Borland®

Borland® JBuilder™ and BEA® WebLogic® Server Integration

Integrating JBuilder 5 Enterprise and WebLogic Server 6

by Peter Derry, Borland

Contents

Preface

Borland® JBuilder™ 5 Enterprise is the most comprehensive set of award-winning visual development tools for creating enterprise-scale applications written entirely in the Java™ programming language for the Java 2 platform. JBuilder 5 Enterprise provides integration with the market-leading J2EE platform application servers Borland AppServer™ 4.1/4.5 and BEA® WebLogic™ Server 5.1/6.0.

This paper provides an overview of the development, deployment, and debugging life cycle of Enterprise JavaBeans™ (EJB™) and JSP™ with Borland JBuilder 5 Enterprise and BEA WebLogic Server 6.0.

This paper is not a reference on developing Enterprise JavaBeans™, JSPs/Servlets, or Java. Rather, it provides a jump-start while using JBuilder 5 with WebLogic Server 6.0, allowing developers to reach maximum productivity in the minimum amount of time.

To become familiar with the terminology used in this paper, readers are referred to the *Definitions* and *JBuilder AppBrowser* sections at the end of this paper.

JBuilder™

white paper

Installing JBuilder and WebLogic Server

When you install JBuilder 5 Enterprise, be sure to opt for "Full Installation." This will install JBuilder 5 as well as Borland AppServer 4.5.1 (BAS 4.5.1). BAS4.5.1 must be installed and configured in order to develop EJBs, even if the target AppServer is WebLogic Server. WebLogic Server (service pack 1 or higher) must be installed on the same machine as JBuilder. For more information on installation, refer to the relevant installation guide. Note that JBuilder 5 does not include any software or licenses for WebLogic Server.

Configuring JBuilder for WebLogic Server

In order to enable EJB development, the enterprise setup and default project properties need to be configured. Select enterprise setup (Tools | Enterprise Setup)

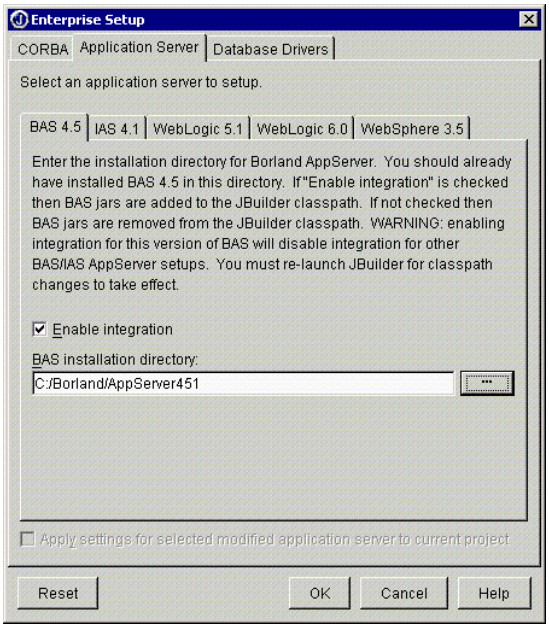


Figure 1: Enterprise Setup Dialog

The enterprise setup dialog enables configuration of JBuilder for CORBA,® application servers and database drivers. For the purposes of this paper, both the application server and CORBA setup will be completed. The EJB 2.0 specification mandates the use of RMI/IIOP for inter-server application interoperability. IIOP is the communications protocol of CORBA; hence, the CORBA capabilities of JBuilder often prove essential even to the EJB developer.

EJB Configuration

Select the application server tab at the top of the Enterprise Setup dialog box. On the BAS 4.5 tab, enter the installation directory of BAS and ensure the "enable integration" check box is selected. This will add the BAS JARs to the JBuilder classpath and enable the JBuilder EJB wizards and utilities once JBuilder is restarted.

Complete the WebLogic Server 6.0 tab by entering both the WebLogic installation directory (default is C:/bea/wlserver6.0) and the BEA home directory (default is C:/bea). The compiler for ejbc usage can also be specified if there is a preferred external compiler to compile the stub files.

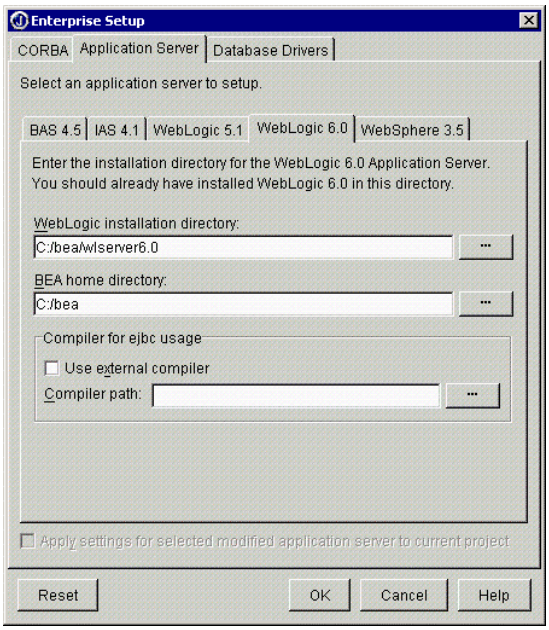


Figure 2: WebLogic 6.0 - Enterprise Setup Dialog

Restart JBuilder to enable the integration. Once restarted, check that the configuration has been successful by creating a new project (File | New Project), opening the Object Gallery (File | New) and checking that the EJB wizards are enabled on the enterprise tab.

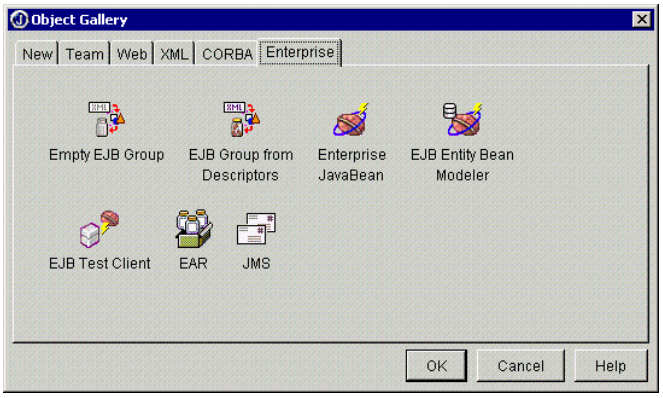


Figure 3: EJB Wizards - Object Gallery

Additionally, this EJB configuration will have created three new libraries (Tools | Configure Libraries) called BAS 4.5, WebLogic 6.0, and WebLogic 6.0 Deploy.

COBRA Configuration

The CORBA capabilities of JBuilder can prove important even for EJB developers. JBuilder ships with Borland AppServer 4.5, which is based upon the market-leading ORB® implementation, VisiBroker® for Java 4.5. JBuilder 5 automatically configures VisiBroker after BAS 4.5 is configured. To view or edit this configuration, open the Enterprise Setup dialog (Tools | Enterprise Setup) and select the CORBA setup tab. Select VisiBroker from the list of object request brokers and click "Edit."

Default Project Properties

Default project settings are stored in a project called Default.jpr located in the /.jbuilder5 subdirectory of the home directory. This Default.jpr file is used as a template whenever a new project is created. JBuilder5 offers a number of properties to configure projects but for the purposes of this document just the properties that directly effect WebLogic integration will be edited. Open the Default Project Properties dialog box (Project | Default Project Properties)

There are known problems with the ejbc compiler when the project is created in a directory that contains spaces: default project properties allow a default directory structure to be defined that does not contain spaces. On the Paths tab of default project properties, specify a default working directory (e.g. C:\JB5Projects), output path (e.g. C:\JB5Projects\classes), backup path (e.g. C:\JB5Projects\bck), source path (e.g. C:\JB5Projects\src), and documentation path (e.g. C:\JB5Projects\docs).

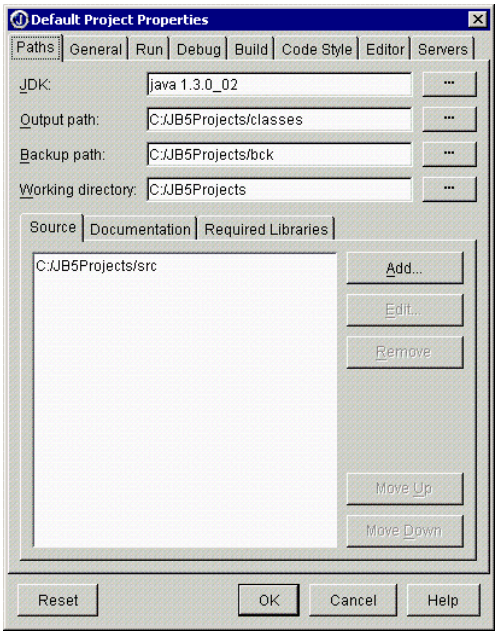


Figure 4: Path Configuration - Default Project Properties

Next, select the servers tab of the Default Project properties dialog. Select WebLogic 6.0 as the default application server for EJB development. The servers tab also allows the Web server to be selected. JBuilder provides Web server plug-ins for WebLogic 6.0, WebLogic 5.1, Tomcat 3.1, Tomcat 3.2, and Tomcat 4.0. If a choice in the drop-down list is in red, it is available but needs to be added to JBuilder's classpath before it can be used. Select the "Application server is web server" check box to use WebLogic 6.0. Enter the directory that is the root of the Web application in the default root field or leave it blank, and it will default to the root directory of the current project.

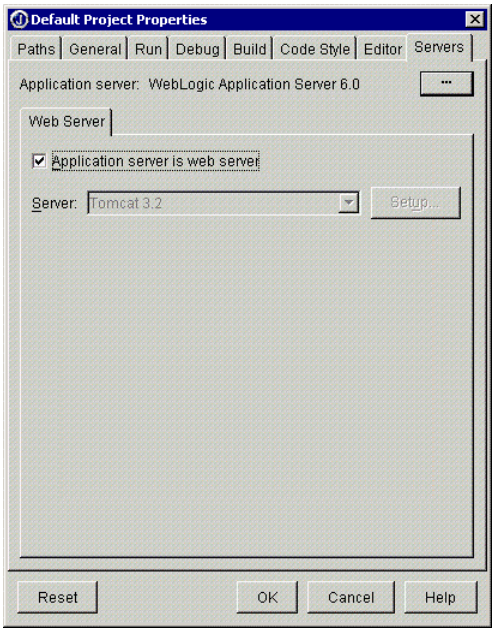


Figure 5: Application Server Selection - Default Project Properties

While in the Select Application Server dialog, the "Edit" button can be used to open the Application Server Properties dialog. This dialog allows parameters such as classpath and VM parameters to be configured. A common use of this dialog is installing new WebLogic service packs into the WebLogic classpath.

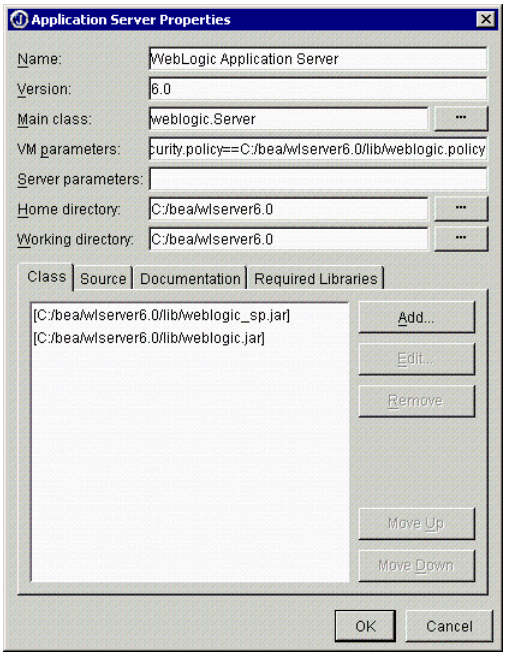


Figure 6: WebLogic Server Configuration - Default Project Properties

The EJB run properties allow the VM and application parameters to be configured for WebLogic when running it internally to JBuilder. It also provides a way to specify which EJBs get deployed. Later, we will discuss this topic in greater detail.

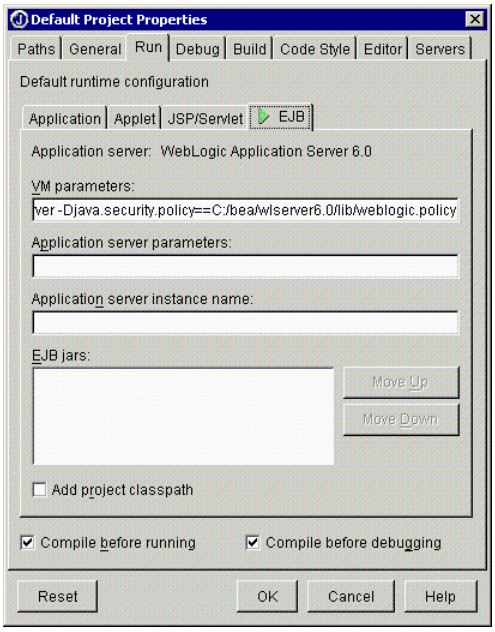


Figure 7: EJB Run Parameters - Default Project Properties

Creating the number guess application

The following sections provide a step–by-step guide to:

- Creating a stateful session bean
- Running the EJB in WebLogic Server internally to JBuilder
- Creating an EJB test client and a JSP client
- Deploying the EJB into a remote WebLogic Server
- Debugging the EJB in the remote WebLogic Server
- Deploying an EAR to the remote WebLogic Server

Technical tips and product tips are included in the relevant places. These are not required to complete the examples in this paper, but they do provide some useful information.

Create a new project

Open the new project wizard (File | New Project). Creating the new project is a three-step process:

Step 1 of 3: Enter the project name as NumberGuess, and leave the remaining values as the default values set in the *Default Project Properties* section of this paper.

Step 2 of 3: Leave default values for paths. Make sure Java 1.3 is selected as the JDK and WebLogic 6.0 is the only library under Required libraries.

Step 3 of 3: Enter Number Guess under the title and your name and company under Author/Company.

Create the Enterprise JavaBean

Open the Object Gallery (File | New) and select Enterprise JavaBean from the Enterprise tab. This also is a three-step process:

Step 1 of 3: Select the "New" button to create a new EJB group. Enter the name of the EJB group as NumberGDemo, and leave the remaining values as the defaults. Select "OK," then "Next."

Step 2 of 3: Enter the class name as NumberGDemoBean, and check the radio button next to the stateful session bean. Select "Next."

Step 3 of 3: Leave the default values and select "Finish."

Product Tips

An EJB group is a logical grouping of one or more beans that will be deployed in a single JAR file. There are two formats for EJB groups, binary (ejbgrp) and XML (ejbgrpx). If working with a version control system, the .ejbgrpx file extension may be more suitable, as it makes it easy to merge results.

Adding business logic to the EJB

The EJB wizard has created 3 Java files:

NumberGDemoBean.java is the actual EJB class;

NumberGDemo.java is the remote interface, and

NumberGDemoHome.java is the home interface.

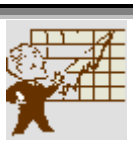
Add the following code to the EJB (NumberGDemoBean.java) :

```
import java.util.*;

private int theValue;
private int numberGuesses;
private String theResult;
public String makeGuess(String guess) {
  numberGuesses++;
  int intGuess = Integer.parseInt(guess);
  if (intGuess == theValue) {
    theResult = "Well Done! You guessed it in " + numberGuesses  +
    ". Number reset, why not try better next time!!";
    this.setValue();
  }
  else if (intGuess < theValue) {
    theResult = "Try higher";
  }
  else if (intGuess > theValue) {
    theResult = "Try lower";
  }
  return theResult;
}

public void setValue() {
  numberGuesses = 0;
  theValue = Math.abs(new Random().nextInt() % 100) + 1;
}
```

Technical Tips



The enterprise bean provider defines the remote and home interfaces and implements the enterprise bean class itself. The remote interface provides the calling interface to the client for the business logic methods implemented by the enterprise bean. The home interface provides methods to locate and create instances of the remote interface.

Update the remote interface

While NumberGDemoBean.java is the active file, select the bean tab and then the methods tab. The method tab displays the public methods defined in an EJB and automates exposing them through the remote interface. Select the check boxes next to makeGuess and setValue.

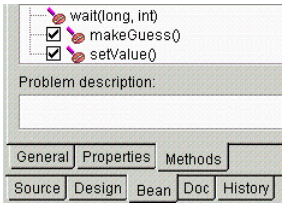


Figure 8: Expose Methods Through Remote Interface

Technical Tips

The remote interface defines the application-specific operations that a client may invoke. These are the public business methods which can be invoked by clients and which are actually implemented in the enterprise bean class. Note that clients of an enterprise bean do not access the bean directly; instead, they access its methods through its remote interface.

Compiling the EJB

Make the project (Project| Make Project "NumberGuess.jpx"). This will make the EJB, invoke the WebLogic EJBC compiler to create the stub files, and package it all into a JAR file called NumberGDemo.jar.

Product Tips

The EJBC compilation process is slow, so it may be a good idea to disable the EJBC compiler until it is needed. This can be done by right-clicking the EJB group in the project pane and selecting "properties." Inside the build tab there is a WebLogic 6.0 tab. Switch off "Use EJBC to generate stub files" by removing the check from the check box.

The run tab of the default project properties (Project | Default Project Properties) contains two check boxes called "Compile before running" and "Compile before debugging." As projects get larger and compilations take longer it is advisable to switch off these options.

Looking at the deployment descriptors

The deployment descriptors (DD) are maintained through the EJB group. Double-clicking the EJB group will activate the DD editor. The DD editor is a GUI that sits on top of the DD XML files. It creates both the standard XML files and the proprietary XML files required by each of the supported application servers.

The XML source files can be viewed directly by selecting the EJB DD source tab at the base of the DD editor.

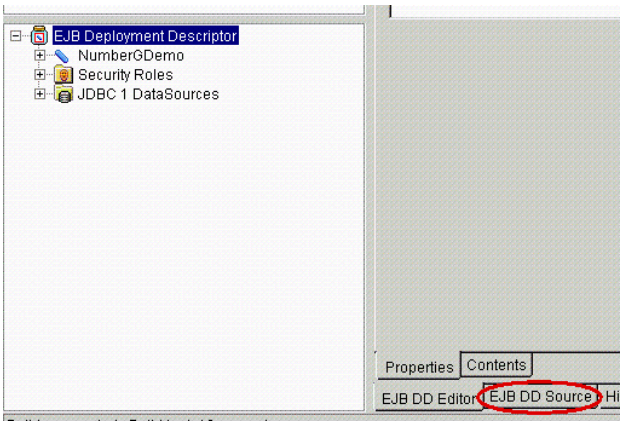


Figure 9: EJB Deployment Descriptor Source Tab

Three XML files now have been created: ejb-jar.xml is the standard DD; ejb-inprise.xml is the Borland-specific DD, and WebLogic-ejb-jar.xml is the WebLogic Server-specific DD. These XML files can be edited directly from this view; to avoid syntax errors, however, it is recommended that the EJB DD editor tab be selected and that changes be made through the GUI. For the purposes of this example, no changes are required in the DD.

Technical Tips

The role of the deployment descriptor is to provide information about each EJB that is to be bundled and deployed in a particular EJB JAR file. The information in the deployment descriptor is used in setting EJB attributes. These attributes define how the EJB operates within a particular environment. For example, when the EJB's transactional attributes are set, they define how the EJB operates when involved with a transaction.

Product Tips

To view WebLogic Server-specific properties, double-click the EJB group node in the project pane. Click the relevant enterprise bean in the structure pane, and select the EJB properties tab.

If an external editor is used to modify DD files, then these can be imported back into JBuilder by right-clicking the EJB group node, selecting properties, and using the "Add," "Copy," and "Delete" buttons on the EJB tab inside the build tab.

Running the EJB inside JBuilder

Right-click the EJB group in the project pane and select "Run." WebLogic Server starts inside the JBuilder environment, and standard out is redirected to a message pane. The EJB is deployed during startup. Start the WebLogic Server console and log into this instance of WebLogic Server to verify that NumberGDemo.jar has been deployed. The EJB is now ready to start taking invocations.

Product Tips

When running WebLogic Server internal to JBuilder, WebLogic Server can be configured as required from the Run/EJB tab of the project properties. For example, specifying -DWebLogic.management.password in the VM parameters allows WebLogic Server to start without prompting for a password, and WebLogic Server can then be closed down cleanly.

Creating an EJB test client

Open the Object Gallery (File | New) and select "EJB Test Client" from the Enterprise tab. Leave all the values as the defaults, and JBuilder will create NumberGDemoTestClient1.java, which contains EJB access code for NumberGDemoBean.

Edit NumberGDemoTestClient1.java to set the variables "user" and "password" equal to the required values; this will allow you to gain access to WebLogic Server. Add the following line of code to the main of NumberGDemoTestClient1.java:

```
    client.create();
    client.makeGuess("50");
```

Right-click NumberGDemoTestClient.java, and then make it run to test the business methods of the EJB.

Product Tips

The EJB test client is designed to allow bean developers to quickly test the EJB that has been created. It is not recommended that this code be used for the final production application.

The main purpose for the EJB test client is to abide the EJB philosophy of allowing the EJB developer to concentrate on writing the business logic and not get mired in other areas, such as updating clients.

Creating a JSP client

Open the Object Gallery (File | New) and select "Web Application" from the Web tab. Type "NumberGClient" as both the name of the Web application and the directory name. Select the "generate WAR" check box and click "OK." A new directory called NumberGClient will be created under the project directory as the default root for this Web application. It is important that the default root is not the same as the project directory.

Open the Object Gallery (File | New) and select "Java Server Page" from the Web tab. Creating the new JSP is a three stage process:

Step 1 of 3: Ensure NumberGClient is selected as the WebApp, leave the remaining values as defaults.

Step 2 of 3: Leave the default values

Step 3 of 3: Select "Finish."

A default JSP file called Jsp1.jsp will be created. If using Borland Enterprise Studio for Java, Macromedia® Dreamweaver® UltraDev™ can be used to modify the .jsp file. Alternatively, replace all the code in Jsp1.jsp with the code below.

```
<html>
<head>
<jsp:useBean id="Jsp1BeanId" scope="session" class="numberguess.Jsp1Bean"
/>
<jsp:setProperty name="Jsp1BeanId" property="*" />
<title>
```

```
Jsp1
</title>
</head>
<body bgcolor="#ffc800">
<h1> Number Guess</h1>
<form method="post">
  <p><br>
    <font size="5" color="#990000">My Guess Is:
    <input name="sample">
    </font></p>
  <p><font size="5" color="#990000">
    <input type="submit" name="Submit" value="Submit Guess">
    <br>
    </font></p>
  <p><font size="5" color="#990000"><font color="#000066">
    <jsp:getProperty name="Jsp1BeanId" property="sample" />
    </font> </font> </p>
</form>
</body>
</html>
```

Edit Jsp1Bean.java as shown below

```
package numberguess;

public class Jsp1Bean {
  private String sample = "Guess a number between 1 and 100";
  NumberGDemoTestClient1 numberGDemoTestClient1
    = new NumberGDemoTestClient1();

  public Jsp1Bean() {
    numberGDemoTestClient1.create();
    numberGDemoTestClient1.setValue();
  }

  /**Access sample property*/
  public String getSample() {
    return sample;
  }
  /**Access sample property*/
  public void setSample(String newValue) {
    if (newValue!=null) {
      sample = numberGDemoTestClient1.makeGuess(newValue);
    }
  }
}
```

JBuilder 5 can check JSPs for errors at build time. Disable this functionality on the Build/JSP tab of Project Properties (Project | Project Properties) by unselecting the "Check JSPs for errors at build-time" checkbox. Alternatively, make sure that WebLogic Server is running and allow JBuilder to check the JSP. Make the project (Project| Make Project "NumberGuess.jpx"). Select Jsp1.jsp from inside the WebApp (NumberGClient/Root Directory/Jsp1.jsp), right-click, and select "Web Run." WebLogic Server will start inside the JBuilder environment and run the JSP.

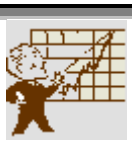
Product Tips

A WebApp describes the structure for a Web application. It is essentially a directory tree containing Web content used in your application. A deployment descriptor file called `web.xml` is always associated with each WebApp. This deployment descriptor contains the information you need to provide to your Web server when you deploy your application.

The JBuilder environment can become crowded when multiple applications are being run, debugged, and edited. This problem can easily be resolved by starting a clean copy of the JBuilder AppBrowser™ (Window | New Browser).

Running the JSP from inside JBuilder starts WebLogic Server on port 8080, so the internal WebLogic Server should not conflict with any existing running instances of WebLogic Server. JBuilder creates a backup of the WebLogic Server configuration file called config.xml.before.JBuilder.WebRun.

Technical Tips



A WAR file is an archive file for a Web application. It's similar to a JAR file. By storing your entire application and the resources it needs within the WAR file, deployment becomes easier.

Deploying the EJB to an external WebLogic Server

Ensure the internal version of WebLogic Server is not running and start WebLogic Server externally. In the project pane, open the NumberGDemo.ejbgrp, right-click the JAR file inside, and select "Deploy Options for NumberGDemo.jar / Deploy." Enter the password and unit name for the deployment. JBuilder calls the WebLogic Server API to deploy the EJB JAR file. Note

that the same method is used to re-deploy, un-deploy, and list deployments. If the EJB already exists in WebLogic Server, it may be necessary either to use re-deploy or to remove it from WebLogic Server before deployment. Test the EJB with the EJB Test Client and/or the JSP Client.

Product Tips

If there is a requirement for editing XML files, JBuilder provides extensive capabilities. Adding XML configuration files to a JBuilder project can save time and reduce the risk of syntax errors. JBuilder also provides a history of changes, which can prove very useful if errors are made while editing configuration files.

The command-line deployment used to deploy from JBuilder to WebLogic Server has several useful options such as –host, -port, and -username. Details can be found in the *WebLogic Server Admin Guide*.

Debugging an EJB running in WebLogic Server

Make a copy of the WebLogic Server startup file and edit the start command to include the following VM parameters:

```
-classic -Xdebug -Xnoagent -Djava.compiler=NONE -
Xrunjdwp:transport=dt_socket,address=8888,suspend=n,server=y
```

Note that –classic must be the first parameter passed to the VM. Start WebLogic Server using this new startup file, and WebLogic Server will start in debug mode. It will be listening on port 8080 for a remote process such as JBuilder to attach and start debugging it.

Open the Run Configurations dialog (Run | Configurations) and click the "New" button. Enter a configuration name of "Remote Debug WebLogic Server." Select the debug tab, select the "Enable remote debugging" checkbox, select "Attach radio button," enter the name of the host, and set the address to 8888.

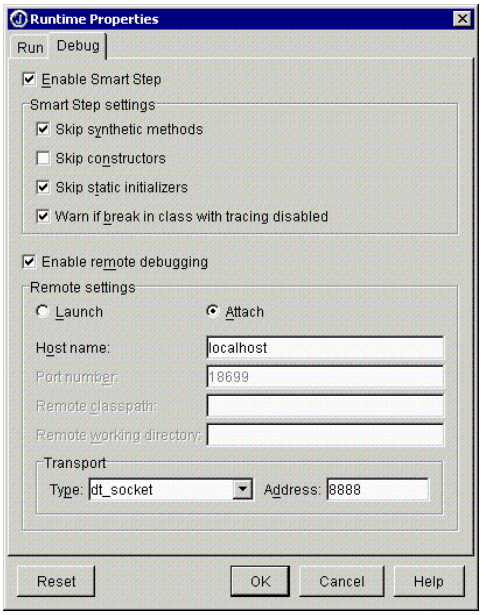


Figure 10: Run Configurations, Enable Remote Debugging

To start this configuration, select "Remote Debug WebLogic Server" from the drop-down list on the main tool bar on the right-hand side of the "Debug project" button.

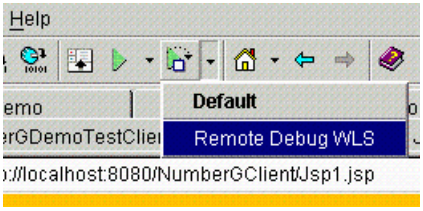


Figure 11: Running "Remote Debug WebLogic Server"

A message pane will appear. It initially should be blank. If there are any connection errors, they will appear in this window. Add breakpoints to the EJB by clicking in the left margin of the source code. Invoke the EJB from the client and execution stops at the breakpoint.

Product Tips

JBuilder supports the Java Platform Debugging Architecture (JPDA), which is used to remote debug any Java program running in any VM that supports JPDA. For example, JBuilder can also debug servlets and JSPs in a remote WebLogic Server. In the case

of JSPs, the generated source must be added to the JBuilder project. For more details of the available options, see: http://java.sun.com/products/jpda/

An alternative way to debug an EJB inside JBuilder is to select the EJB group in the project pane, right-click, and select "debug." WebLogic Server will start internally to JBuilder, and breakpoints can be added to the EJB source code.

Technical Tips

A very common error when starting a VM in debug mode is "Can't load jdwp.dll, because can't find dependent libraries." Most often, this error occurs when JPDA has not been installed. JPDA can be downloaded from http://java.sun.com/products/jpda/

Deploying an enterprise archive file

Open the Object Gallery (File | New) and select "EAR" from the enterprise tab. Creating an EAR file is a 5 step process:

Step 1 of 5: Leave the default name as NumberGuess and select "Next."

Step 2 of 5: Select the EJB group NumberGDemo.ejbgrp and select "Next."

Step 3 of 5: The project does not contain any RAR files, so select "Next."

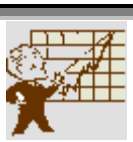
Step 4 of 5: Java client modules are not required, so select "Next."

Step 5 of 5: Select the WebApp node NumberGClient and select "Finish."

Ensure that the external WebLogic Server is running. Make the project to create the EAR file, and select NumberGuess.ear from inside NumberGuess.eargrp in the project pane. Right-click the EAR file inside and select "Deploy Options for NumberGuess.ear / Deploy." If requested, enter the password and unit name for the deployment. JBuilder calls the WebLogic Server API to deploy the EAR file. Assuming WebLogic Server is configured as default, run the application from a browser using the following URL:

http://localhost:7001/NumberGClient/Jsp1.jsp

Technical Tips



EAR files contain other archive files such as WARs, JARs, and RARs that together make up an enterprise application.

Congratulations!

You have just created a simple application using JBuilder and WebLogic Server. We hope this paper helps developers understand the way Borland JBuilder 5 Enterprise continues to improve developer productivity.

JBuilder 5 Enterprise has many more features that could not be covered in such a short paper. Please refer to the *Additional Information* section, the *JBuilder Help System*, or contact your local Borland representative for details about Borland training and consulting services.

Additional information

Borland JBuilder

http://www.borland.com/jbuilder/

Borland Enterprise Studio for Java

http://www.borland.com/estudio/

Sun Microsystems Java 2 Platform Enterprise Edition

http://java.sun.com/j2ee/

Sun Microsystems Enterprise JavaBeans

http://java.sun.com/products/ejb/

BEA WebLogic Server

http://www.beasys.com

Clustering techniques

http://www.borland.com/appserver/papers/clustering.html

JBuilder Open Tools

http://www.borland.com/jbuilder/resources/jbopentools.html

Borland AppCenter™

http://www.borland.com/appcenter/

Borland AppServer

http://www.borland.com/appserver/

Borland newsgroups

http://www.borland.com/newsgroups/

Definitions

Acronym	Description
CORBA	Common Object Request Broker Architecture
EJB	Enterprise Java Beans
RMI	Remote Method Invocation
JPDA	Java Platform Debugger Architecture
IIOP	Internet Inter-ORB Protocol
BAS	Borland AppServer
DD	Deployment Descriptor
EAR	Enterprise Archive
WAR	Web Archive
JAR	Java Archive
VM	Virtual Machine
JSP	Java Server Page
XML	Extensible Markup Language

JBuilder AppBrowser

JBuilder uses one window to perform most development functions: editing, visual designing, navigating, browsing, and debugging. This window is called the AppBrowser. The terminology used when referring to different sections of the AppBrowser is detailed in *Figure 12*.

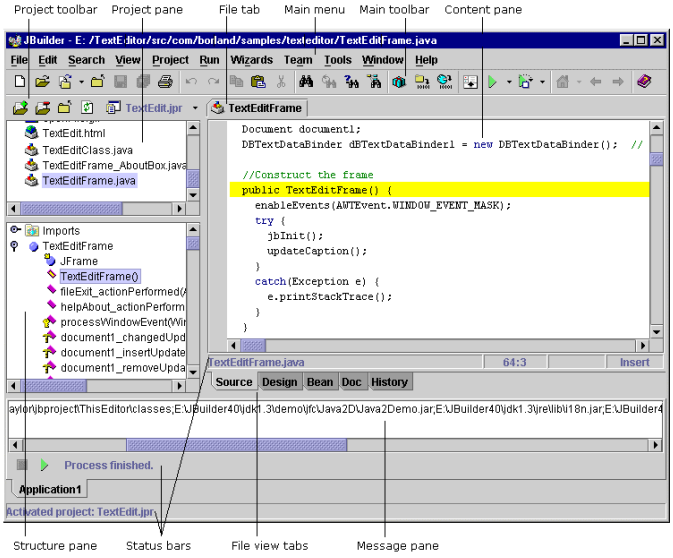


Figure 12: The JBuilder AppBrowser

Borland

100 Enterprise Way
Scotts Valley, CA 95066-3249
www.borland.com | 831-431-1000

Made in Borland® Copyright © 2001 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. CORBA and ORB are registered trademarks of Object Management Group, Inc. in the U.S. and other countries. All other marks are the property of their respective owners. 12289