J2EE application module for deploying our enterprise application to the JBoss application server. The application module consisted of an EAR file that served as a container for the Web and EJB modules. Using

this standard deployment method, our entire application can be deployed as a single archive file.

The next and final paper in this series will focus on deploying and running the application that we have created. It will demonstrate how to choose and create a deployment method to make the application available to the JBoss server. A run configuration will be needed to save options for running and debugging. And finally, seeing our program actually run, and offer some suggestions for some possible problem scenarios.

LITERATURE CITED

[Chan] Allen Chan, "J2EE Application Deployment Considerations", June 11, 2003, http://www.onjava.com/pub/a/onjava/2003/06/11/j2ee_deployment.html

About the authors

After receiving his Ph.D. from Rensselaer Polytechnic Institute, **Dr. Lyon** worked at AT&T Bell Laboratories. He has also worked for the Jet Propulsion Laboratory at the California Institute of Technology. He is currently the Chairman of the Computer Engineering Department at Fairfield University, a senior member of the IEEE and President of DocJava, Inc., a consulting firm in Connecticut. E-mail Dr. Lyon at Lyon@DocJava.com. His website is http://www.DocJava.com.

Martin Fuhrer has a degree as engineer in computer science from the School of Engineering and Information Technology in Biel/Switzerland. He is founder and president of Fuhrer Engineering Inc., a software development company located in Biel/Switzerland. He's mainly working in the field of web-based financial services and the online processing of realtime stock exchange data. He can be reached at info@fuhrer.com or through http://www.fuhrer.com.

Thomas Rowland has a B.S. in Electrical Engineering and an M.S. in Software Engineering. He has been consulting as a Software Engineer for the past four years, working for Pfizer Pharmaceutical, Travelers Life & Annuity, and currently at Pitney Bowes. He has also worked for Hyperion Solutions for over 5 years. Mr. Rowland has also had some teaching stints along the way. He is listed in the National Register's 2005-2006 edition of the Who's Who in Executives and Professionals. He resides in Connecticut and can be reached at rowlandtf@netscape.net.