

Using Borland® JBuilder® and Borland® InterBase® in the development of J2EE™ applications

A Borland White Paper

By Shaunak Mistry, Staff Development Engineer

February 2004

Borland®

Introduction.....	3
The Borland® InterBase® database.....	3
InterBase® and JBuilder®	4
InterBase® and Borland® Enterprise Server	4
Why InterBase®?.....	5
Using InterBase® as a back-end database	5
Installing InterBase® 7.1 and InterClient® 4.0.....	6
Platforms and system requirements	6
Configuring JBuilder® to use InterBase®.....	7
Adding InterClient® to the JBuilder® classpath	7
Adding the InterClient® JAR file to your JBuilder® projects.....	8
Adding and configuring the Borland® Enterprise Server	8
Creating and testing Enterprise JavaBeans™	9
Overview.....	9
Creating a JBuilder® project.....	9
Configuring Borland® Enterprise Server as the project's J2EE™ server.....	10
Creating the EJB module	11
Creating the session bean.....	12
Creating the entity bean	14
Connecting the CustomerSession bean to the Customer entity bean	17
Delploying the EJB	19
Conclusion.....	27
Additional information	27
Feedback and suggestions	28

Introduction

Many of the most powerful applications delivering competitive advantage for global corporations are J2EE™-based. Their success has also driven J2EE solutions into the realm of the small and medium enterprise.

Because the purpose of these applications is to drive business growth, power and scalability are design requirements. Developers building these applications must choose carefully to ensure that the infrastructure they select can support future demands while at the same time delivering immediate productivity and economy.

The typical elements of the developer's J2EE infrastructure:

- Development environment
- Application server
- Database

This white paper describes how to use the Borland® InterBase® database in a J2EE application architecture that includes an application built using Borland® JBuilder® and the Borland® Enterprise Server application server.

The Borland® InterBase® database

Borland® InterBase® is a powerful, versatile database that is well-suited for supporting J2EE applications. It provides all of the features developers expect of a database that can support complex business logic and hundreds or even thousands of concurrent users (and here we mean truly concurrent users). Yet the design of InterBase is such that it possesses the small footprint and nearly zero maintenance required of a database that is to be “embedded” in an application.

InterBase ships with InterClient® 4.0, a highly performant, Type 4, JDBC® 2 driver. It is this driver that allows developers to use InterBase as the database serving J2EE applications. It

also allows developers to use the Borland JBuilder integrated development environment and the technically advanced and economical application server, Borland® Enterprise Server.

InterBase® and JBuilder®

The InterBase and JBuilder combination is a powerful one. JBuilder is an extremely efficient development technology. Developers using JBuilder find their productivity enhanced. But when JBuilder is coupled with the Borland® Together® ControlCenter® modeling technology and the Borland® Optimizeit™ performance optimization technology, developer productivity increases dramatically. A developer who can leverage the productivity benefits of JBuilder in development and the virtually zero maintenance requirement of InterBase in production contributes substantially to achieving the business goals that rise above simple development schedules.

InterBase® and Borland® Enterprise Server

The InterBase and Borland Enterprise Server (BES) combination is also worth discussion. IT managers (and developers themselves) are always looking for ways to speed development and get applications into production faster. Yet the design/development/ test phase of the application lifecycle is relatively short compared to the time that an application lives in production.

Studies have shown that the design/development/test phase lasts just 6 to 9 months for a typical ERP project. The application, however, will live in production for an average of 8 years. Perhaps more important, development costs typically represent just 40% of the total cost of an application, while production costs—database and application server acquisition, deployment, and maintenance—represent 60%. Clearly, the selection of database and application server is key to the success of a development project. Together, Borland Enterprise Server and InterBase represent rock-solid deployment, economically defensible technologies that help guarantee the success of application development projects.

Why InterBase®?

InterBase is a powerful, versatile database remarkable for its low acquisition cost and extremely low maintenance costs. This is not surprising: InterBase was specifically designed to require nearly zero maintenance in production. It features automatic crash recovery and self-tuning facilities that virtually eliminate the need for the attention of a database administrator or other IT support in the field. In addition, InterBase has a small footprint that not only makes it easy to deploy but requires only modest system resources. For these reasons, InterBase is often chosen to support applications that must be deployed in many locations where IT support is unavailable or too costly.

InterBase possesses another feature that makes it attractive for J2EE application development: conformance to standards. InterBase supports SQL92 without resorting to proprietary syntax or markers. The JDBC® driver supplied with InterBase, InterClient, is a fully compliant, Type 4, JDBC2 driver. This means developers can confidently use InterBase in development and deployment and, if necessary, easily swap in an Oracle or IBM® DB2® database should business requirements dictate. Many developers and IT managers are pleasantly surprised, however, to find that InterBase is more than adequate for their J2EE applications in the vast majority of cases.

Borland® Together® ControlCenter,™ JBuilder, Borland Enterprise Server, and InterBase represent an extremely efficient, economical solution for developers working to quickly deploy powerful J2EE applications without the burden of a costly support ecosystem.

Using InterBase® as a back-end database

The following is an overview of how InterBase can be used as the back-end database to develop J2EE-based applications using the JBuilder development environment, Borland Enterprise Server, and InterBase 7.1 with InterClient 4.0. The paper assumes the use of JBuilder 9 but applies to JBuilder 8 as well.

Software used for this paper is Borland Enterprise Server 5.2 , Borland JBuilder 9, Borland InterBase 7.1, and Borland InterClient 4.0 on a Windows® 2000 operating system.

The http://www.borland.com/interbase/pdf/ib71_techview.pdf link provides a technical overview of InterBase 7.1.

You can obtain a trial version of InterBase 7.1 that allows use of four CPUs and 20 concurrent users for 90 days at the following link:

http://www.borland.com/products/downloads/download_interbase.html#

Installing InterBase® 7.1 and InterClient® 4.0

Choose **Install InterBase 7.1** from the launcher screen, type **F:\Program Files\Borland\InterBase** for the install directory.

After you have completed installing InterBase, choose **Connectivity Drivers** from the launcher, pick **JDBC driver** from the next window. Choose the same directory **F:\Program Files\Borland\InterBase** for the install directory. InterClient.jar will be installed in the **F:\Program Files\Borland\InterBase\lib** directory. You will need the location of this JAR file so that you can configure JBuilder to use it.

For the purpose of this white paper, we will assume that InterBase has been installed to the **F:\Program Files\Borland\InterBase** directory.

Platforms and system requirements

InterBase 7.1 and InterClient 4.0 are certified on the Windows, Solaris,™ and Linux® platforms.

InterBase requires 32 MB RAM and 20 MB of hard disk for installation.

InterClient requires JDK® 1.2 or higher, and about 240 KB for the JAR file.

Click on http://www.borland.com/jbuilder/pdf/jb9_sysreqs.pdf for JBuilder 9 Enterprise system requirements.

Click on <http://info.borland.com/techpubs/bes/platforms/5xaseindex.html> for Borland® Enterprise Server, App Server™ Edition system requirements. Borland Enterprise Server requirements are needed for the final deployment of the application. Because this paper focuses on the development of the application and how it can be run within JBuilder, without the need to start anything other than the Borland JBuilder, JBuilder Enterprise system requirements will suffice.

Configuring JBuilder® to use InterBase®

Adding InterClient® to the JBuilder® classpath

Start JBuilder and follow the steps below to set up InterClient (the InterBase JDBC driver)

1. Open JBuilder and choose **Tools|Enterprise Setup**.
2. Click the **Database Drivers** tab in the Enterprise Setup dialog box. The Database Drivers tab displays .config files for all currently defined database drivers. If InterClient.config (or interclient.config) already exists and JBuilder is already set up to work with InterBase, cancel out of the Enterprise setup; otherwise, continue with the steps below. Click **Add** to add a new driver, then **New** to create a new library file for the driver. The library file is used to add the driver to the required libraries list of projects.
3. Type the name **InterClient** and select a location for the new file in the Create New Library dialog box. Click **Add**, and browse to the location of the driver. You can select the directory containing the InterClient.jar file **F:\Program Files\Borland\InterBase\lib**.
4. Click **OK** to close the file browser. This displays the new library at the bottom of the library list and selects it.

5. Click **OK**. JBuilder creates a new .library file in the JBuilder /lib directory with the name you specified (for example, **InterClient.library**). It also returns you to the Database Drivers page, which displays the name of the corresponding .config file in the list, which will be derived from the library file (for example, **InterClient.config**).
6. Select the new .config file in the database driver list and click **OK**. This places the .config file in the JBuilder /lib/ext directory.
7. Close and restart JBuilder so that the changes to the database drivers will take effect and the new driver will be put on the JBuilder classpath.

Adding the InterClient® JAR file to your JBuilder® projects

1. Start JBuilder and close any open projects.
2. Choose **Project|Default Project Properties**.
3. Select the **Required Libraries** tab on the Paths page, and click **Add**.
4. Select the **InterClient** library list, and click **OK**.
5. Click **OK** to close the Default Project Properties dialog box.

Adding and configuring the Borland® Enterprise Server

Refer to the white paper referenced by this link

http://www.borland.com/products/white_papers/pdf/jbuilder_configuration_with_j2ee_appservers.pdf and review the "Install the Borland Enterprise Server" section. In particular, make a note of the AppServer user port number port number. For the purpose of this paper, this port number is assumed to be 16530. The install directory for the Borland Enterprise Server is assumed to be "E:\BorlandEnterpriseServer".

Creating and testing Enterprise JavaBeans™

Overview

We are going to develop and test two Enterprise JavaBeans™ —a session bean and an entity bean. The entity bean **Customer** makes the connection with the CUSTOMER table in the InterBase database employee.gdb to retrieve information. The session bean, **CustomerSession**, will reference the entity bean and use it to get the customer name for a given customer number.

Creating a JBuilder® project

From the JBuilder main menu bar, click **File|New|Project** tab from **JBuilder**. In the **Project Wizard**, enter the **Name** as **IBDemo** and the **Directory** as **C:\Demo\IBDemo**. Click the **Finish** button. JBuilder will create a new project called **IBDemo**.

Now check to make sure that the project created contains the right JAR files. Click **Project|Project Properties**, and select the **Paths** tab. Under this tabbed window, select the **Required Libraries** tab. If you have set up the InterClient and Borland Enterprise Server as instructed above, they should both appear in your **Required Libraries** tab. If you do not see them, then add them manually here. Figure 1 shows how the expected **Required Libraries** tab should look.

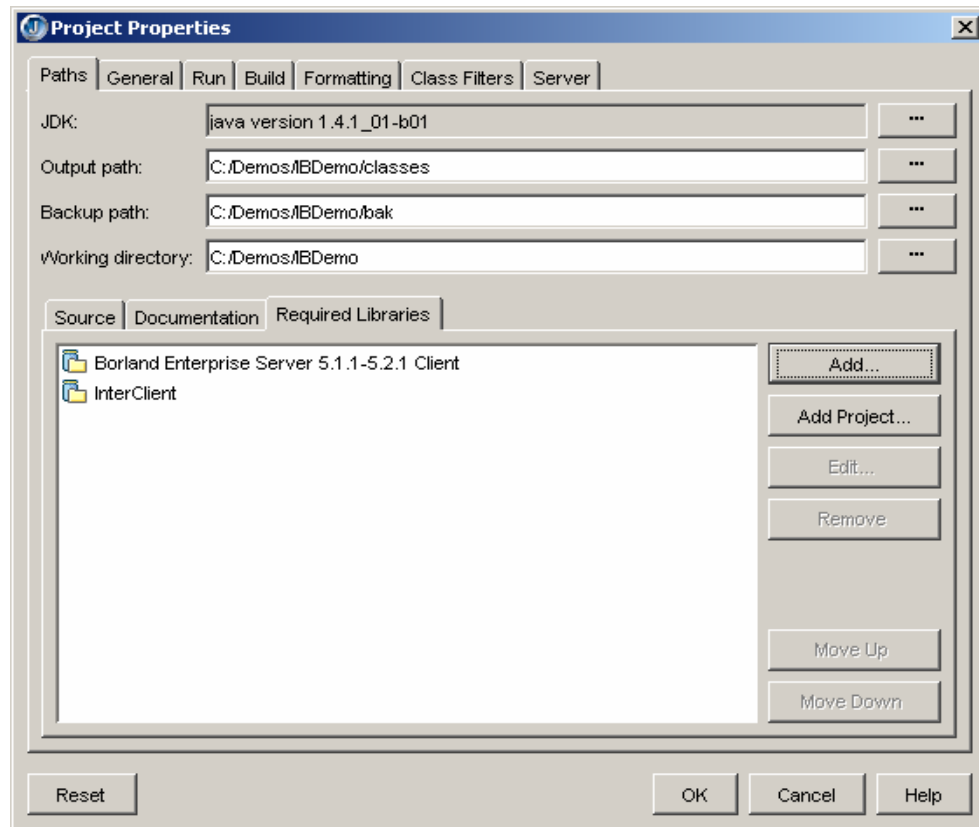


Figure 1: Required libraries set for our project

Configuring Borland® Enterprise Server as the project's J2EE™ server

From JBuilder main menu bar, click **Project|Project Properties** click the **Server** tab. Select **Borland Enterprise Server AppServer Edition 5.x** from the combo box. The radio button **Single server for all services in project** should already be the default selection. Click the ... next to this box. This will bring up the **Edit or Select Server** window. Change the partition used to **standard**. If this is not the case click the ... next to the **Server parameters** box. This will bring up the **Change Server Parameters** window. Make sure you have the standard partition entered here e.g. **-partition standard -server agni** (replace agni with your server name) Figure 3 show the expected input. Click **OK** on both the windows. Click **OK** on the **Project Properties** window.

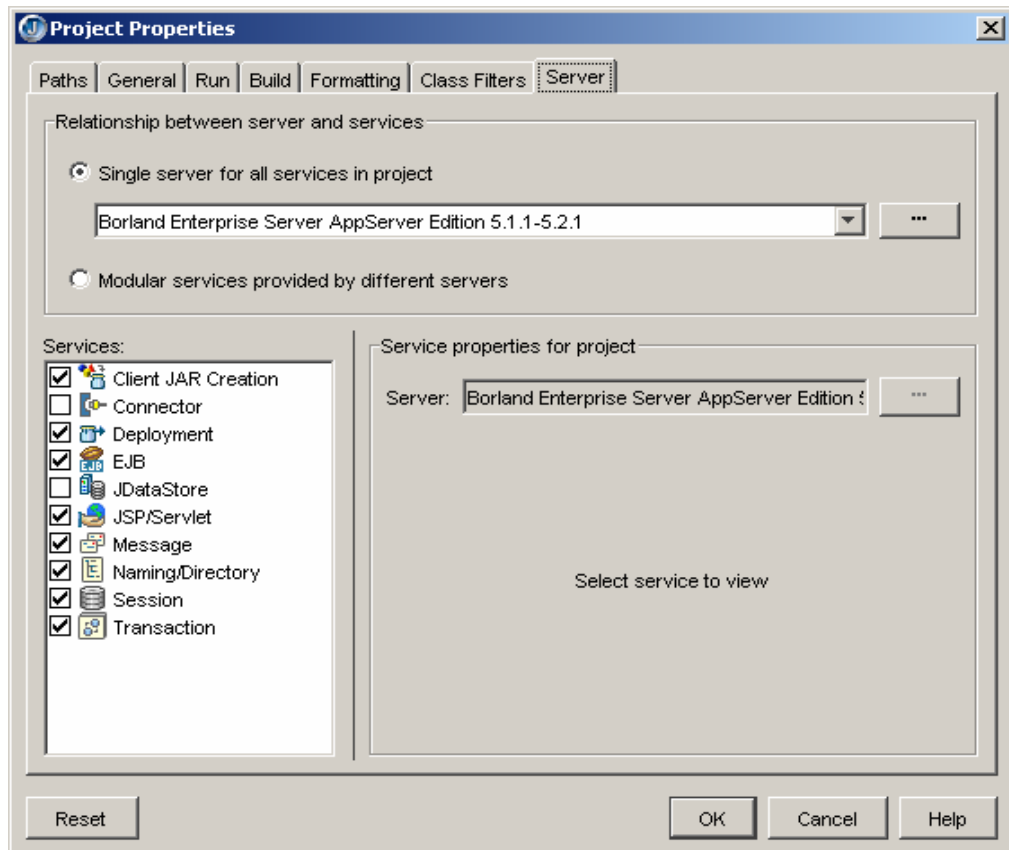


Figure 2: Configuring Borland Enterprise server as the J2EE server for our project

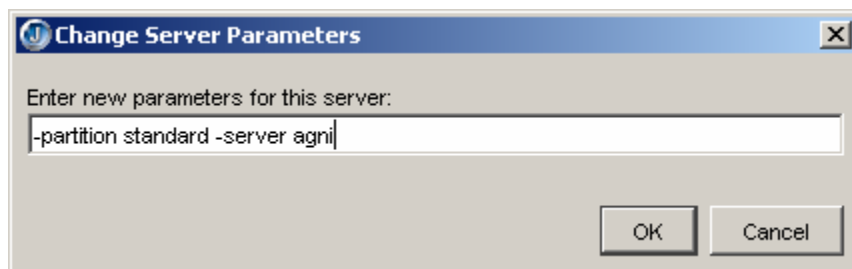


Figure 3: Ensure that we have changed the partition used by BES to standard

Creating the EJB module

An EJB module is a logical grouping of one or more Enterprise JavaBeans that will be deployed in a single JAR file. We will create an EJB module for our project. The EJB module can be edited using the **Deployment Description Editor**.

To get to the EJB module form, use **File|New**, use the **Enterprise** tab, and click the **EJB Module**.

Figure 4 shows how the completed EJB module form should look. Hit **OK** so that JBuilder can create the Module named **EJBModule** for you.

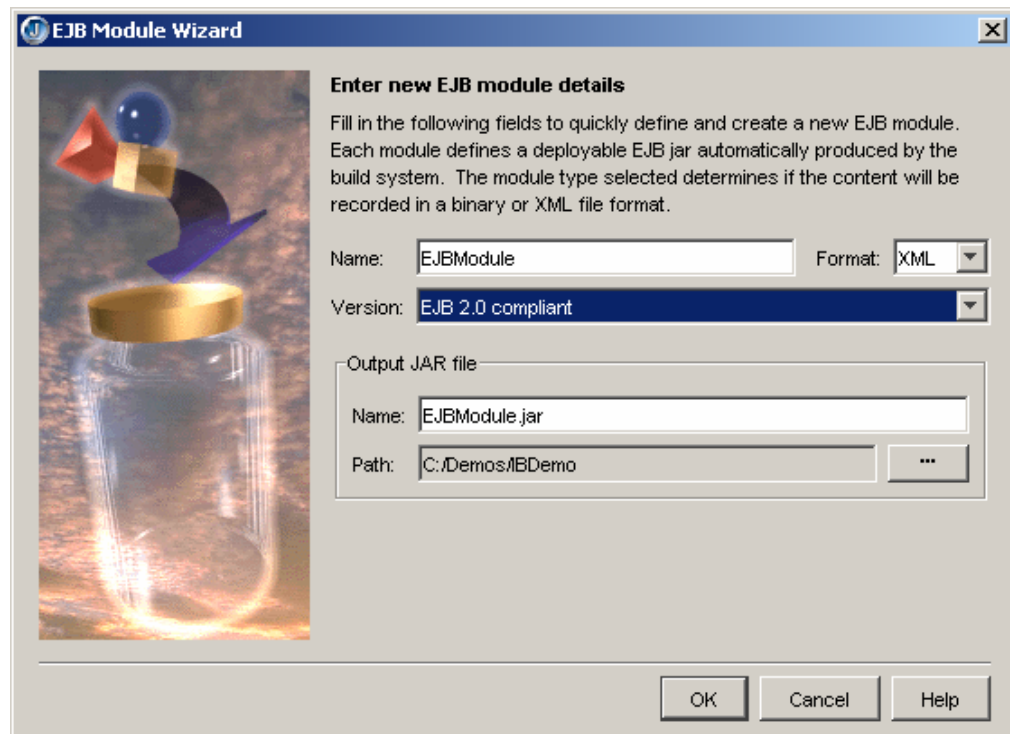


Figure 4: Filled-out EJB module form for our project

The **EJBModule** in the JBuilder window should appear under our project **IBDemo**.

Creating the session bean

Double-click the **EJBModule** and right-click the editor pane., on the resulting window select **Create EJB** and click **Session Bean**. Enter the Bean name as **CustomerSession** in the Bean.

Figure 5 shows the values needed to create this session bean. Click anywhere in the editor pane to close the Bean properties window. Select **Save All** to move on

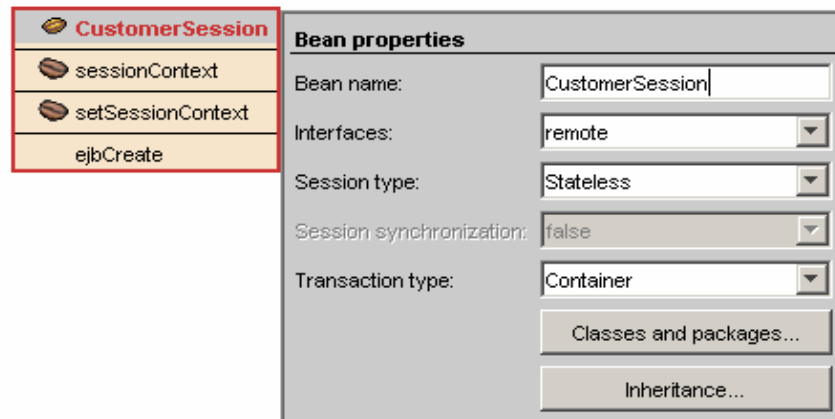


Figure 5: Completed session bean form

JBuilder has created three EJB files:

CustomerSessionBean.java – the bean class of the customer session bean.

CustomerSession.java – the remote interface of the customer session bean.

CustomerSessionHome.java– The home interface of the customer session bean.

Adding a method to the CustomerSession bean

To add a method to the session bean, right-click the **CustomerSession**. Choose **Add**, and then click **Method**. Enter the following values

Method name: getCustomer

Return type: String

Input parameters: Integer id (the input parameter is of type Integer and called id)

Interfaces: remote

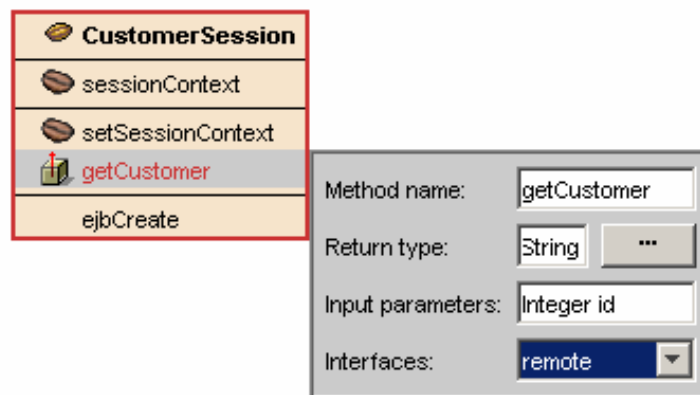


Figure 6: Adding the method *getCustomer* to *CustomerSession*

Creating the entity bean

This section explains how to use JBuilder to build entity beans using the features of the EJB designer. With the flexible IDE of JBuilder, you can create the entity beans in one of these ways:

- Import a database schema from an existing **datasource** and select desired tables to construct entity beans,
- Right-click the EJB designer and choose **CMP 2.0 Entity Bean**.

In this example, we will import a database schema into the project and create a CMP. We will use the **employee.gdb** database that is found in the InterBase example directory **F:\Program Files\Borland\InterBase\examples\databases**.

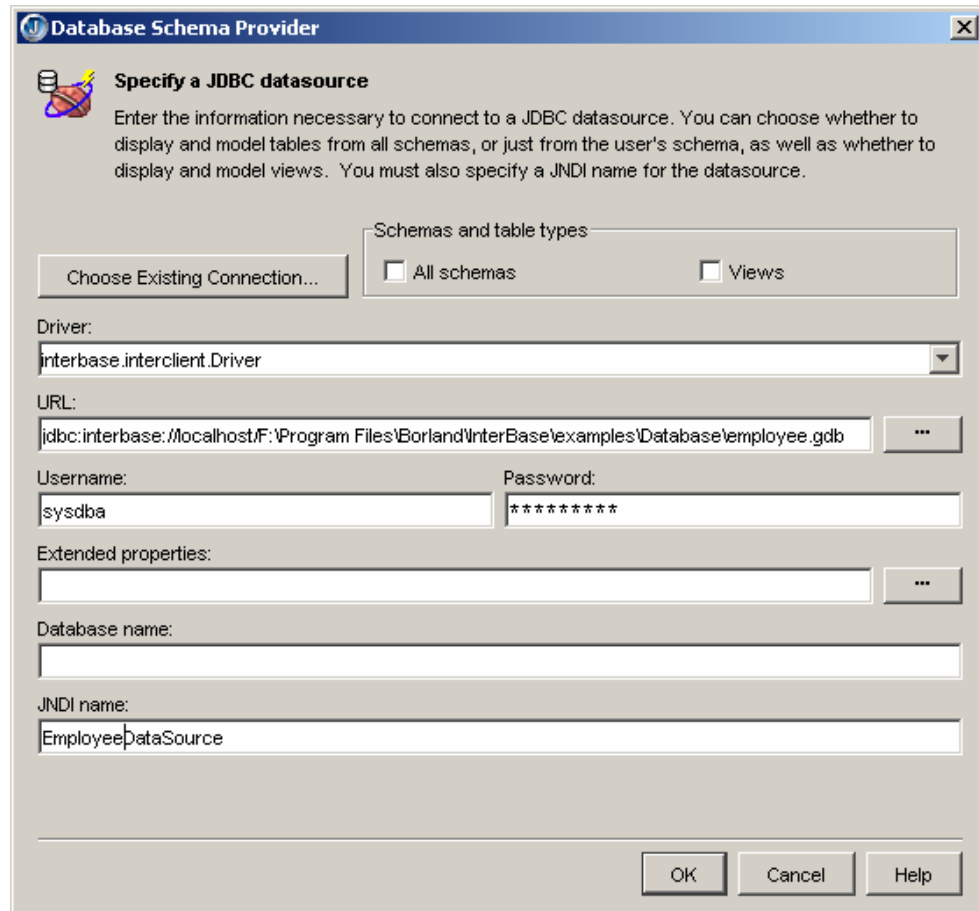


Figure 7: Setting up the entity bean to use the employee.gdb database as a datasource

To import a data source, right-click the DataSources node in the structure pane and choose **Import Schema From Database**. You can also right-click the EJB designer pane and choose **Import Schema From Database**. JBuilder will display the **Database Schema Provider** dialog box, as shown in the Figure 7 Database Schema provider box filled out for employee.gdb

In this sample, the default **sysdba** password **masterkey** has been used.

Hit **OK** to continue. You must **Save all** at this point.

Creating the entity bean

To create a CMP 2.0 entity bean for the CUSTOMER table, right-click **CUSTOMER** from the structure pane and select **Create CMP 2.0 Entity Bean**. JBuilder will create an entity bean Customer in the EJB designer pane.

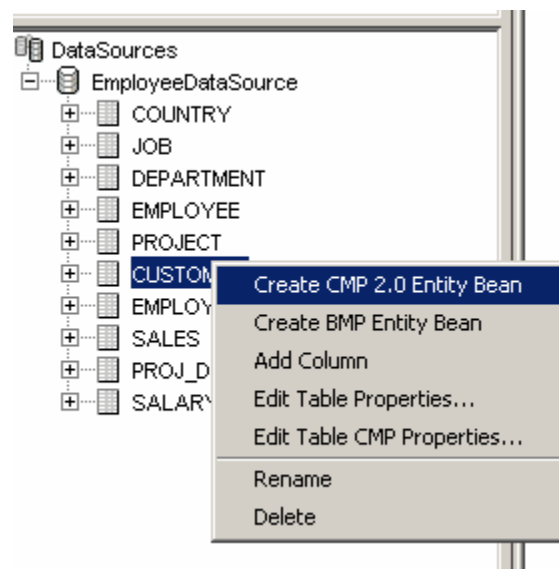


Figure 8: Select the Customer table to create a CMP entity bean

JBuilder will also create three EJB files which can be seen in the project tree:

CustomerBean.java – the bean class of the customer bean

Customer.java – the remote interface of the customer bean

CustomerHome.java – the home interface of the customer bean

Connecting the CustomerSession bean to the Customer entity bean

We will change the code in the CustomerSession bean for the previously added method `getCustomer()`. This method will access the entity bean Customer and obtain Customer name from the CUSTOMER table, given the customer number (CUST_NO).

Modifying the `getCustomer()` method.

Right-click the EJB **CustomerSession** in the designer pane and click **View Bean Source** in the pop-up. JBuilder will take you to the Editor window and open the **CustomerSessionBean.java** file. Replace the existing `getCustomer` method with the following code:

```
public String getCustomer(Integer ID) {
    String strCustomerName = null;
    try {
        javax.naming.Context ctx =
            new javax.naming.InitialContext();
        Object obj = ctx.lookup("java:comp/env/ejb/Customer");
        CustomerHome aCustomerHome =
            (CustomerHome)obj;
        Customer aCustomer = aCustomerHome.findByPrimaryKey(ID);
        strCustomerName = aCustomer.getCustomer();
        return strCustomerName;
    }
    catch (ClassCastException ex) {
        return "ClassCastException";
    }
}
```

```
        catch (NamingException ex) {  
            return "NamingException";  
        }  
        catch (FinderException ex) {  
            return "FinderException";  
        }  
    }  
}
```

While we are here, add the following import to the Java source file

```
import javax.naming.*;
```

just after the line

```
import javax.ejb.*;
```

Setting the session beans local references

We will be setting the **CustomerSession** bean's EJB local references to point to the **Customer** entity bean.

When you double-click the **CustomerSession** bean in the project pane (see figure 9), the deployment descriptor editor is shown by JBuilder. Select the **EJB Local Reference** tab, and click **Add** to define a new local reference to the Customer bean.

Once this is done, import javax.naming. **Save all** and then build the complete project by selecting **Project|Rebuild Project IBDemo.jbx**

JBuilder will compile and build the EJBModule.jar file

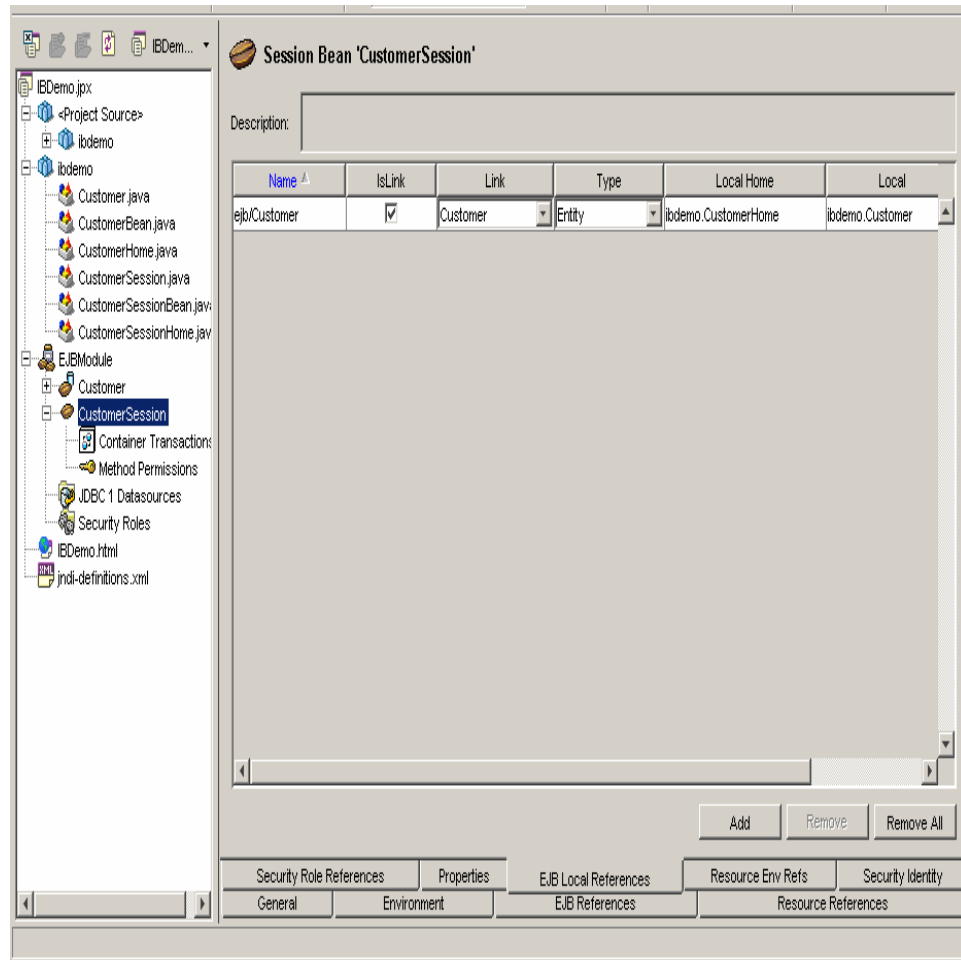


Figure 9: EJB local references filled out for the CustomerSession bean

Deploying the EJB

Creating runtime configurations for Borland Enterprise Server (BES)

From the JBuilder main menu bar, click **Run|Configurations**. The **Project Properties** window appears with the **Run** tab selected. Click **New** to create a new runtime configuration. Name it **BES Server**, as shown in Figure 9.

Should you have to change the **Name**, select <None> from the drop-down list for **Build Target**. Next, change the **Type** to **Server** from the drop down list. You should also uncheck the JSP/Servlet box, as our project does not use these Web-based services.

Make sure that the partition name here is set to **standard**. You might notice that the port and the server name are different than the one shown in Figure 10. This will not be a issue. We need to ensure that the partition name matches the your setup and the directory you copied the Interclient.jar file to uses this partition name. All of these should be **standard**.

Click **OK** twice to get back to the main JBuilder window.

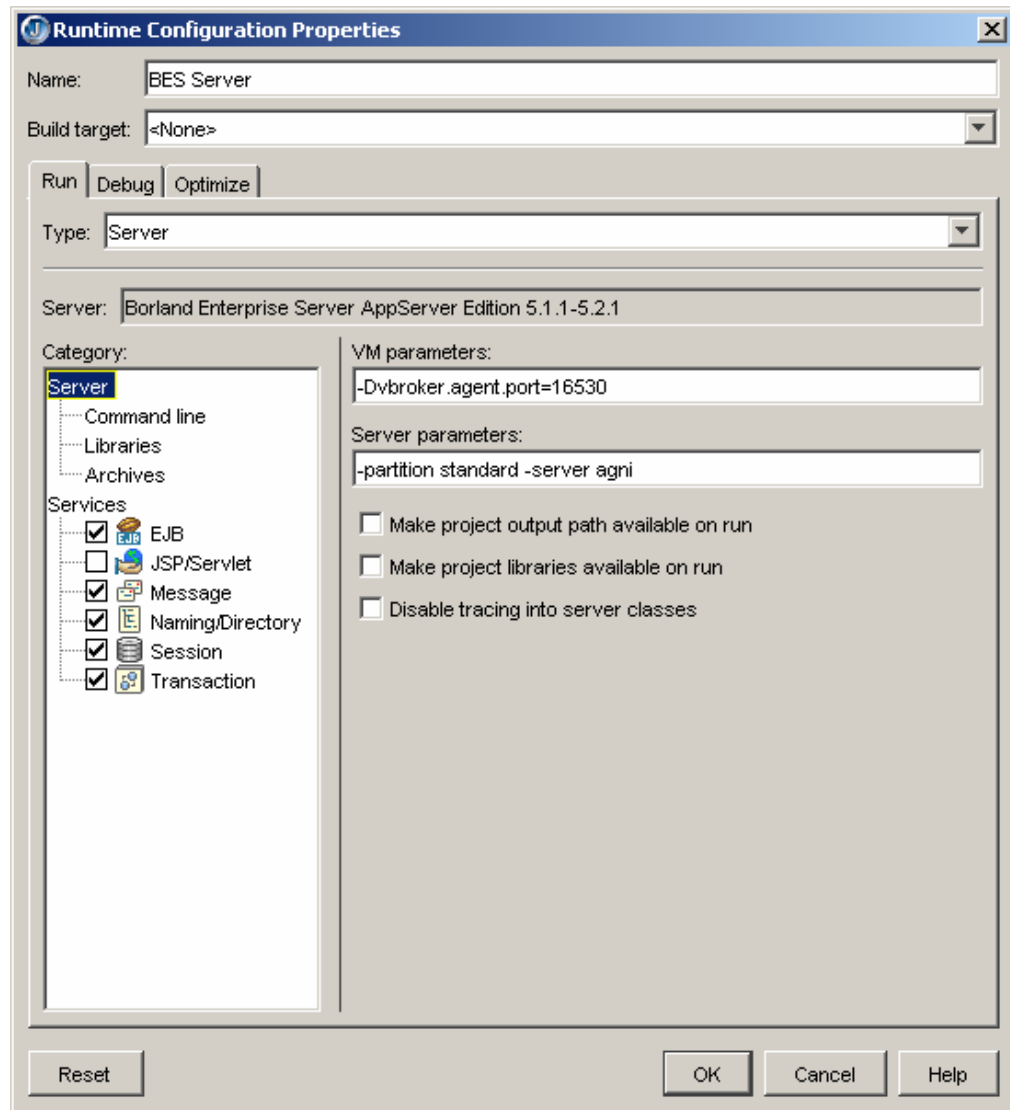
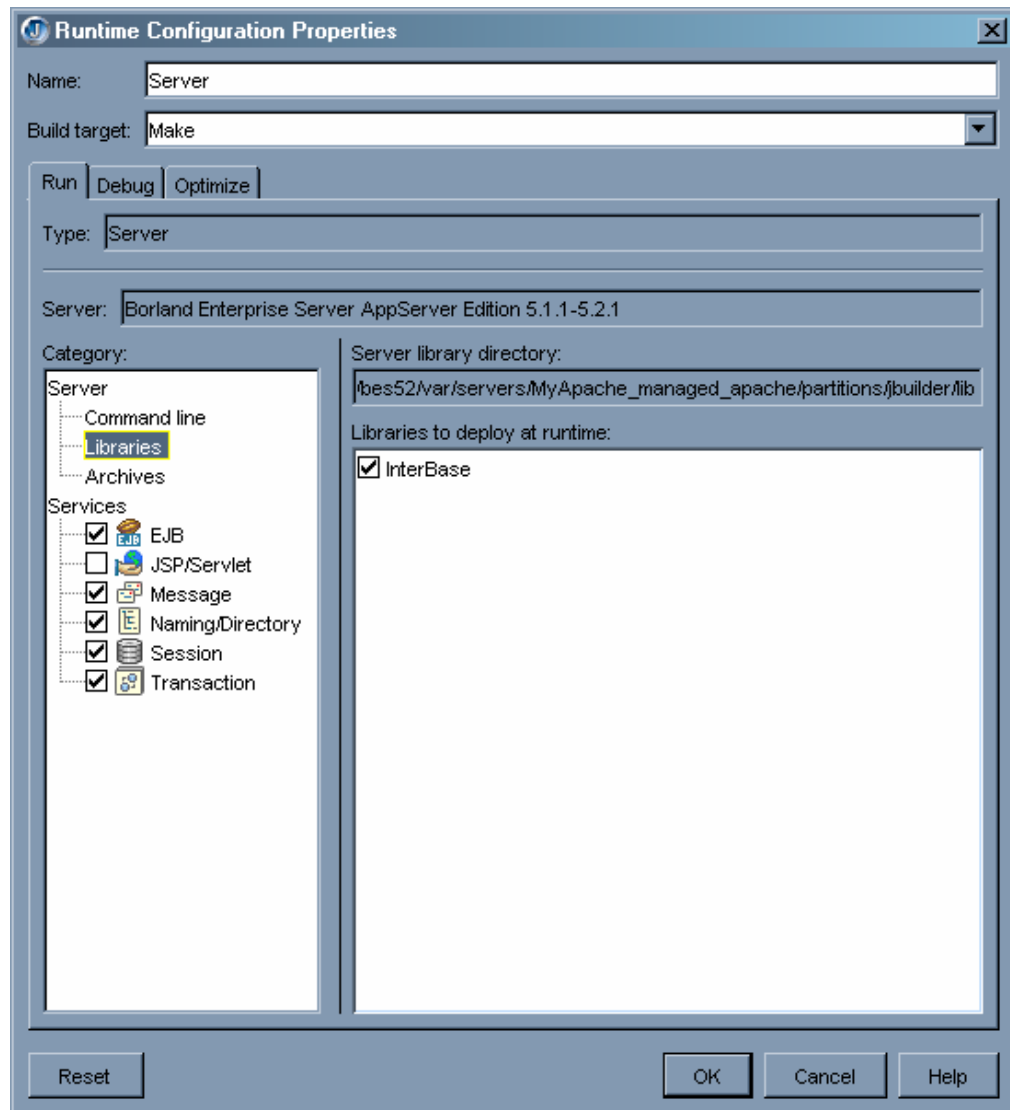


Figure 10: Runtime configurations for Borland Enterprise Server

Running Borland® Enterprise Server 5.2 inside JBuilder®

Click on the Libraries Category. Check the InterClient library box. This library will be automatically deployed to the partition and will be added to the partition's classpath.



```
Copyright (c) 1996-2002 Borland Software Corporation. All Rights Reserved.  
Covered by U.S. Patent 5,787,431, and other issued and pending patents.  
Portions Copyright (c) 1997-1999 Sun Microsystems, Inc. All Rights Reserved.  
Portions Copyright (c) 1999-2001 The Apache Software Foundation. All Rights Reserved.  
Developer's License (no connection limit)  
Copyright (c) 1996-2002 Borland Software Corporation. All rights reserved.  
License for JDataStore development only - not for redistribution  
Registered to:  
Development license  
Borland  
Server agni started
```



Figure 11: *JBuilder output for Borland Enterprise Server Management Agent*

From the JBuilder main menu bar, click **Tools|Borland Enterprise Server Management Agent** to start **Borland Enterprise Server 5.2**. The message pane should display the following information as shown in Figure 11.

Deploy the EJBModule JAR file to Borland® Enterprise Server

Now, we select the BES server runtime configuration, which will launch the Borland Enterprise Server. Click the **Run| Run Project (Run| Run BES Server** for JBuilder 8 users) from the main JBuilder window. Once the server is launched, the EJBModule.jar file is deployed to the server. The JBuilder output is shown in Figure 12.

```
ejbcontainer:      Memory (total)      65152 Kb      (max 65152 Kb)
ejbcontainer:      Memory (free)        93.0%
ejbcontainer:      -----
ejbcontainer:      Home (remote)       CustomerSession
ejbcontainer:      Total in memory     0
ejbcontainer:      Total in use        0
ejbcontainer:      -----
ejbcontainer:      Home (local)        Customer
ejbcontainer:      Total in memory     0
ejbcontainer:      Total in use        0
ejbcontainer:      =====
```

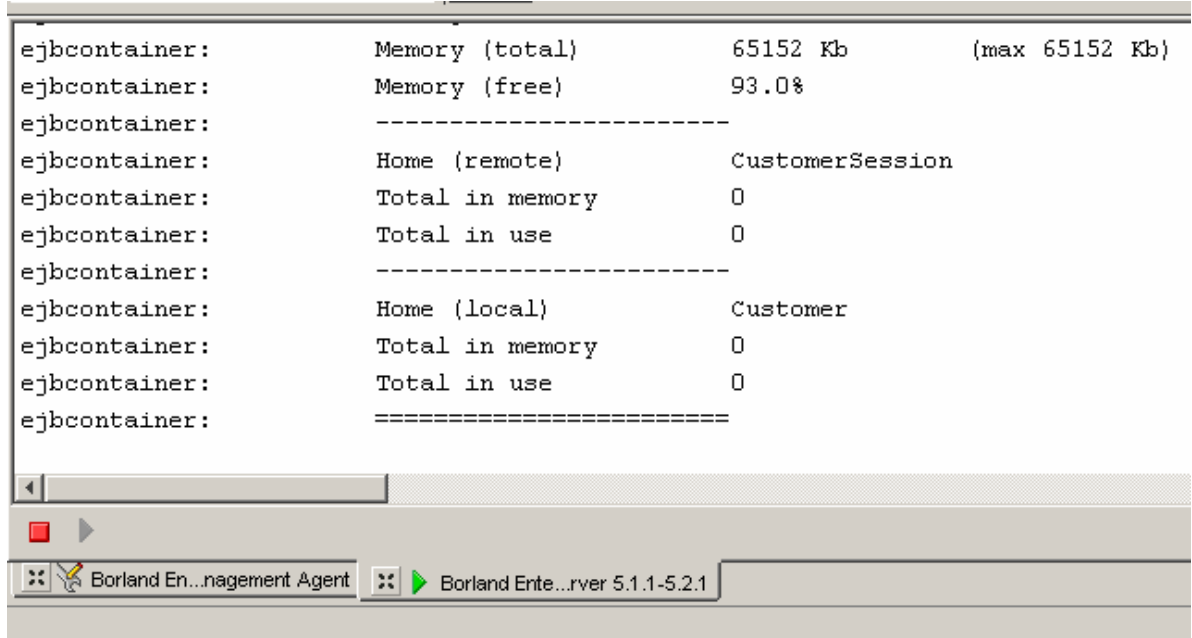


Figure 12: JBuilder message windows after starting and deploying the EJBModules

Creating the EJB test client

Now we need to test everything we have implemented. JBuilder provides a wizard to create an EJB test client.

From the JBuilder main menu bar, click **File | New | Enterprise** tab click **EJB Test Client** to start the three-step wizard.

Step 1. Select **Application** for the based type of EJB test client.

Step 2. Confirm EJB test client details, shown in Figure 13.

Step 3. Select **Create a runtime configuration** for the EJB test client, and accept the default name **CustomerSessionTestClient1**.

main method of **ProfileSessionTestClient1.java**.

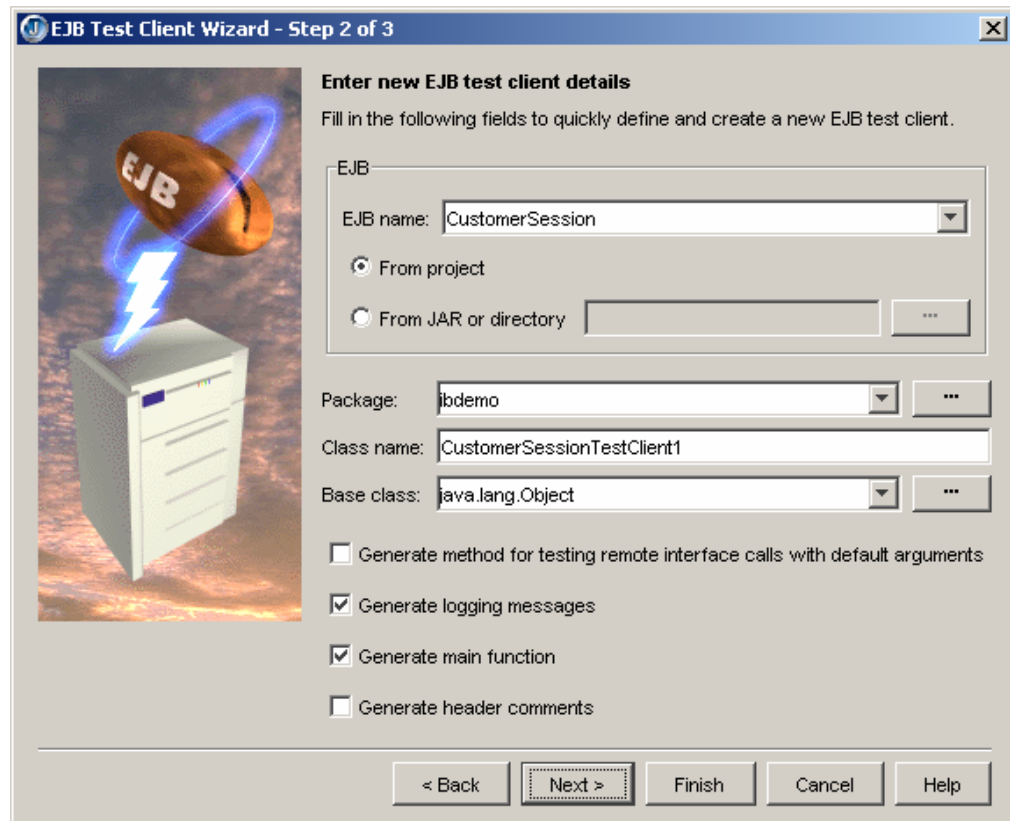


Figure 13: Step 2 of The EJB wizard

Upon completing the wizard, JBuilder will generate **CustomerSessionTestClient1.java** that contains methods for initializing the **CustomerSession** bean and wrappers for **CustomerSession** home and remote interfaces. We will use the **CustomerSessionHome** interface wrappers to create a remote interface reference to the actual deployed bean. Next, we can use the remote interface reference for accessing the remote interface methods. Here is sample code for the main method of **CustomerSessionTestClient1.java**

```
public static void main(String[] args) {  
  
    CustomerSessionTestClient1 client = new  
    CustomerSessionTestClient1();  
  
    // Create an instance of the ProfileSession bean  
  
    client.create();  
}
```

```
        client.getCustomer(new Integer("1002"));
    }
}
```

Running the test client

We will run the CustomerSessionTestClient1 with the CustomerSessionClient1 runtime configuration. Select **Run|Run CustomerSessionClient1.java using "CustomerSessionClient1"** from the main JBuilder window.

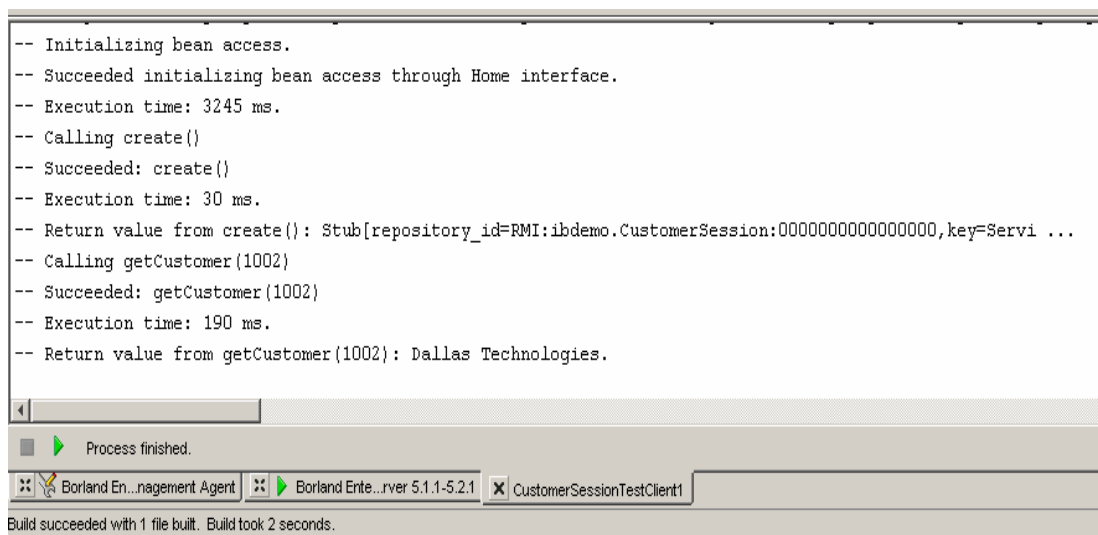


Figure 14: *The EJB test client running, with the result from the CUSTOMER table*

In order to verify that the Borland Enterprise Server has created an instance of the actual session bean, click the **Borland Enterprise Server 5.1.1** as shown in figure 15, you will see the updated statistics for the EJB Container.

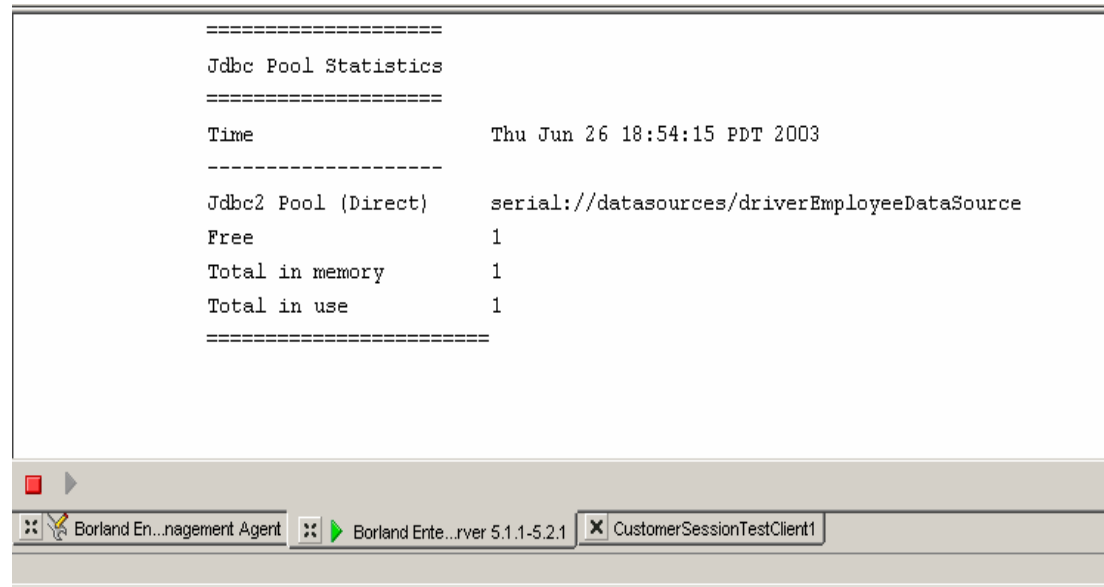


Figure 15: *Updated Container statistics*

Conclusion

This paper provides an overview of developing and testing J2EE-based EJB applications using Borland JBuilder, Borland InterBase, and Borland Application Server. JBuilder, Borland Enterprise Server, and InterBase integrate seamlessly to provide a fast, simple, and graphical way to create complex J2EE applications.

Additional information

Borland InterBase - <http://www.borland.com/interbase/index.html>

Borland JBuilder - <http://www.borland.com/jbuilder/index.html>

Borland Enterprise Server - <http://www.borland.com/besappserver/index.html>

Feedback and suggestions

Please send your feedback and suggestions to smistry@borland.com

Made in Borland® Copyright © 2004 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. All other marks are the property of their respective owners. Corporate Headquarters: 100 Enterprise Way, Scotts Valley, CA 95066-3249 • 831-431-1000 • www.borland.com • Offices in: Australia, Brazil, Canada, China, Czech Republic, Finland, France, Germany, Hong Kong, Hungary, India, Ireland, Italy, Japan, Korea, Mexico, the Netherlands, New Zealand, Russia, Singapore, Spain, Sweden, Taiwan, the United Kingdom, and the United States. • 21483