



BEA WebLogic Integration Kit for IBM VisualAge for Java

Tutorial

BEA WebLogic Integration Kit for IBM VisualAge for Java,
Version 3.5
Document Edition 1.1
February 2001

Copyright

Copyright © 2001 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, WebLogic, Tuxedo, and Jolt are registered trademarks of BEA Systems, Inc. How Business Becomes E-Business, BEA WebLogic E-Business Platform, BEA Builder, BEA Manager, BEA eLink, BEA WebLogic Commerce Server, BEA WebLogic Personalization Server, BEA WebLogic Process Integrator, BEA WebLogic Collaborate, BEA WebLogic Enterprise, and BEA WebLogic Server are trademarks of BEA Systems, Inc.

All other product names may be trademarks of the respective companies with which they are associated.

BEA WebLogic Integration Kit for IBM VisualAge for Java Installation Guide

Document Edition	Part Number	Date	Software Version
1.1	860-001004-001	February 2001	3.5

Contents

About This Document

What You Need to Know	vii
e-docs Web Site	viii
How to Print the Document	viii
Related Information	viii
Contact Us!	ix
Documentation Conventions	ix

1. Introduction

Overview	1-1
Using the Examples	1-2

2. Developing, Deploying, Using and Debugging EJBs

Overview	2-2
Developing, Deploying, Using and Debugging a Stateless Session EJB	2-3
Developing the EJB JAR	2-4
Looking at the StatelessSession Package	2-5
Starting the Build	2-7
Naming the EJB JAR File	2-8
Generating the Undeployable JAR	2-11
Generating the Container Classes	2-11
Generating the EJB JAR	2-12
Changing the Name or Location of the JAR file	2-13
Configuring BEA WebLogic Server to Run the EJB	2-13
Verifying the EJB Deployment	2-14
Starting the WebLogic Server	2-14
Checking the Server Messages on the consoles	2-17

Running a Client Java Application that Communicates with the Deployed EJB.....	2-19
Verifying the ClassPath.....	2-19
Using Command-Line Parameters	2-22
Running the Client Application.....	2-24
Debugging the Client Application and the Server Object.....	2-25
The IBM VisualAge for Java Integrated Debugger	2-26
Configuring the Server	2-27
Setting Breakpoints	2-29
Running the Client Application and the Server	2-31
Following the Processes in the Console and the Debugger	2-32
Stepping through the code.....	2-36
Developing, Deploying, Using and Debugging a Container Managed Entity EJB.....	2-40
Setting up the Oracle Database.....	2-42
Installing Oracle	2-42
Testing the Connection to the Oracle database.	2-43
Creating the Database Tables	2-44
Developing the EJB JAR.....	2-49
Looking at the ContainerManaged Package.....	2-49
Starting the Build	2-51
Naming the EJB JAR File	2-52
Generating the Undeployable JAR.....	2-52
Generating the Container Classes	2-53
Generating the EJB JAR	2-53
Configuring BEA WebLogic Server to Run the EJB	2-54
Verifying the EJB Deployment	2-58
Starting the WebLogic Server	2-58
Checking the Server Messages on the consoles	2-59
Running a Client Java Application that Communicates with the Deployed EJB.....	2-60
Verifying the ClassPath.....	2-60
Using Command-Line Parameters	2-61
Running the Client Application.....	2-62
Debugging the Client Application and the Server Object.....	2-68

3. Combining EJB with JMS and Servlet Technologies

Overview	3-2
Configuring BEA WebLogic Server for JMS	3-3
Creating a Database for JMS.....	3-4
Defining a JDBC Connection Pool for the JMS Database.....	3-6
Defining JMS ConnectionFactories	3-7
Defining JMS Topics and Queues.....	3-7
Configuring BEA WebLogic Server for the Servlet	3-8
Registering the Servlet	3-8
Setting the Servlet Classpath and Reloading Properties	3-8
Running the Client Application.....	3-10
Calling the Servlet from a Web Browser	3-14
Exporting the Classes to the Production BEA WebLogic Server	3-18

4. Developing an Applet Application

Overview	4-1
Setting Up the Database	4-2
Running and Debugging the Applet in the IBM VisualAge for Java	
Environment	4-3
Developing the Applet	4-3
Verifying the Classpath.....	4-4
Using Attributes and Parameters.....	4-6
Running the Applet	4-7
Debugging the Applet	4-9
Running the Applet in BEA WebLogic Server and a Web Browser	4-12

5. Exporting Classes

Overview	5-1
Exporting Classes to a Production BEA WebLogic Server	5-1

A. Appendix - Tips and Troubleshooting

Overview	A-1
Installing Service Packs.....	A-1
Using EJB Dynamic Deployment	A-3
Using Cloudscape Database	A-3



About This Document

This document describes how to use the BEA WebLogic Integration Kit for IBM VisualAge for Java Version 3.5 (the Integration Kit) to develop and debug your BEA WebLogic Server application from within IBM VisualAge for Java Version 3.5.

This document covers the following topics:

- Chapter 1, “Introduction”
- Chapter 2, “Developing, Deploying, Using and Debugging EJBs”
- Chapter 3, “Combining EJB with JMS and Servlet Technologies”
- Chapter 4, “Developing an Applet Application”
- Chapter 5, “Exporting Classes”

What You Need to Know

This document is intended mainly for application developers who are interested in building distributed Java applications that can be deployed within BEA WebLogic Server using The Integration Kit. It assumes familiarity with The Integration Kit platform, Java programming, and BEA WebLogic Server.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the “e-docs” Product Documentation page at <http://e-docs.bea.com>.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option.

A PDF version of this document is available on the Integration Kit documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the Integration Kit documentation Home page, click the PDF files button and select the document you want to print.

If you do not have Adobe Acrobat Reader, you can get it for free from the Adobe Web site at <http://www.adobe.com>.

Related Information

The following Integration Kit documents contain information that is relevant to using IBM VisualAge for Java Version 3.5 and BEA WebLogic Server.

For more information in general about IBM VisualAge for Java Version 3.5 and BEA WebLogic Server, refer to the following sources:

- BEA WebLogic Server Web Site at <http://e-docs.bea.com>
- IBM VisualAge for Java Version 3.5 site at <http://www.ibm.com/software/ad/vajava>

Contact Us!

Your feedback on the BEA Integration Kit documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the Integration Kit documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA Integration Kit release.

If you have any questions about this version of Integration Kit, or if you have problems installing and running Integration Kit, contact BEA Customer Support through BEA WebSupport at www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.

Convention	Item
<i>italics</i>	Indicates emphasis or book titles.
monospace text	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard. <i>Examples:</i> #include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float
monospace boldface text	Identifies significant words in code. <i>Example:</i> void commit ()
<i>monospace italic text</i>	Identifies variables in code. <i>Example:</i> String <i>expr</i>
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. <i>Examples:</i> LPT1 SIGNON OR
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
[]	Indicates optional items in a syntax line. The brackets themselves should never be typed. <i>Example:</i> buildobjclient [-v] [-o name] [-f <i>file-list</i>]... [-l <i>file-list</i>]...

Convention	Item
	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
...	<p>Indicates one of the following in a command line:</p> <ul style="list-style-type: none">■ That an argument can be repeated several times in a command line■ That the statement omits additional optional arguments■ That you can enter additional parameters, values, or other information <p>The ellipsis itself should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
. . . .	<p>Indicates the omission of items from a code example or from a syntax line.</p> <p>The vertical ellipsis itself should never be typed.</p>



1 Introduction

Topics discussed in this section include:

- Overview
- Using the Examples

Overview

IBM VisualAge for Java Version 3.5 (Professional or Enterprise Edition) is an integrated, visual environment that supports the complete cycle of Java program development.

BEA WebLogic Server is an award-winning Java application server for developing, deploying, and managing Web applications. It simplifies development of portable and scalable applications, and it provides interoperability with other applications and systems. BEA WebLogic Server also offers the most complete implementation of the Java 2 Enterprise Edition standard.

The *BEA Weblogic Integration Kit for IBM VisualAge for Java Version 3.5* (the Integration Kit) is a Java application that helps you develop and debug your BEA WebLogic Server applications from within IBM VisualAge for Java Version 3.5. All activity in VisualAge for Java Version 3.5 is organized around a workspace that contains the Java programs that you are developing. The workspace also contains all the packages, classes, and interfaces that are found in the standard Java class libraries, and other libraries that your classes may need. The Integration Kit for IBM VisualAge for Java Version 3.5 adds the classes required to run BEA WebLogic Server to your workspace. As you develop and debug your applications you can add classes and projects to your workspace.

This document discusses some of the examples included in the installation and, using these examples, describes the most common scenarios in which the tools provided by The Integration Kit are used in developing an application.

The focus of the scenarios is the usage of EJBs by themselves or in combination with other Java technologies, such as JDBC, JMS, Servlets, and Applets. The descriptions of these scenarios address the development and testing phase of the application, and they provide brief hints on how to use the developed application in a production environment. You can get more information about using BEA WebLogic Server in a production environment from the BEA WebLogic Server documentation.

Using the Examples

The Integration Kit installation creates an IBM VisualAge for Java Version 3.5 project, `WebLogic Examples`, that contains example code illustrating how to use many of the capabilities of BEA WebLogic Server. It is recommended that you work through all the examples included in the Integration Kit before attempting to create your own EJBs. The examples illustrate all the process steps involved.

In addition to the Java code in the project, some of the examples also require configuring the server settings in order to run properly, and some have command-line arguments which you can modify.

For instructions on how to run each example, refer to the documentation on the BEA WebLogic Server examples. These documents describe how to build and run the examples from the command line. Most of the instructions also apply to running the examples within IBM VisualAge for Java Version 3.5, but there are a few differences to keep in mind:

- **Compiling the examples**

Code is *automatically* compiled in the IBM VisualAge for Java workspace. For this reason, you can ignore instructions concerning compiling the examples from BEA WebLogic Server.

- Setting server properties

For most examples, properties must be configured in the `weblogic.properties` file. The server reads its properties from the `weblogic.properties` file in the WebLogic Server installation directory. If the server is running in IBM VisualAge for Java Version 3.5, you must stop the server and restart it after setting the example properties.

- Using command-line parameters to pass arguments to the executable class

To set command-line arguments, complete the following procedure:

- a. In the workspace, select the main class for the example.
- b. From the menu bar, select Selected→Properties to open the Properties window.
- c. On the Program tab in the Properties window, enter the arguments in the Command Line Arguments field.
- d. Click OK.

- Cloudscape database

The BEA WebLogic Integration Kit for IBM VisualAge for Java Version 3.5 does not support the use of a Cloudscape database because of problems that occur when Cloudscape is run in a VisualAge environment. This issue is currently being investigated by IBM Support (PMR 15142,519,000).

Most of the examples shipped with BEA WebLogic Server use the `demoPool` database connection pool, which is set up to use a pre-configured Cloudscape database that is installed with BEA WebLogic Server. In order to run these examples in IBM VisualAge for Java Version 3.5, you must create the example tables in another database, such as Oracle, and change the `demoPool` configuration in `weblogic.properties` to use that database.

2 Developing, Deploying, Using, and Debugging EJBs

Topics discussed in this section include:

- Overview
- Developing, Deploying, Using, and Debugging a Stateless Session EJB
- Developing, Deploying, Using, and Debugging a Container-Managed Entity EJB

The latter two of these sections provide information about the following:

- Setting up the Oracle Database (for the Container Managed Entity EJB only)
- Developing the EJB JAR
- Configuring BEA WebLogic Server to Run the EJB
- Verifying the EJB Deployment
- Running a Client Java Application that Communicates with the Deployed EJB
- Debugging the Client Application and the Server Object

Overview

In this section, we will look at developing, deploying, using, and debugging Enterprise JavaBeans using two examples that demonstrate different aspects of Enterprise JavaBeans. The first example uses a Stateless Session EJB; the second uses a Container-Managed Entity EJB.

The code used in both examples is based on examples that are shipped with BEA Weblogic Server and adapted to work with an IDE (IBM VisualAge for Java). The code elements are provided in the `WebLogic Examples` project when you install the Integration Kit.

The `WebLogic Examples` project also contains other EJB examples included in the BEA WebLogic Server distribution. You can easily experiment with them from inside IBM VisualAge for Java without any further configuration (besides that required by the specific example).

To get the most out of these examples, first read through the source code files. Start with the XML deployment files to see the general structure of the EJB. Notice which classes are used for the different objects and interfaces. Then look at the client file, `Client.java`, to see how the application works.

Developing, Deploying, Using, and Debugging a Stateless Session EJB

In this section you will build, deploy, and debug a stateless session Enterprise JavaBean called `TraderBean`.

Using this Enterprise JavaBean, the client application will perform the following tasks:

1. Create a Trader.
2. Buy and sell shares of BEAS, MSFT, AMZN, and HWP. (The EJB does not actually buy or sell; it simulates the actions of accessing a database.)
3. Remove the Trader.

This application will demonstrate the following:

- How the client maintains a persistent state, such as the change in the cash account, across repeated calls to the session EJB
- How to use application-defined exceptions and utilities

The section will walk you through several steps that correspond to the steps in a typical EJB application development process:

1. Developing the EJB JAR
2. Configuring BEA WebLogic Server to Run the EJB
3. Verifying the EJB Deployment
4. Running a Client Java Application that Communicates with the Deployed EJB
5. Debugging the Client Application and the Server Object

In this example you can configure BEA WebLogic Server to both run the EJB and verify the EJB deployment before you develop the EJB JAR. This is because a pre-built JAR file is installed in the `webLogic\myserver` directory. Verifying the EJB Deployment can be done before developing the EJB JAR because the example includes the pre-built JAR file installed in the correct location. However, Developing the EJB JAR has to be successfully completed before you can do Running a Client Java

Application that Communicates with the Deployed EJB and Debugging the Client Application and the Server Object. This is because these steps depend on having all the generated container classes in the workspace.

The code used in this section is based on the example `examples.ejb.basic.statelessSession` which is shipped with BEA WebLogic Server and adapted to work with an IDE (IBM VisualAge for Java). All the code elements (`.java` and `.xml`) for this example were included in the `WebLogic Examples` project when you installed the Integration Kit.

In general, you will need to adjust certain BEA WebLogic Server properties to match your setup. To deploy the EJB you will need to edit the property that begins with `weblogic.ejb.deploy` in the `weblogic.properties` file. This property is commented out in the default properties file; make sure that you uncomment all the lines of the property.

Note: If you change the BEA WebLogic Server installation root to another location *after* installing it, you must reconfigure the EJB tools using the Configure Tools tool (From the menu bar, select `Selected→Tools→WebLogic Server Tools→Configure Tools`). This tool associates the EJB tools with the new root directory of the BEA WebLogic Server distribution. This is necessary because the EJB tools depend on classes in the BEA WebLogic Server distribution that have not been imported into the IBM VisualAge for Java workspace, and because the Generate EJB JAR tool must know where to install the generated JARs. This tool was run as part of the installation process and does *not* need to be re-run unless you move the location of the BEA WebLogic Server distribution after installation. For more information see the *Installation Guide* for the Integration Kit.

Developing the EJB JAR

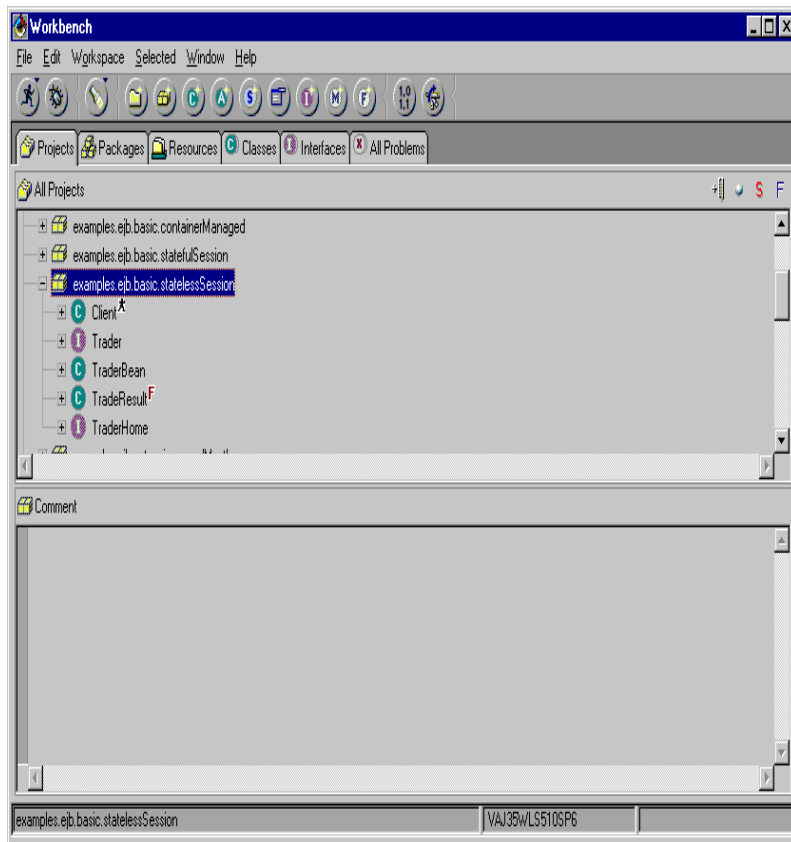
All the code elements for the stateless session example have already been developed and are installed in the `WebLogic Examples` project. We will look at these elements in the `StatelessSession` package and then build the deployable JAR file. A *pre-built* EJB JAR file is already installed in the directory `WebLogic\myserver` (where `WebLogic` is the installation directory of BEA WebLogic Server), but in order to use and debug the EJB you need to generate the container classes.

Looking at the StatelessSession Package

To examine the StatelessSession package in the workspace:

1. Start IBM VisualAge for Java.
2. On the Projects tab of the Workbench, select the package `examples.ejb.basic.statelessSession` in the project `WebLogicExamples`.

Figure 2-1 The `statelessSession` Package in the Projects Tab



Notice that the `examples.ejb.basic.statelessSession` package already contains:

- Executable class: `Client`
- Server EJB class: `TraderBean`
- Helper classes: `TradeResult`
- EJB interfaces: `Trader` and `TraderHome`

During the Integration Kit installation the Deployment Descriptors were included in the project's resource folder:

```
VisualAge\IDE\project_resources\WebLogic Examples\examples\ejb\
basic\statelessSession
```

The installation directory for IBM VisualAge for Java, in this case, is *VisualAge* (in this example, `C:\Program Files\IBM\VisualAge for Java`). It contains the following files:

- `ejb-jar.xml`
- `weblogic-ejb-jar.xml`

In a real development situation, you must do the following tasks:

- Create the java and XML files yourself using an IDE, such as IBM VisualAge for Java. (It would be a good exercise to walk through the code, so that you understand what is required to build an EJB.)
- Create a project (`WebLogic Examples`) and a package (`examples.ejb.basic.statelessSession`) that you can use to develop your EJB and client inside the IBM VisualAge for Java workspace. Using the IBM VisualAge for Java Import utility, import the Deployment Descriptors into the corresponding package (`examples.ejb.basic.statelessSession`).

Warning: Do not modify any files in the IBM VisualAge for Java `VisualAge\IDE\project_resources` file system directory tree.

The deployable JAR file (`ejb_basic_statelessSession.jar`) is already built and installed in the directory `WebLogic\myserver` (where *WebLogic* is the directory in which BEA WebLogic Server is installed).

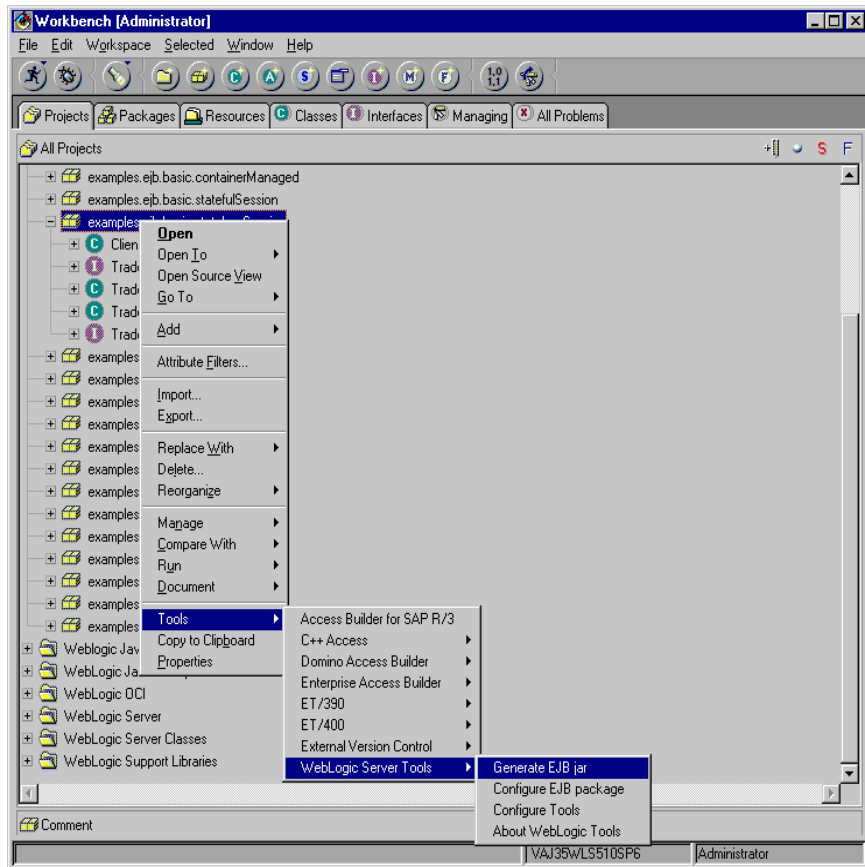
Starting the Build

For the client application to be able to communicate with the deployed EJB, you need to have all the generated container classes in the workspace, so you must build the EJB JAR.

To start the build of the EJB JAR:

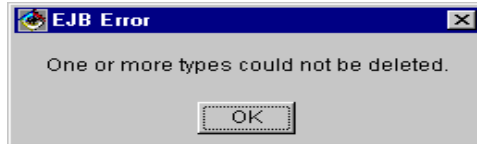
1. Select the `examples.ejb.basic.statelessSession` package in the Project WebLogic Examples.
2. From the menu bar, select **Selected**→**Tools**→**Generate EJB JAR**.

Figure 2-2 Selecting Generate EJB JAR in the Projects tab



Note: Before using the Generate EJB JAR tool, users of the Enterprise Edition of IBM VisualAge for Java must first create an open edition of the EJB package. To create an open edition, select the package and then select the menu items Selected→Manage→Create Open Edition. If you are not using an open edition of the package, the Enterprise Edition reports an error message (see Figure 2-3).

Figure 2-3 EJB Error message

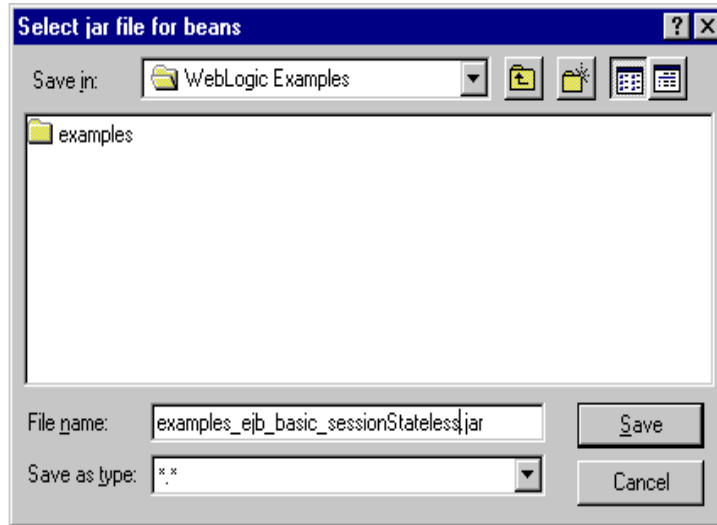


The build process involves several steps, each of which is announced by a Generating EJB message as it happens.

Naming the EJB JAR File

If you are building the package for the first time, you are prompted for an output JAR filename.

Figure 2-4 Select JAR File for Beans Window

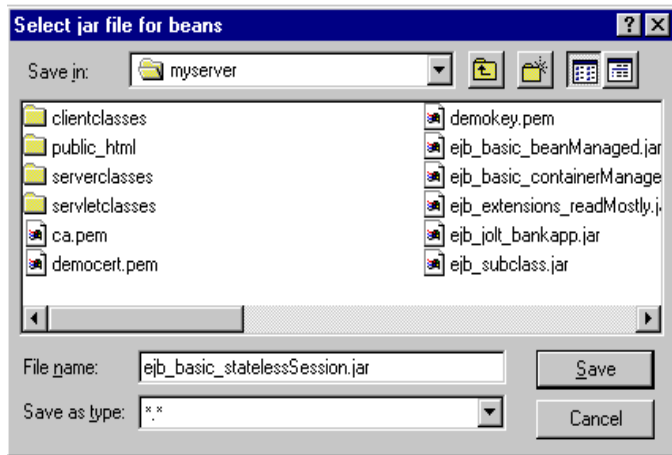


The tool makes the following suggestions for the location and name of the (deployable) JAR file:

- **Location:** The resource directory in IBM VisualAge for Java where the project's resources reside (in this case *VisualAge\IDE\project_resources\WebLogic Examples* where *VisualAge* is the installation directory for IBM VisualAge for Java, in this example *C:\Program Files\IBM\VisualAge for Java*).
- **Name:** The name of the project appended with the suffix (in this case *examples_ejb_basic_sessionStateless.jar*).

In this example we will change the name and location to the values used in BEA WebLogic Server Examples to avoid confusion.

Figure 2-5 Setting Values for JAR File Location and Name



As shown in Figure 2-5, we will use the following settings:

- Location: *WebLogic\myserver*

Here *WebLogic* is the installation directory for BEA WebLogic Server. In this example, the installation directory is `C:\weblogic`

- File Name: `ejb_basic_sessionStateless.jar` (similar to the name of the corresponding file in the BEA WebLogic Server example)

Note: If the values suggested by the tool are used, you must modify the `weblogic.properties` file accordingly (For details, see the following section “Configuring BEA WebLogic Server to Run the EJB”).

Generating the Undeployable JAR

Using the compiled classes in the package (EJB class, Interfaces, Client) and the Deployment Descriptors, the build process will generate an undeployable JAR (meaning the JAR cannot be deployed in any Web Application Server).

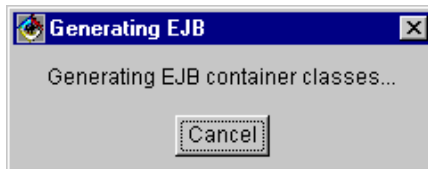
The name of the undeployable JAR file will be created by adding the prefix `std_` to the JAR name provided above. In this example the undeployable JAR will be called `std_ejb_basic_statelessSession.jar`. The file will be installed in the specified directory, in this example `WebLogic/myserver` where *WebLogic* is the installation directory for BEA WebLogic Server (in this example, `C:\weblogic`).

Generating the Container Classes

If you have previously done a build and the generated container classes are already in the `statelessSession` package, subsequent build processes will delete these existing container classes before generating new ones. (A message will inform you of the removal process and you will be able to see the classes being removed, one by one, from the workspace.)

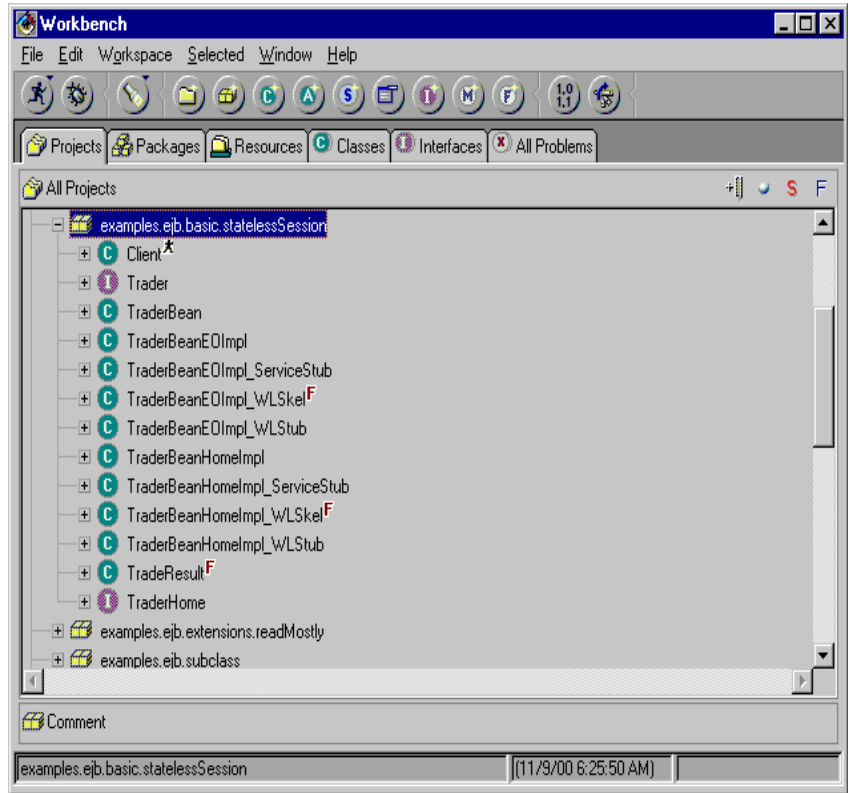
The Generating EJB message will announce the generation of the container classes.

Figure 2-6 The Generating EJB message



You will see the classes being added one by one to the workspace.

Figure 2-7 The generated container classes in the `statelessSession` package



Generating the EJB JAR

Using the compiled classes in the package (EJB class, Interfaces, Client), the Deployment Descriptors and the newly generated container classes, the build process will generate a deployable JAR (meaning that the JAR can be deployed in BEA WebLogic Server). In this example the JAR will be called `ejb_basic_statelessSession.jar`. The file will be installed in the specified directory, `WebLogic/myserver` where `WebLogic` is the installation directory for BEA WebLogic Server (in this example, `C:\weblogic`).

When the build process is complete, you will get an EJB Generated confirmation message. This same message will be recorded in the log.

Figure 2-8 The EJB Generated confirmation message



Changing the Name or Location of the JAR file

If, at a later stage, you need to change the name and location of the JAR file, use the Configure EJB package tool.

To change the name and location of the JAR file:

1. In the Projects tab in the Workbench, select the project or the EJB package inside it.
2. From the menu bar, select Selected→Tools→Weblogic Tools→Configure EJB package.

Configuring BEA WebLogic Server to Run the EJB

In this example the EJB does not communicate with a database, so you are not required either to set a database and its tables, or configure BEA WebLogic Server for Database/Pool access. “Developing, Deploying, Using and Debugging a Container Managed Entity EJB,” which discusses the case of an Entity EJB that connects to a Database, shows you how to do that.

The bean’s JAR is already in the correct location in the *WebLogic/myserver* folder.

To successfully deploy and use the EJB you must add the path to the JAR file to the `weblogic.ejb.deploy` property in the `weblogic.properties` file. The `weblogic.properties` file is located in the root installation of the BEA WebLogic Server (the default: `c:\weblogic`).

A commented-out version of this path can be found in the `weblogic.ejb.deploy` property. You will need to uncomment and adjust the property depending on which EJBs you are building and deploying, or if the location of the files differs from the installed location. For this example you will have to uncomment:

```
weblogic.ejb.deploy=\n\n    C:/weblogic/myserver/ejb_basic_statelessSession.jar
```

If you did not change the default name and location for the JAR file in the previous section, “Developing the EJB JAR”, you will have to set the property in the `weblogic.properties` file to correspond to your JAR file:

```
weblogic.ejb.deploy=\n\n    VisualAge\\IDE\\project_resources\\WebLogic Examples/\n    examples_ejb_basic_sessionStateless.jar
```

where *VisualAge* is the installation directory for IBM VisualAge for Java (in this example, `C:\\Program Files\\IBM\\VisualAge for Java`).

Note: If you configure server properties in the `weblogic.properties` file while the server is running in IBM VisualAge for Java you will need to stop the server and restart it.

Verifying the EJB Deployment

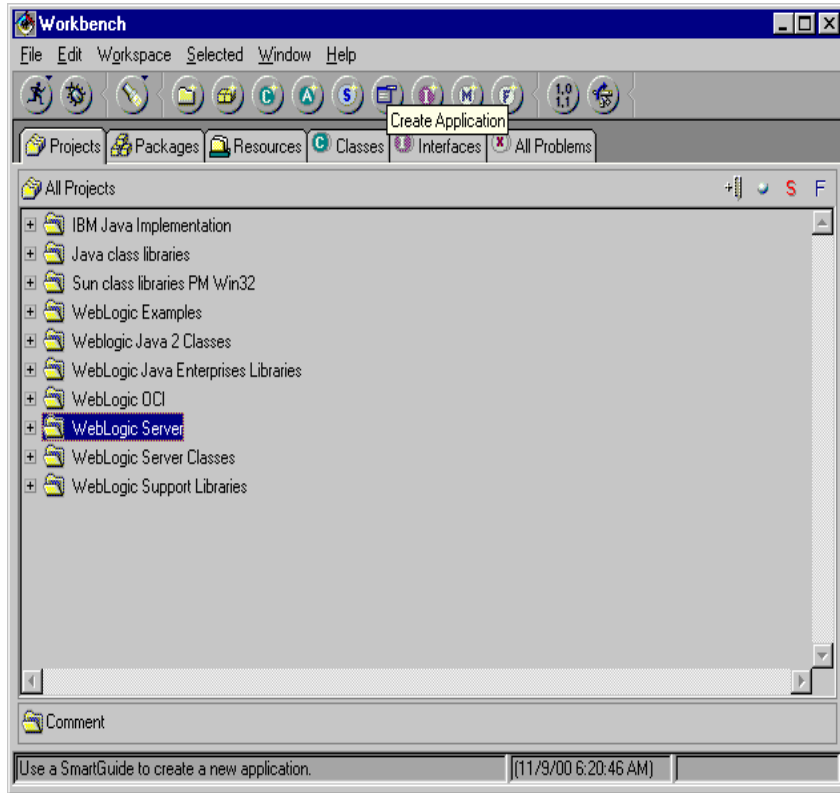
To verify if the EJB deploys correctly you have to start the server and look at the messages displayed on the VisualAge console and the WebLogic console.

Starting the WebLogic Server

To start the WebLogic server:

1. Start IBM VisualAge for Java.
2. In the Projects tab of the Workbench select `WebLogic Server`.

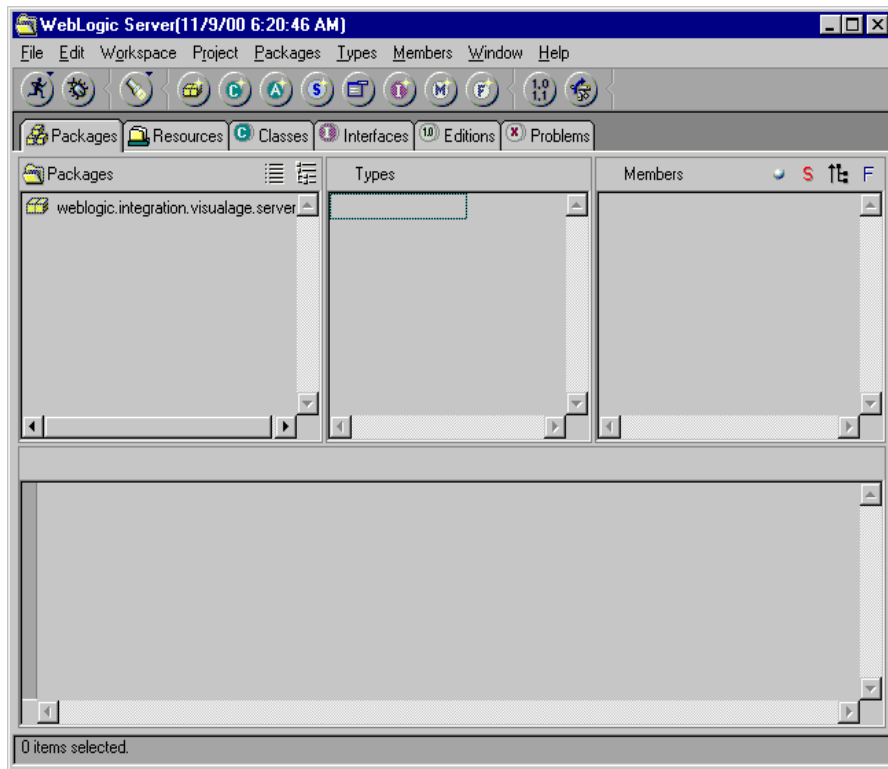
Figure 2-9 The `weblogic` Server project in the Projects tab



3. Do one of the following:

- Click the Run button.
- Right-click on the `WebLogic Server` project and select Run→Run main.
- Double-click the `WebLogic Server` project. This will open a separate Weblogic Server window containing just the `weblogic.integration.visualage.server` package. In the Weblogic Server window click the Run button, or right-click on `WebLogic Server` and select Run→Run main.

Figure 2-10 The Weblogic Server window



While the server is running, a WebLogic Server message will be displayed that can be used to shut down the server.

Figure 2-11 The Weblogic Server message

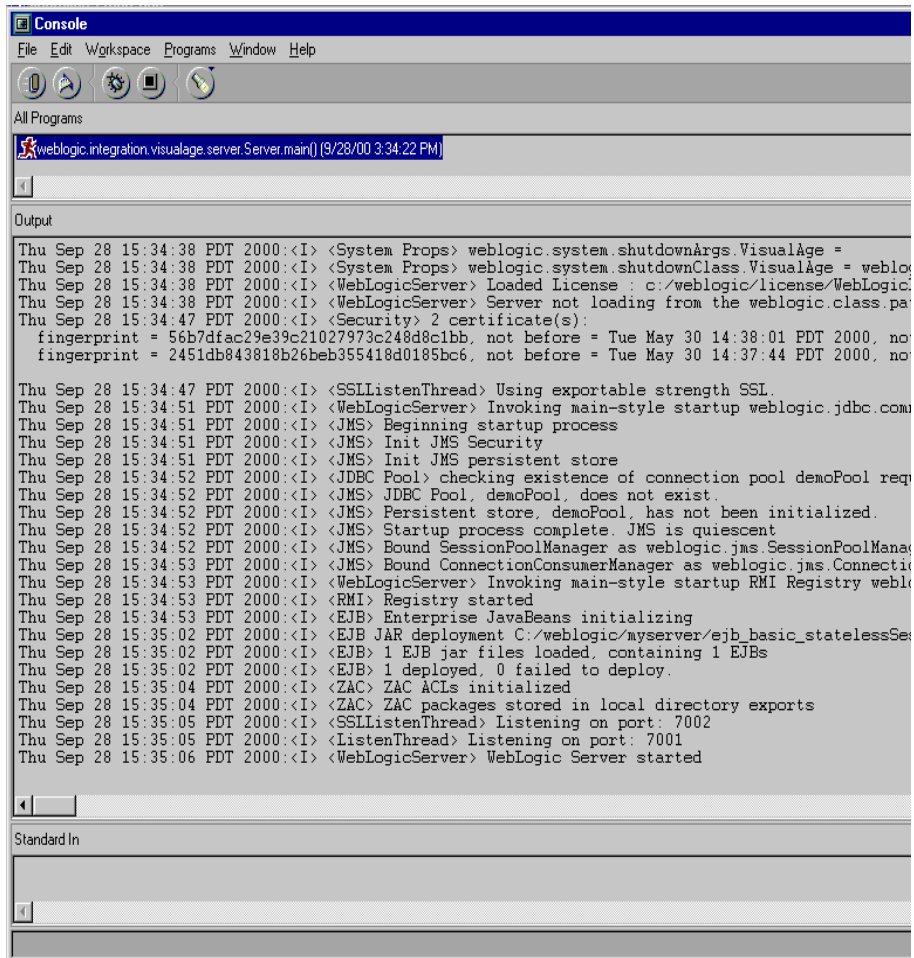


Checking the Server Messages on the consoles

To check whether the server has started correctly and whether the EJB has been deployed correctly do either of the following:

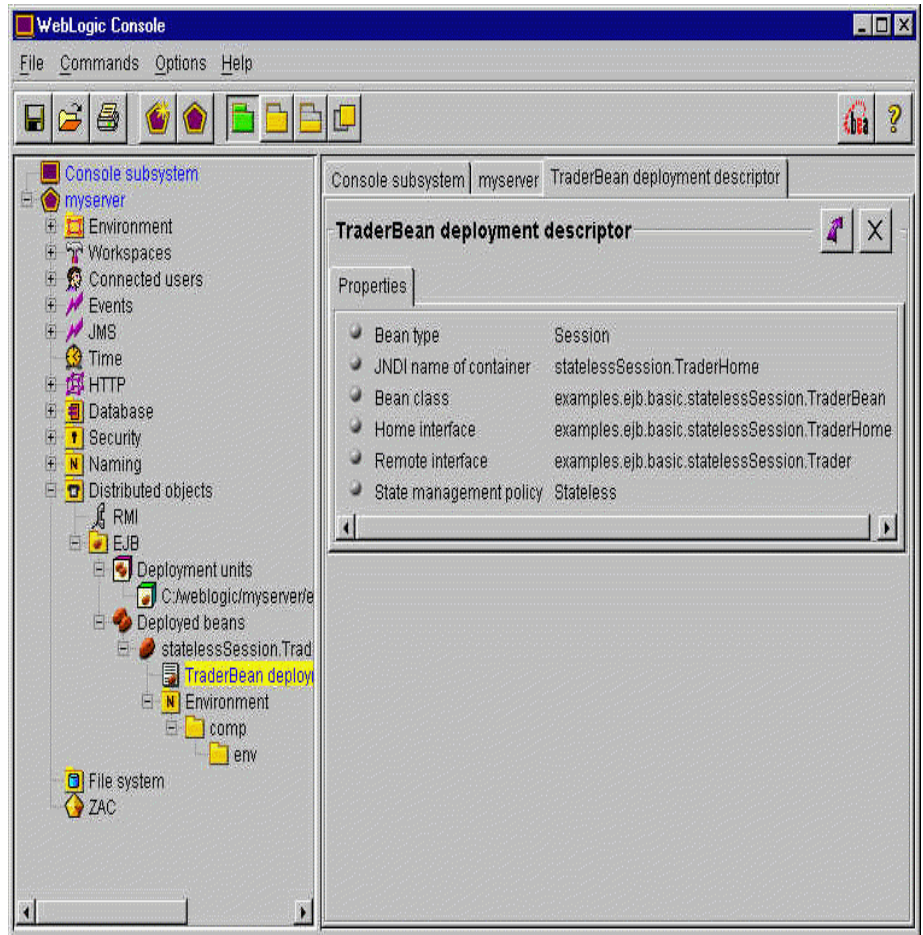
- Check the messages displayed on the console

Figure 2-12 The console showing messages from the server



- Open the *WebLogic* console, attach to the BEA WebLogic Server, and examine the EJB under Distributed Objects. You should see, and be able to monitor the activity of, the deployed EJB interface `statelessSession.TraderHome`.

Figure 2-13 The WebLogic console



Running a Client Java Application that Communicates with the Deployed EJB

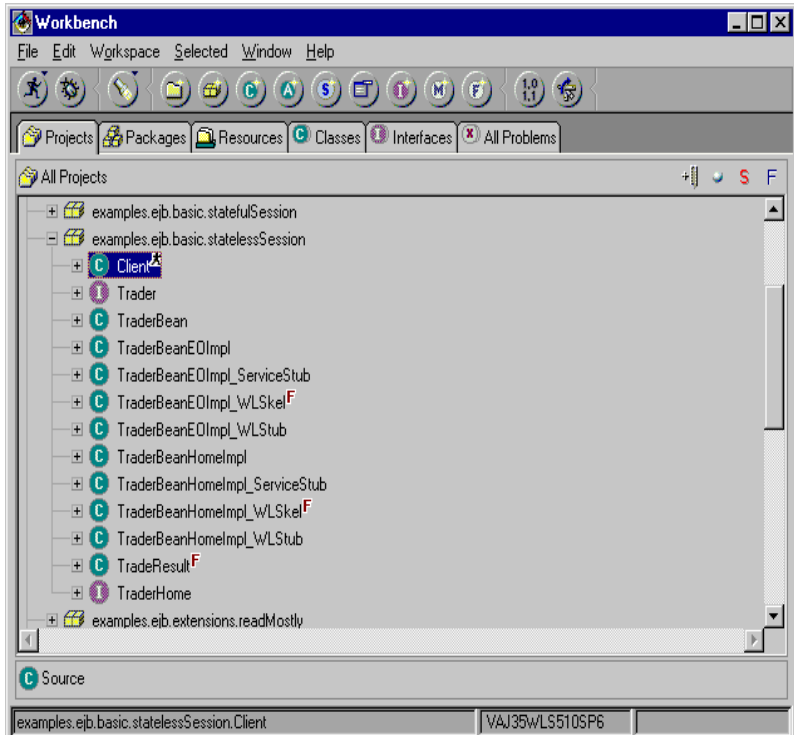
Before running the client application you must verify that the Integration Kit has provided the complete classpath for the application. You can also set Command-line parameters. The output from the client application will appear on the console.

Verifying the ClassPath

To verify the complete classpath for the client application:

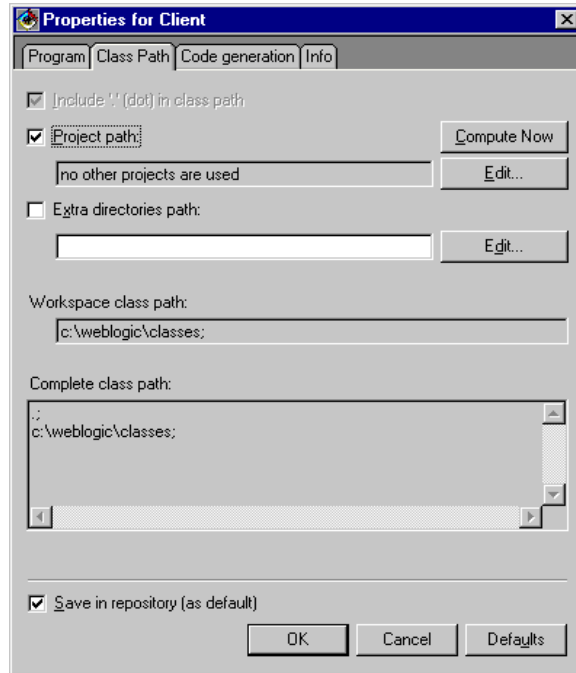
1. Right-click on the `Client` runnable class in the `examples.ejb.basic.statelessSession` package in the project `WebLogicExamples`.

Figure 2-14 The Client class in the Projects tab of the Workbench



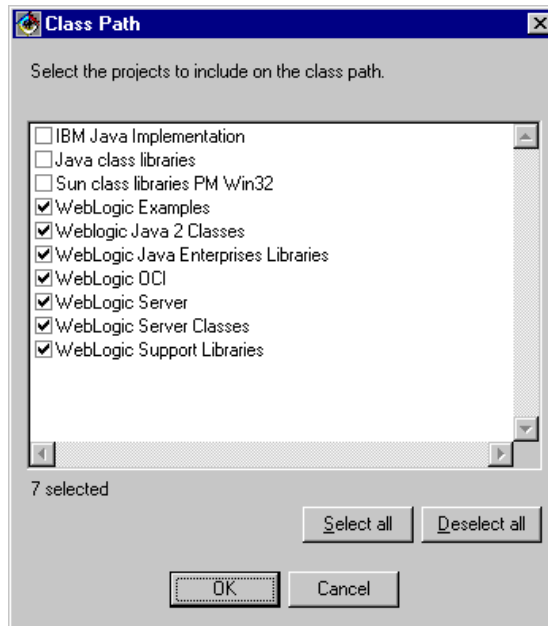
2. Select Properties from the context menu to open the Properties window.

Figure 2-15 The Class Path tab in the Properties window



3. In the Class Path tab of the Properties window, click the Edit button for the Project Path field to open the Class Path window.

Figure 2-16 The Class Path window



4. Verify that there are checkmarks in all the boxes corresponding to the BEA WebLogic Server projects that were added to the IBM VisualAge for Java workspace by the Integration Kit's Installer. If any of these projects are not checked, check them.
5. Click OK.

The path to each of the checked projects is displayed in the Complete class path field on the Class Path tab of the Properties window.

Using Command-Line Parameters

There are three command-line parameters in this application. The first parameter (*url*) gets a default argument and only needs to be changed if the default settings are not being used. The other two parameters are optional.

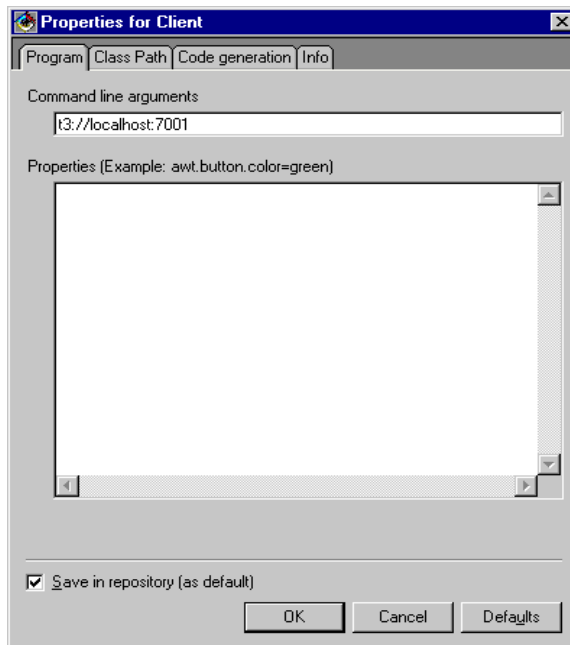
Command-line parameters are interpreted in this order:

1. *url*: URL of server (default such as `t3://localhost:7001`)
2. *user*: User name (default null)
3. *password*: User password (default null)

To edit the command-line arguments:

1. In the Properties window select the Program tab.
2. Enter the arguments in the Command line arguments text field with spaces between each argument.
3. Press OK.

Figure 2-17 The Program tab in the Properties window



Note: If you are not running the BEA WebLogic Server with its default settings, you will have to supply the command-line argument:

```
t3://WebLogicURL:Port
```

where:

WebLogicURL is the domain address of the BEA WebLogic Server

Port is the port that is listening for connections
(weblogic.system.ListenPort)

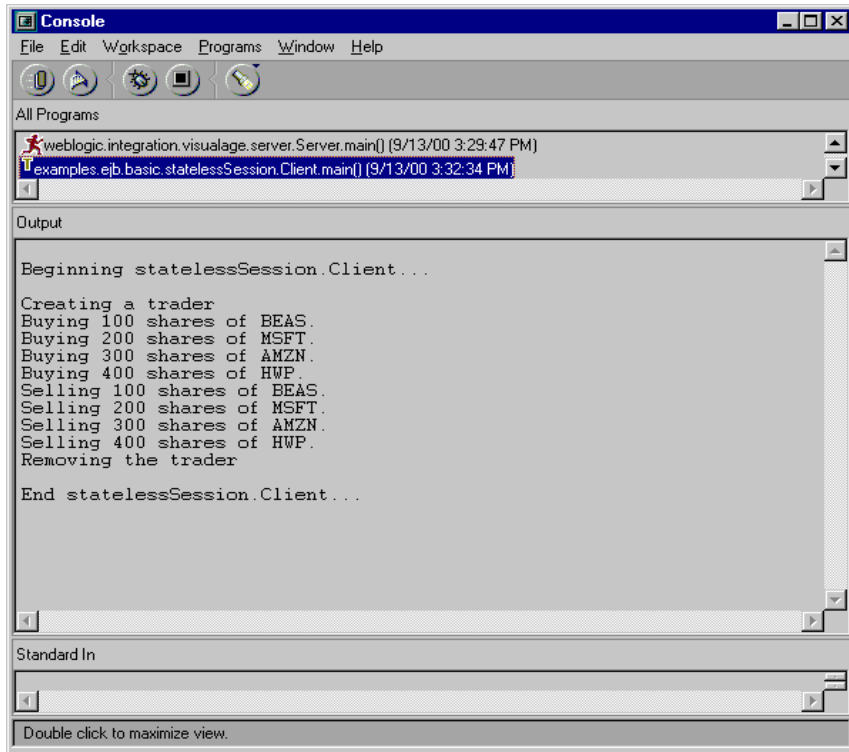
Running the Client Application

To run the client do one of the following:

1. Select the `Client` class in the `examples.ejb.basic.statelessSession` package and click on the Run button.
2. Right-click on the `Client` class in the `examples.ejb.basic.statelessSession` package and select Run→Run main.

When running this example, you should get output from the client application similar to that shown in Figure 2-18 below.

Figure 2-18 The console showing messages from the client application



Debugging the Client Application and the Server Object

We will use the IBM VisualAge for Java integrated debugger to debug the Client application and the server object. From the debugger, you can launch Inspectors to look at and modify variable values for suspended threads, watches to evaluate expressions as you step through a program, and an Evaluation window where you can evaluate an expression during debugging.

Before running the application you have to configure the server.

You will set breakpoints in the server-side code and the client code and then follow the progress of the two processes using the console and the debugger and trace the code when it is invoked from a client.

The IBM VisualAge for Java Integrated Debugger

IBM VisualAge for Java integrated debugger assists in debugging applets and applications running in the IDE.

You can open the debugger manually while a program is running to inspect threads and variables. Also, the debugger will open automatically, with the current thread suspended, for any of several reasons:

- A breakpoint in the code is encountered (We will experiment with this in the example.)
- A conditional breakpoint that evaluates to true is encountered.
- An exception is thrown and not caught.
- An exception selected in the Exceptions page is caught.
- A breakpoint in an external class is encountered.

Once the debugger is open and a thread is suspended, you can work with the program in the following ways:

- Inspect visible variable values
- Modify most variable values
- Step through methods
- Modify source code for methods in the workspace
- Replace methods with other editions from the repository
- Modify, clear or disable breakpoints
- Evaluate expressions in the Source pane or the Evaluation window
- Define expressions to watch as you step through programs

Using the debugger, you can optionally generate and view the class loading and initialization trace.

To use the IBM VisualAge for Java debugger, server-side classes must be in an IBM VisualAge for Java project. You can either create the project in IBM VisualAge for Java, or if the code already exists, you can import it into IBM VisualAge for Java. It is acceptable for client-side and server-side code to be in the same project.

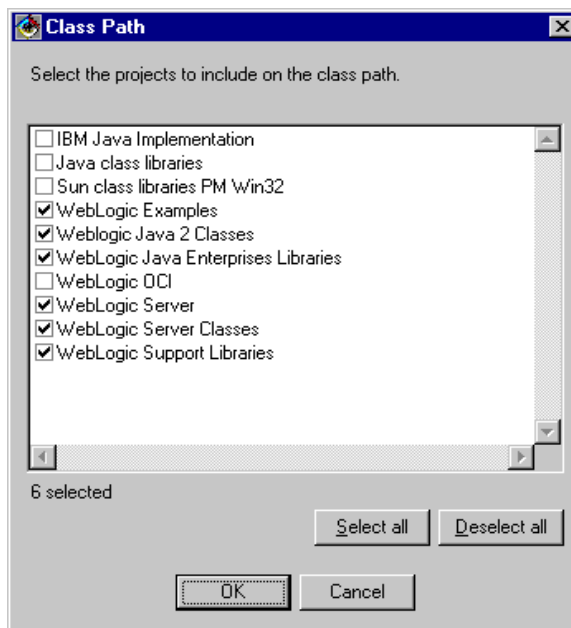
Configuring the Server

Because the object we want to debug is a server-side object, make sure that the `WebLogic Examples` project is added to classpath for the `Server` class. The server is configured in the `weblogic.properties` file.

To add the project to the classpath for the `Server` class.

1. In the Projects tab of the Workbench, expand the `WebLogic Server` project, then the `weblogic.integration.visualage.server` package.
2. Right-click on the `Server` runnable class and select Properties from the context menu to open the Properties window.
3. In the Class Path tab, click the Edit button for the Project Path field to open the Class Path window.

Figure 2-19 The Class Path window



4. Make sure that the check box for the project `WebLogic Examples` is checked.
5. Make sure that the check box for `WebLogic OCI` is not checked.
6. Click OK.

To configure the BEA WebLogic Server, if you have not yet done this, edit `weblogic.properties` as required (see “Configuring BEA WebLogic Server to Run the EJB”).

Setting Breakpoints

We will set breakpoints in both the server-side code and the client code, and then trace the code when it is invoked from a client.

To set a breakpoint in source code in the IDE:

1. Go to the Workbench or any browser that shows the source code for the program where you want to suspend the thread.
2. Do either of the following:
 - Place the cursor in the line of code where you want the breakpoint and select Edit→Breakpoint.
 - Double-click to the left of the line of code where you want the breakpoint.

A breakpoint symbol is placed in the margin of the Source pane next to the line in which you placed the cursor. If you try to set a breakpoint at an invalid location (for example, a comment line), the breakpoint will be set at the closest valid location. If you try to set a breakpoint in a method in which breakpoints cannot be used, a message will inform you that there are no valid locations in the method for breakpoints.

Set breakpoints in the following places:

- In the `buy()` method in the `TraderBean.java` EJB class (see Figure 2-20)
- In the `example()` method in the `Client` class, after creating the `Trader` and processing the buys and before processing the sells (see Figure 2-21)

Figure 2-20 Setting a breakpoint in the `TraderBean` class

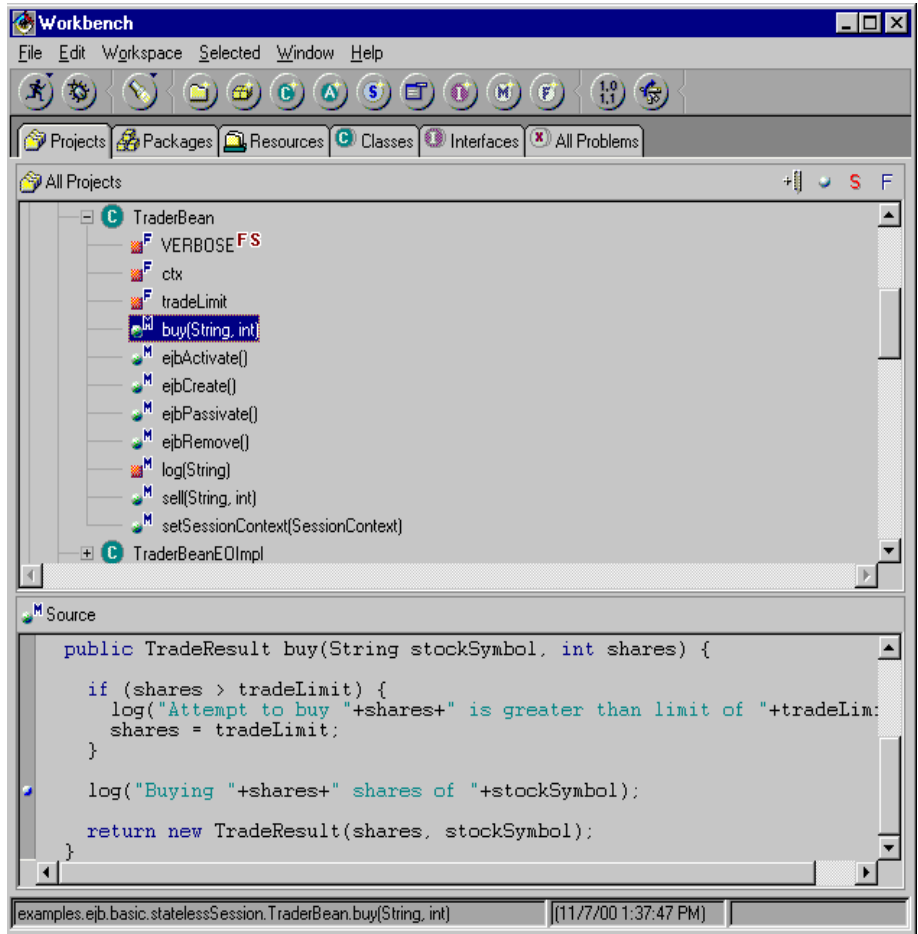
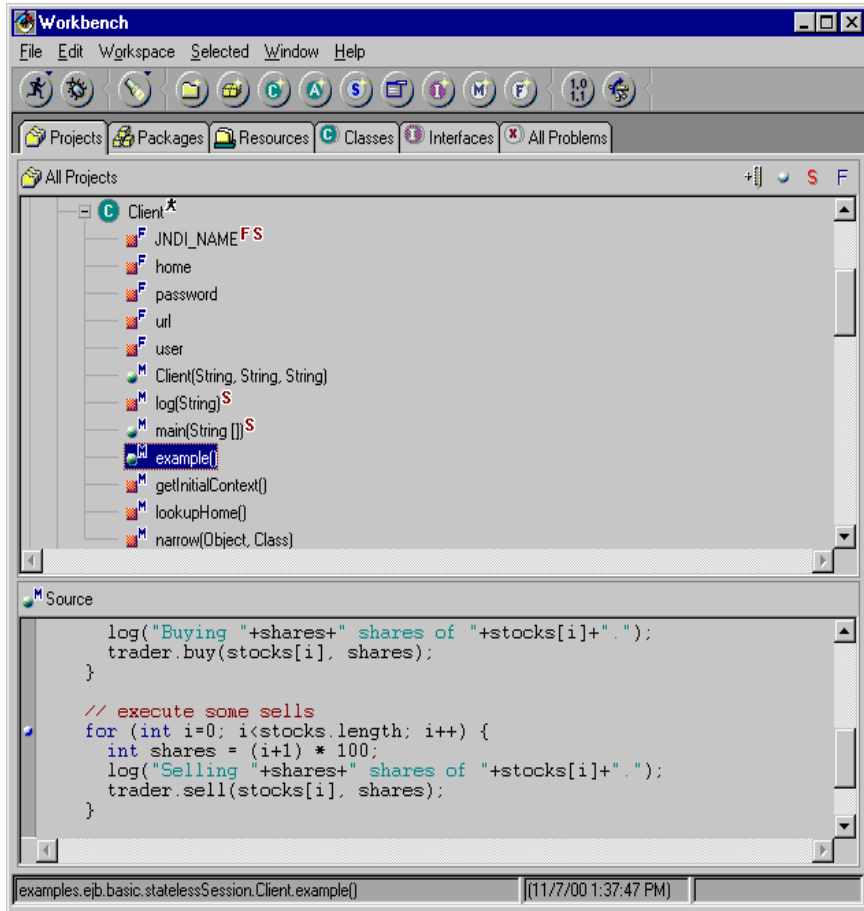


Figure 2-21 Setting a breakpoint in the client's `example()` method



Running the Client Application and the Server

You may choose to run the client either from within or outside of IBM VisualAge for Java, whichever is more convenient.

To start the debugging process:

1. Run the server (see “Verifying the EJB Deployment”).

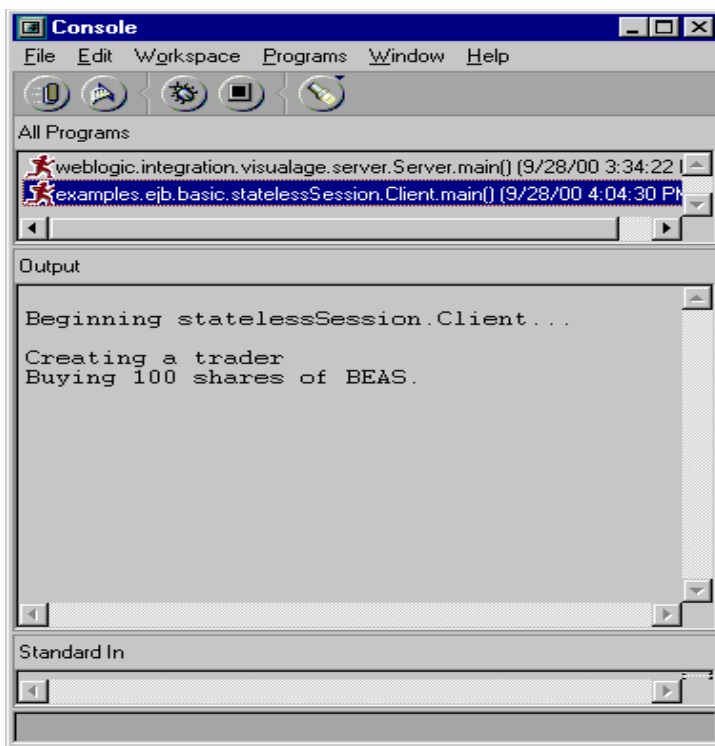
2. Run the client application (see “Running a Client Java Application that Communicates with the Deployed EJB”).

Note: IBM VisualAge for Java’s class loader allows you to modify and continue debugging server-side code without restarting WebLogic Server as long as you change only the content of the object methods. Changes to an object’s interface will require restarting the server.

Following the Processes in the Console and the Debugger

The console will show two processes running: the `Server` process and the `Client.main()` process, suspended at the first breakpoint.

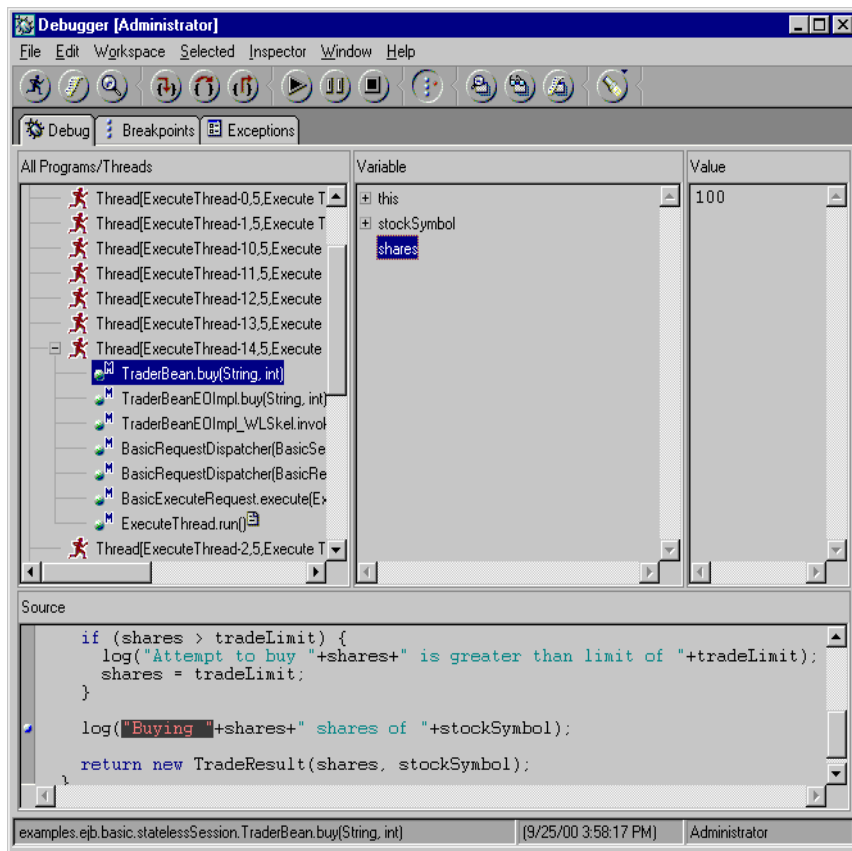
Figure 2-22 The console showing the suspension of the client’s messages



The Debug tab in the Debugger will display:

- All currently running threads, grouped by process.
- When a running thread has been suspended:
 - The methods in the thread
 - The visible variables and their values for the methods
 - The source code for the methods

Figure 2-23 The Debug tab in the Debugger

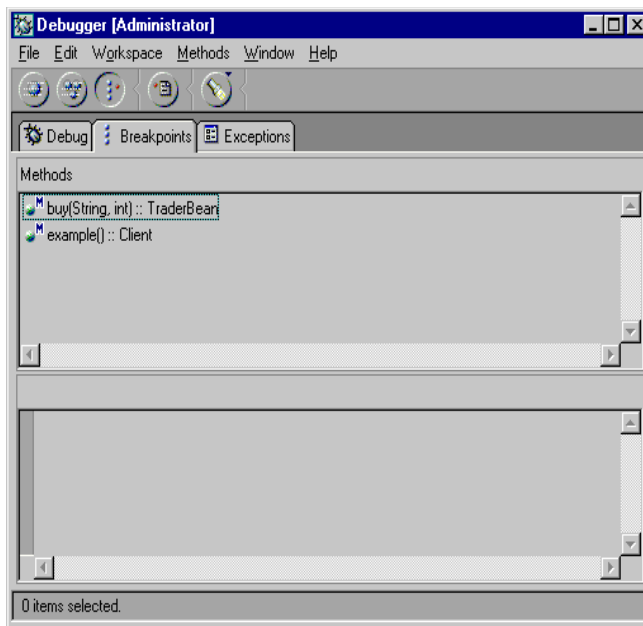


In this tab you can also watch, change or try different values for specified variables. For example, you can view and change the values of `shares`.

The Breakpoints tab in the Debugger will display:

- All methods in the workspace that have breakpoints set in them.

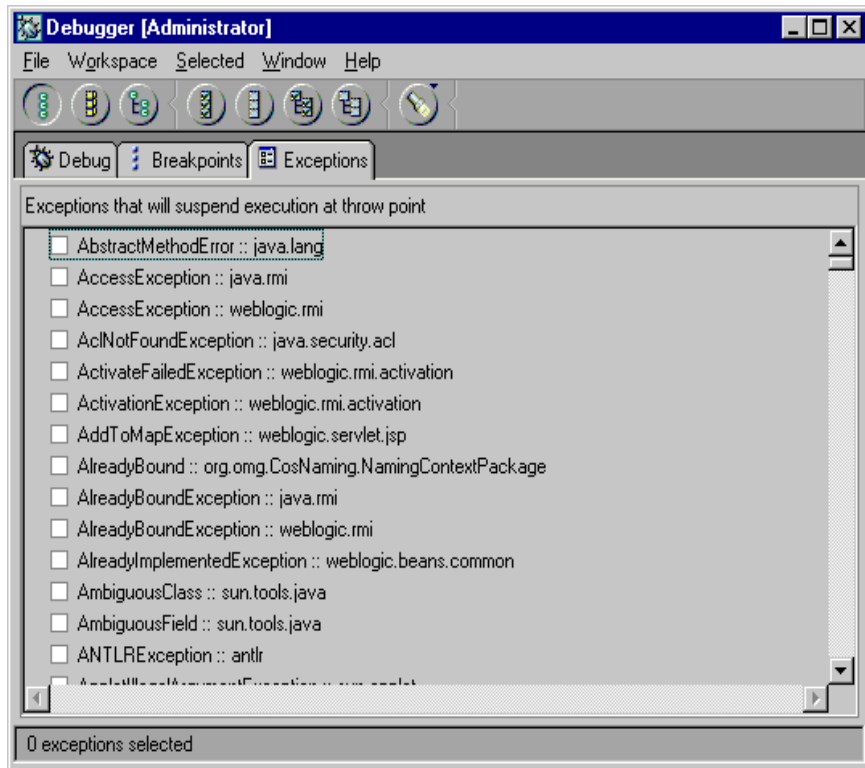
Figure 2-24 The Breakpoints tab in the Debugger



The Exceptions tab in the Debugger will display:

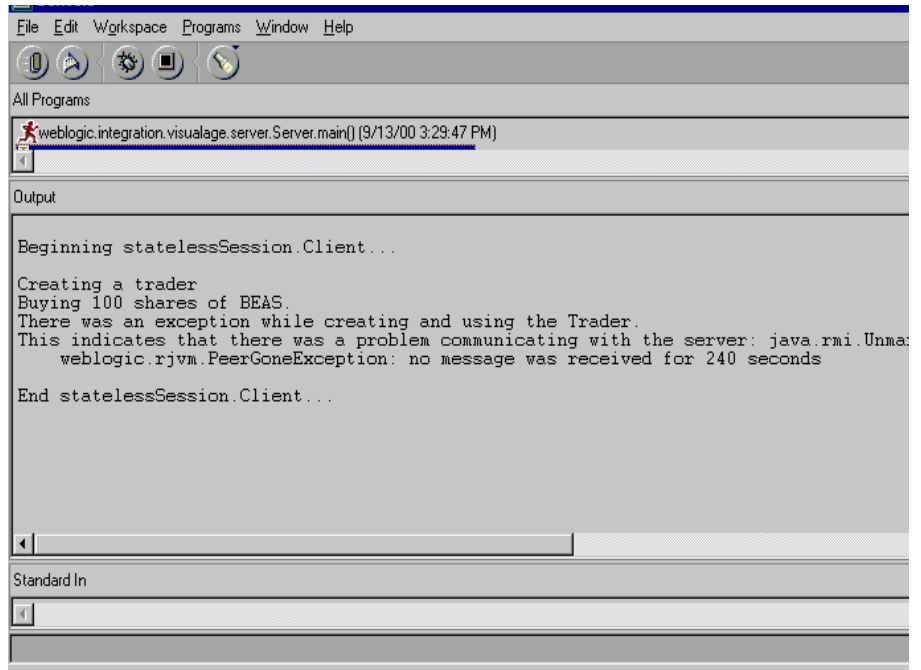
- The exceptions that will suspend the thread. (Currently none are set. To set them, check the corresponding checkboxes.)

Figure 2-25 The Exceptions tab in the Debugger




Note: The client application may time out if the server is stopped at a breakpoint for too long. (See Figure 2-26).

Figure 2-26 The console showing a time-out by the client application



Stepping through the code

To step through the code:


1. Go to the Debug tab in the Debugger.
2. Click the Step Over button  several times.

This runs the current statement, including all methods called within the statement, and stops before the next statement. If you step over a method that takes a significant amount of time to run, the string

```
/* Thread is currently stepping*/
```

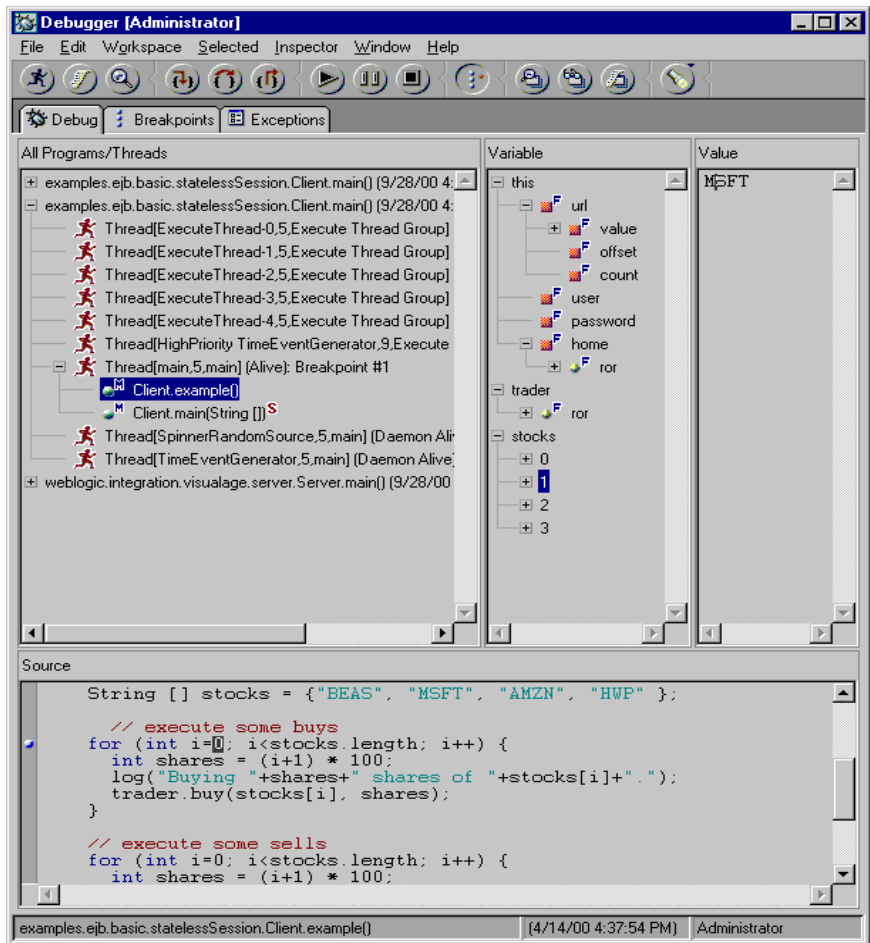
will be inserted into the Source pane. You may wait until it returns or resume the process.

You can also step into methods. Some methods, however, cannot be stepped into as the code for them is not visible to IBM VisualAge for Java. The classes they pertain to are part of the WebLogic libraries.

To resume the process, click the Resume button .

The process will continue until the next break point or until it terminates.

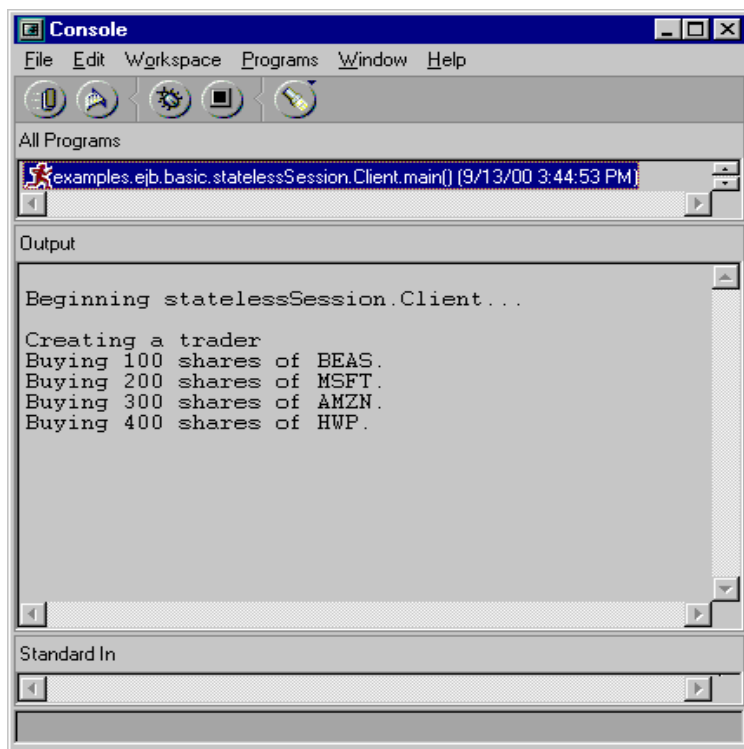
Figure 2-27 The Debugger showing a breakpoint in the Debug tab



In this example you will have to click Resume several times in order for the client to complete its task.

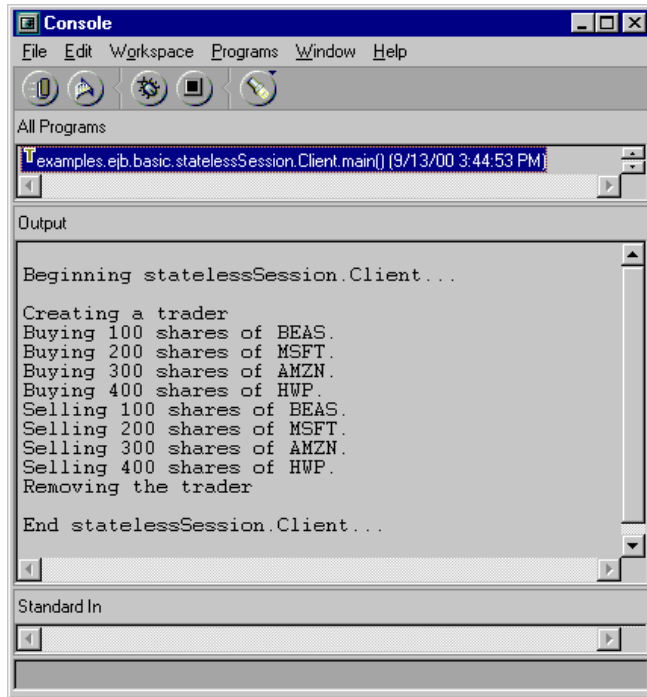
Figure 2-28 below shows the console after several *buys* have taken place:

Figure 2-28 The console showing the client's output



When the `Client` process terminates a `T` will be displayed next to the process in the All Programs pane of the console.

Figure 2-29 The `T` indicates that the process has terminated



For more information on the Integrated Debugger consult the IBM VisualAge for Java documentation.

Developing, Deploying, Using and Debugging a Container Managed Entity EJB

In this section you will build, deploy and debug a container managed entity Enterprise JavaBean called `AccountBean`.

Using this Enterprise JavaBean, the client application will:

1. Find or create 20 separate accounts
2. Display the balance for each account
3. Find all accounts with balances over \$5000
4. Find the first account with a balance of zero
5. Find any accounts with a null type
6. Remove all accounts

This application will demonstrate:

- Container-managed JDBC persistence (the code in the EJB never directly accesses the data storage)
- How to use BEA WebLogic Server's EJB finders to find both single and Enumerations of accounts
- How to use application-defined exceptions
- How to use the `is-modified-method-name` to reduce database access

A persistent storage for the entity EJB is required. We will use a database. The persistent storage is completely invisible to the client; the actual storage is handled automatically by the container and not by the EJB. All database properties, such as the login name and password, are defined in the connection pool within the `weblogic.properties` file.

The section will walk you through several steps that correspond to a typical EJB application development process:

1. Setting up the Oracle Database
2. Developing the EJB JAR
3. Configuring BEA WebLogic Server to Run the EJB
4. Verifying the EJB Deployment
5. Running a Client Java Application that Communicates with the Deployed EJB

In this example “Setting up the Oracle Database” must be done first.

“Configuring BEA WebLogic Server to Run the EJB” and “Verifying the EJB Deployment” can be done before “Developing the EJB JAR” because the example includes the *pre-built* JAR file installed in the correct location. However, “Developing the EJB JAR” has to be successfully completed before you can do “Running a Client Java Application that Communicates with the Deployed EJB” and “Debugging the Client Application and the Server Object” because these steps depend on having all the generated container classes in the workspace.

The code used in this section is based on the example `examples.ejb.basic.containerManaged` which is shipped with BEA Weblogic Server and adapted to work with another DataBase (Oracle) and an IDE (IBM VisualAge for Java). All the code elements (`.java` and `.xml`) for this example were included in the `WebLogic Examples` project when you installed the Integration Kit.

In general, you will need to adjust certain BEA WebLogic Server properties to match your setup. To deploy the EJB you will need to edit the property that begins with `weblogic.ejb.deploy` in the `weblogic.properties` file. This property is commented out in the default properties file; make sure that you uncomment all the lines of the property.

This section will not be as detailed as the previous section, “Developing, Deploying, Using, and Debugging a Stateless Session EJB”. For more details refer to the corresponding paragraphs in this previous section.

Note: If you change the BEA WebLogic Server installation root to another location *after* installing it, you must reconfigure the EJB tools using the Configure Tools tool (select Workspace→Tools→WebLogic Server Tools→Configure tools). (For more information refer to the Installation manual for BEA WebLogic Server) This tool associates the EJB tools with the new root directory of the BEA WebLogic Server distribution. This is necessary because the EJB tools depend on classes in the BEA WebLogic Server distribution that have not been imported into the IBM VisualAge for Java workspace, and because the Generate EJB JAR tool must know where to install the generated JARs. This tool was run as part of the installation process and does *not* need to be re-run unless you move the location of the BEA WebLogic Server distribution after installation. For more information see the *Installation Guide* for the Integration Kit.

Setting up the Oracle Database

The container managed entity EJB example uses the `demoPool` database connection pool. The BEA WebLogic Server distribution contains this same example, but it is configured to use a pre-configured Cloudscape database that is installed with BEA WebLogic Server. In order to run this example in IBM VisualAge for Java, you will need to create the example tables in another database, such as Oracle, and change the `demoPool` configuration in `weblogic.properties` to use this database. This section shows you how to do this.

Installing Oracle

1. Install both Oracle8i Standard Edition Release 2 Version 8.1.6 and Oracle Client (Administration Version).

During the installation set `Global DB name` to `Demo` and leave `DB system identifier(SID)` as `Demo`. By default, the DB will be registered as `Demo`

2. Edit the `PATH` by adding `WebLogic\bin\oci815_8`.

Testing the Connection to the Oracle database.

To test the connection to the Oracle database use the convenience program `dbping.exe` with the following arguments:

- *DBTYPE*: Use one of the following values:
 - ORACLE,
 - MSSQLSERVER4
 - INFORMIX4
- *USER*: A valid username for database login. Use the same values and format that you would use with `isql` (for SQL Server), `sqlplus` (for Oracle) or `DBACCESS` (for Informix).
- *PASS*: A valid password for the user. Use the same values and format that you would use with `isql`, `sqlplus` or `DBACCESS`.
- *DB@SERVER:PORT*: The name of the database. The format varies depending on the database and version. Use the same values and format that you would use with `isql`, `sqlplus` or `DBACCESS`. Type 4 drivers, such as `MSSQLServer4` and `Informix4`, need additional information to locate the server since they cannot access the environment.

In this example use the command:

```
C:\weblogic\bin>dbping ORACLE scott tiger demo
```

Listing 2-1 Output after running dbping

```
+++ WebLogic Native Layer for OCI 8.x (BETA-2)

**** Success!!! ****

You can connect to the database in your app using:

    Class.forName("weblogic.jdbc.oci.Driver").newInstance();

    java.sql.Connection conn =
DriverManager.getConnection("jdbc:weblogic:oracle:demo", "tiger");

**** or ****

    java.util.Properties props = new java.util.Properties();
    props.put("user",      "scott");
    props.put("password", "tiger");
    props.put("server",    "demo");
    Class.forName("weblogic.jdbc.oci.Driver").newInstance();
    java.sql.Connection conn =
        DriverManager.getConnection("jdbc:weblogic:oracle", props);

C:\>
```

Creating the Database Tables

In this example you will create the database tables in the Oracle server using the `utils.Schema` Java utility.

BEA WebLogic Server includes a Data-Definition Language (DDL) file for the examples database and a Schema tool to create a database from a DDL file.

To execute `utils.Schema`, your `CLASSPATH` must contain the `WebLogic/classes` directory.

The `utils.Schema` command uses the following syntax:

```
java utils.Schema url JDBC_driver [options] DDL_file
```

where:

- `url` is the database connection URL. This is a colon-separated URL as defined by the JDBC specification.
- `JDBC_driver` is the full package name of the JDBC Driver class.
- `options` can be:
 - Options like `-u username` or `-p password` which are used if the database requires a username and password.
 - The `-verbose` option which causes `utils.Schema` to echo the SQL commands as they are executed.
- `DDL_file` is the full pathname of the text file containing the SQL commands to execute. Lines beginning with pound signs (#) are comments. An SQL command can span several lines and is terminated with a semicolon (;).

To create a `utils.Schema` command that will create the database tables in an Oracle server named `demo`, with the username `scott` and password `tiger`:

1. Open a DOS prompt window.
2. Type the following single command:

```
c:\weblogic\jre1_2\jre\bin\java -classpath  
c:\weblogic\jre1_2\jre\lib\rt.jar;c:\weblogic\classes;  
c:\weblogic\license utils.Schema jdbc:weblogic:oracle:demo  
weblogic.jdbc.oci.Driver -u scott -p tiger -verbose  
"c:\weblogic\examples\utils\ddl\demo.ddl"
```

If you are not using JDK1.2.2, the Java executable and `rt.jar` must be replaced.

Note: You must include the double quotes around the path of the DDL file.

On execution of this command you will see the messages shown in Listing 2-2 in the DOS window.

Listing 2-2 Messages shown in DOS window as DB entries are created

```
C:\>c:\weblogic\jre1_2\jre\bin\java -classpath
c:\weblogic\jre1_21\jre\lib\rt.jar;c:\weblogic\classes;c:\weblogic
\license utils.Schema jdbc:weblogic:oracle:demo
weblogic.jdbc.oci.Driver -u scott -p tiger -verbose
"c:\weblogic\examples\utils\ddl\demo.ddl"

utils.Schema will use these parameters:

url: jdbc:weblogic:oracle:demo

driver: weblogic.jdbc.oci.Driver

dbserver: null

user: scott

password: tiger

SQL file: c:\weblogic\examples\utils\ddl\demo.ddl

+++ WebLogic Native Layer for OCI 8.x (BETA-2)

DROP TABLE ejbAccounts

CREATE TABLE ejbAccounts (id varchar(15), bal float, type
varchar(15))

DROP TABLE idGenerator

CREATE TABLE idGenerator (tablename varchar(32), maxkey int)

DROP TABLE CUSTOMER

CREATE TABLE customer(custid int not null,name varchar(30),address
varchar(30), city varchar(30), state varchar(2), zip varchar(5),
area varchar(3), phone varchar(8))

insert into customer values(100,'Jackson','100 First
St.','Pleasantville','CA','95404','707','555-1234')

insert into customer values(101,'Elliott','Arbor Lane, #3','Centre
```

```
Town','CA','96539','415','787-5467')

insert into customer values

(102,'Avery','14Main','Arthur','CA','97675','510','834-7476')

DROP TABLE emp

CREATE TABLE emp(empno int not null, ename varchar(10), job
varchar(9), mgr int, hiredate date, sal float, comm float, deptno
int)

create unique index empno on emp(empno)

insert into emp values

(7369,'SMITH','CLERK',7902,DATE'1980-12-17',800,NULL,20)

insert into emp

values(7499,'ALLEN','SALESMAN',7698,DATE'1981-02-20',1600,300,30)

insert into emp

values(7521,'WARD','SALESMAN',7698,DATE'1981-02-22',1250,500,30)

insert into emp

values(7566,'JONES','MANAGER',7839,DATE'1981-04-02',2975,NULL,20)

insert into emp

values(7654,'MARTIN','SALESMAN',7698,DATE'1981-09-28',1250,1400,3)

insert into emp

values(7698,'BLAKE','MANAGER',7839,DATE'1981-05-1',2850,NULL,30)

insert into emp

values(7782,'CLARK','MANAGER',7839,DATE'1981-06-9',2450,NULL,10)

insert into emp

values(7788,'SCOTT','ANALYST',7566,DATE'1981-06-9',3000,NULL,20)

insert into emp

values(7839,'KING','PRESIDENT',NULL,DATE'1981-11-17',5000,NULL,10
)

insert into emp
```

```
values(7844,'TURNER','SALESMAN',7698,DATE'1981-09-8',1500,0,30)
insert into emp
values(7876,'ADAMS','CLERK',7788,DATE'1981-06-9',1100,NULL,20)
insert into emp
values(7900,'JAMES','CLERK',7698,DATE'1981-12-3',950,NULL,30)
insert into emp
values(7902,'FORD','ANALYST',7566,DATE'1981-12-3',3000,NULL,20)
insert into emp
values(7934,'MILLER','CLERK',7782,DATE'1982-01-23',1300,NULL,10)
DROP TABLE dept
create table dept(deptno int not null, dname varchar(10), loc
varchar(9))
insert into dept values(10,'ACCOUNTING','NEW YORK')
insert into dept values(20,'RESEARCH','DALLAS')
insert into dept values(30,'SALES','CHICAGO')
insert into dept values(40,'OPERATIONS','BOSTON')
DROP TABLE finderEnum
create table finderEnum(id varchar(10), bal float not null)
insert into finderEnum values('PK1', 0)
insert into finderEnum values('PK2', 0)
insert into finderEnum values('PK3', 0)
insert into finderEnum values('PK4', 0).2 Developing, Deploying,
Using and Debugging EJBs
insert into finderEnum values('PK5', 0)
insert into finderEnum values('PK6', 1)
insert into finderEnum values('PK7', 1)
insert into finderEnum values('PK8', 1)
insert into finderEnum values('PK9', 1)
```



```
insert into finderEnum values('PK10', 1)

DROP TABLE StockTable

create table StockTable(symbol varchar(10), price float, yearHigh
float, yearLow float, volume int)
```

Developing the EJB JAR

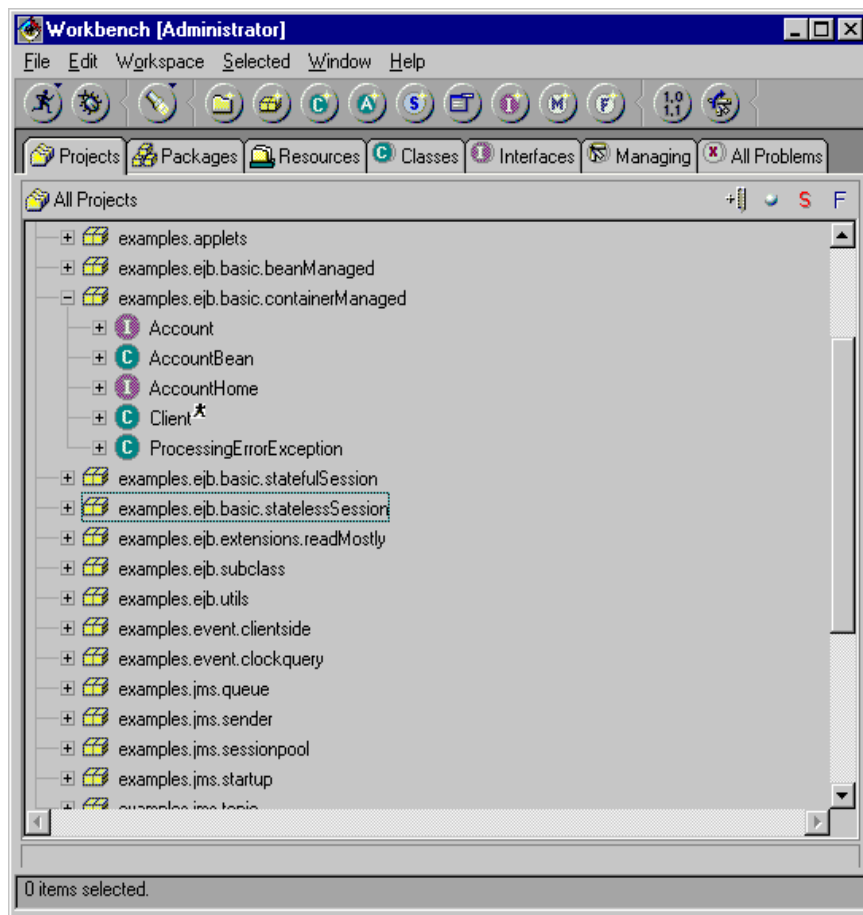
All the code elements for the container managed example have already been developed and are installed in the `WebLogic Examples` project. We will look at these elements in the `ContainerManaged` package and then build the deployable JAR file. A *pre-built* EJB JAR file is already installed in the directory `WebLogic\myserver` (where *WebLogic* is the installation directory of BEA WebLogic Server), but in order to use and debug the EJB you need to generate the container classes.

Looking at the ContainerManaged Package

To examine the `ContainerManaged` package in the workspace:

1. Start IBM VisualAge for Java.
2. In the Projects tab of the Workbench, select the package `examples.ejb.basic.containerManaged` in the project `WebLogic Examples`.

Figure 2-30 The ContainerManaged package in the Projects tab



Notice that the `examples.ejb.basic.containerManaged` package already contains:

- The client runnable class: `Client`
- The server EJB class: `AccountBean`
- The exception class: `ProcessingErrorException`
- The EJB interfaces: `Account`, `Account PS`, and `AccountHome`

During the Integration Kit installation the Deployment Descriptors were included in the project's resource folder:

```
VisualAge\IDE\project_resources\WebLogic Examples\examples\  
ejb\basic\containerManaged
```

where *VisualAge* is the installation directory for IBM VisualAge for Java (in this example, C:\Program Files\IBM\VisualAge for Java).

This resource folder contains the following files:

- `ejb-jar.xml`
- `weblogic-ejb-jar.xml`
- `weblogic-cmp-rdbms-jar.xml`

In a real development situation, you will:

- create the Java and XML files yourself using an IDE, such as IBM VisualAge for Java. (It would be a good exercise to walk through the code, so that you understand what is required to build an EJB.)
- create a project (`WebLogic Examples`) and a package (`examples.ejb.basic.statelessSession`) inside IBM VisualAge for Java workspace to develop your EJB and Client. Using the IBM VisualAge for Java Import utility, import the Deployment Descriptors into the corresponding package (`examples.ejb.basic.statelessSession`).

Warning: Do not modify any files in the IBM VisualAge for Java
`VisualAge\IDE\project_resources` file system directory tree.

The deployable JAR file (`ejb_basic_containerManaged.jar`) is already built and installed in the directory `WebLogic\myserver` (where *WebLogic* is the installation directory for BEA WebLogic Server).

Starting the Build

For the client application to be able to communicate with the deployed EJB, you need to have all the generated container classes in the workspace, so you must build the EJB JAR.

To start the build of the EJB JAR:

1. Select the `examples.ejb.basic.containerManaged` package in the Project WebLogic Examples.
2. From the menu bar, select Selected→WebLogic Server Tools→Generate EJB JAR.

Note: To use the Generate EJB JAR tool, users of the Enterprise Edition of IBM VisualAge for Java will first need to create an open edition of the EJB package. To create an open edition, select the package and then select the menu items Selected→Manage→Create Open Edition. If you are not using an open edition of the package, the Enterprise Edition will report an error message.

The build process involves several steps, each of which will be announced by a Generating EJB message as it happens.

Naming the EJB JAR File

If it is the first time that you are building the package, you will be prompted for an output JAR filename.

Change the default settings in the window to:

- Location: *WebLogic\myserver*

where *WebLogic* is the actual BEA WebLogic Server installation directory, such as `C:\weblogic`

- Name: `ejb_basic_containermanaged.jar`

Generating the Undeployable JAR

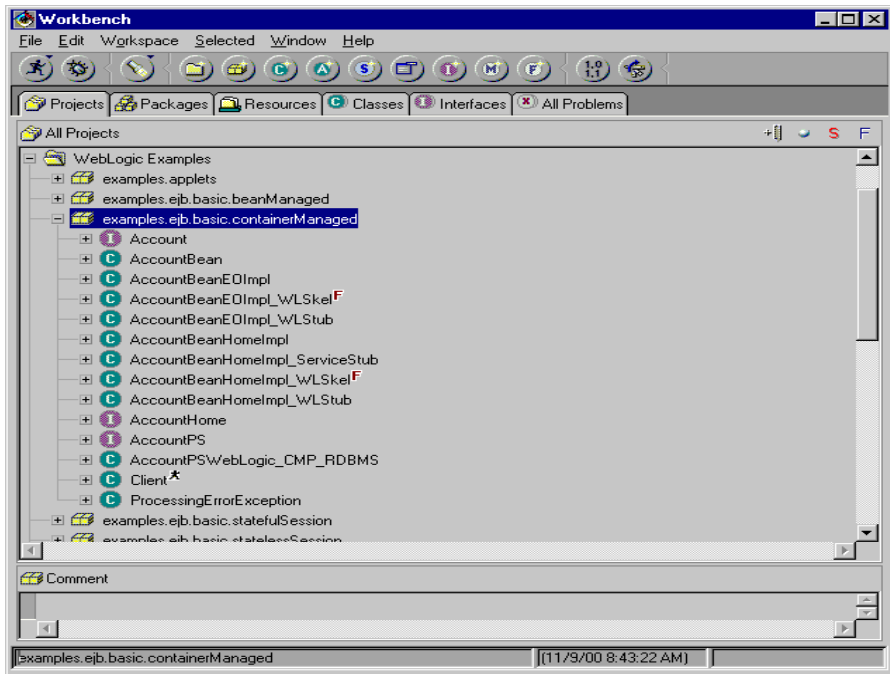
Using the compiled classes in the package (EJB class, Interfaces, Client) and the Deployment Descriptors, the build process will generate an undeployable JAR.

The name of the undeployable JAR file will be created by adding the prefix `std_` to the JAR name provided above. In this example the undeployable JAR will be called `std_ejb_basic_containerManaged.jar`. The file will be installed in the specified directory, in this example *WebLogic/myserver* where *WebLogic* is the installation directory for BEA WebLogic Server (in this example, `C:\weblogic`).

Generating the Container Classes

The build process will generate the container classes and add them, one by one, to the package. A Generating EJB message will inform you of the generation.

Figure 2-31 The package showing the generated container classes



Generating the EJB JAR

Using the compiled classes in the package (EJB class, Interfaces, Client), the Deployment Descriptors and the newly generated container classes, the build process will generate a deployable JAR. In this example the generated JAR will be called `ejb_basic_containerManaged.jar`. The file will be installed in the specified directory, `WebLogic/myserver` where `WebLogic` is the installation directory for BEA WebLogic Server (in this example, `C:\weblogic`).

When the build process is complete, you will get an EJB Generated confirmation message.

Figure 2-32 The EJB Generated confirmation message



Configuring BEA WebLogic Server to Run the EJB

In this example the EJB communicates with a database. After setting up the database driver and tables (see “Setting up the Oracle Database” on page 2-42) you will have to configure the server’s properties in the `weblogic.properties` file to allow access to the database and pool, and the JAR file. For Container Managed persistence, you have to add the path to the JAR file to the server’s classpath in order to run the application and start the server from inside IBM VisualAge for Java.

Note: If you configure server properties in the `weblogic.properties` file while the server is running in IBM VisualAge for Java you will need to stop the server and restart it.

To successfully deploy and use an EJB which communicates with a database:

1. Configure BEA WebLogic Server for the pool and DataSource of your EJB.

Make the necessary configuration changes in the `weblogic.properties` file by uncommenting the following lines and editing the name of the pool by changing it from `oraclePool` to `demoPool` as shown below in Listing 2-3. The `weblogic.properties` file is located in the root installation of the BEA WebLogic Server (in this example, `c:\weblogic`).

Listing 2-3 The demoPool entry in the Properties file

```
weblogic.jdbc.connectionPool.demoPool=\
url=jdbc:weblogic:oracle,\
driver=weblogic.jdbc.oci.Driver,\
loginDelaySecs=1,\
initialCapacity=4,\
maxCapacity=10,\
capacityIncrement=2,\
allowShrinking=true,\
shrinkPeriodMins=15,\
refreshMinutes=10,\
testTable=dual,\
props=user=SCOTT;password=tiger;server=demo
weblogic.jdbc.TXDataSource.weblogic.jdbc.jts.demoPool=demoPool
weblogic.allow.reserve.weblogic.jdbc.connectionPool.demoPool=ever
yone
```

2. Add the path to the JAR file to the `weblogic.ejb.deploy` property in the `weblogic.properties` file. A commented-out version of this path can be found in the `weblogic.ejb.deploy` property. You will need to uncomment and adjust the property depending on which EJBs you are building and deploying, or if the location of the files differs from the installed location. For this example you will have to uncomment:

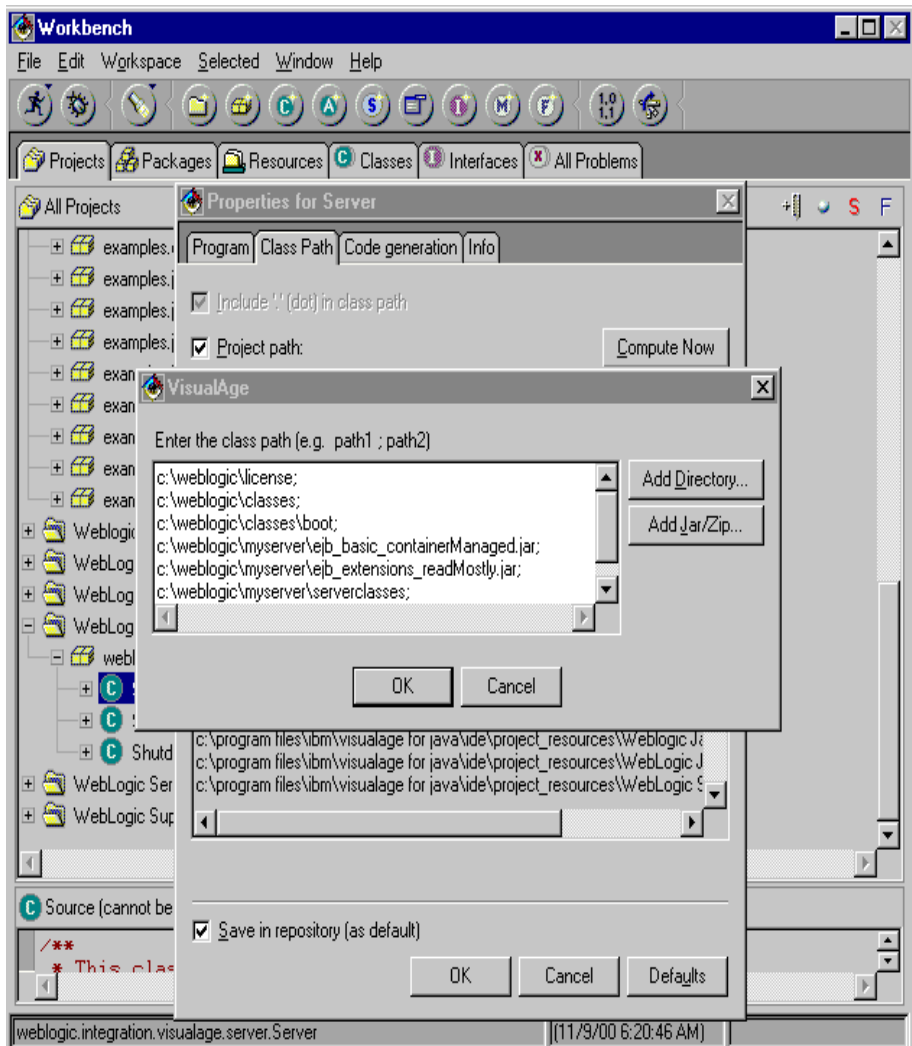
```
weblogic.ejb.deploy=\
    C:/weblogic/myserver/ejb_basic_containerManaged.jar
```

For Container Managed persistence, you must add the path to the EJB JAR to the BEA WebLogic Server class in order to run the application and start the server from inside IBM VisualAge for Java.

In this example it was done by the Integration Kit Installer, but to verify, do the following:

1. In the Projects tab of the Workbench, expand the `WebLogic Server` project, then the `weblogic.integration.visualage.server` package.
2. Select the `Server` runnable class in the `WebLogic Server` project.
3. Right-click on the `Server` class and select Properties from the context menu.
4. In the Properties window select the ClassPath tab.
5. On the Class Path tab, click the Edit button for the Extra Directories Path field.
6. In the IBM VisualAge for Java window click on Add JAR/Zip and Browse for the JAR you need,
`C:\weblogic\myserver\ejb_basic_containerManaged.jar` in this example.
7. Click OK on all the popped up windows.

Figure 2-33 Adding the JAR file to the server's classpath



Verifying the EJB Deployment

To verify if the EJB deploys correctly you have to start the server and look at the messages displayed on the console or the WebLogic console.

Starting the WebLogic Server

Note: If running Oracle, the minimum amount of memory required for running the server is 256MB.

To start the WebLogic server:

1. Start IBM VisualAge for Java.
2. In the Projects tab of the Workbench select WebLogic Server.
3. Do one of the following:
 - Click the Run button.
 - Right-click on the webLogic Server project and select Run→Run main.
 - Double-click on the WebLogic Server project. This will open a separate Weblogic Server window containing just the `weblogic.integration.visualage.server` package. In the Weblogic Server window click the Run button, or right-click on WebLogic Server and select Run→Run main.

While the server is running, a WebLogic Server message will be displayed that can be used to shut down the server.

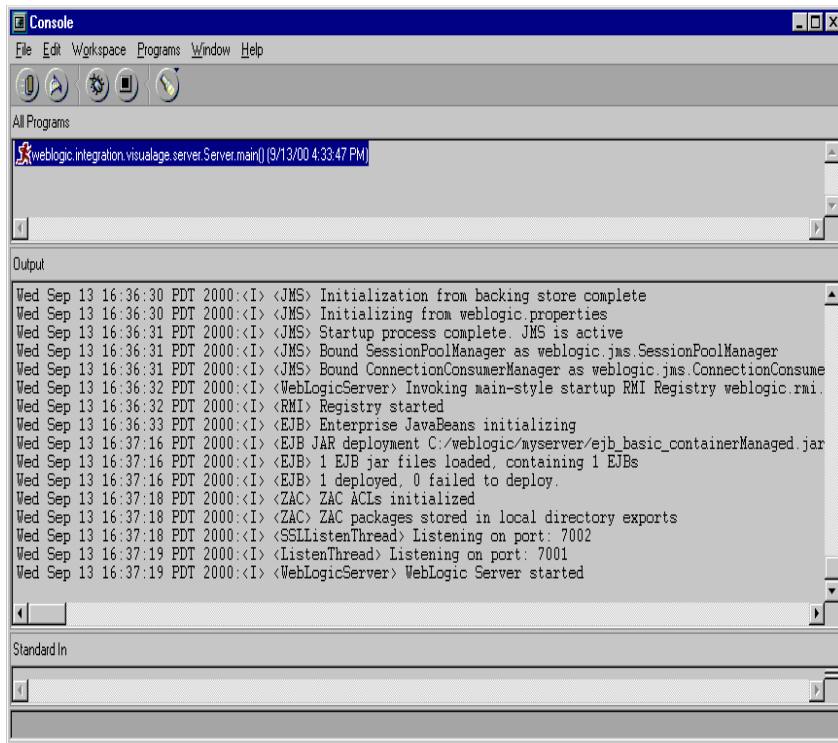
Figure 2-34 The Weblogic Server message



Checking the Server Messages on the consoles

To check whether the server has started correctly and whether the EJB has been deployed correctly you can either check the messages displayed on the VisualAge console or open the *Weblogic* console attached to the BEA WebLogic Server, and examine the EJB under Distributed Objects.

Figure 2-35 The console showing messages from the server



Running a Client Java Application that Communicates with the Deployed EJB

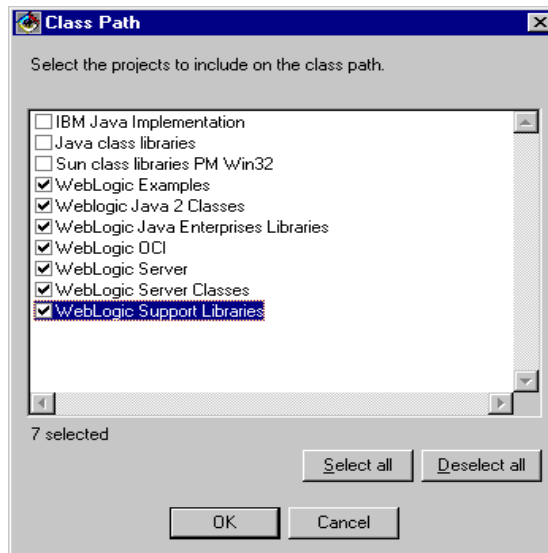
Before running the client application you must verify that the Integration Kit has provided the complete classpath for the application. You can also set Command-line parameters. The output from the client application will appear on the console.

Verifying the Classpath

To verify the complete classpath for the client application:

1. Right-click on the `Client` runnable class in the `examples.ejb.basic.containerManaged` package in the project `WebLogic Examples`.
2. Select `Properties` from the context menu to open the `Properties` window.
3. In the `Class Path` tab, click the `Edit` button to the right of the `Project Path` field to open the `Class Path` window.

Figure 2-36 The Class Path window



4. Verify that there are checkmarks in all the boxes corresponding to the BEA WebLogic Server projects that were added to the IBM VisualAge for Java workspace by the Integration Kit's Installer. If any of these projects are not checked, check them.
5. Click OK.

The path to each of the checked projects is displayed in the Complete Class Path on the Class Path tab of the Properties window.

Using Command-Line Parameters

There are three command-line parameters. The first parameter (*url*) gets a default argument and only needs to be changed if the default settings are not being used. The other two parameters are optional.

Command-line parameters are interpreted in this order:

1. *url*: URL of server (default such as `t3://localhost:7001`)
2. *user*: User name (default null)
3. *password*: User password (default null)

To edit the command-line arguments:

1. In the Properties window select the Program tab.
2. Enter the arguments in the Command Line Arguments text field.
3. Press OK.

Note: If you are not running the BEA WebLogic Server with its default settings, you will have to supply the command-line parameter:

```
t3://WebLogicURL:Port
```

where:

WebLogicURL is the domain address of the BEA WebLogic Server

Port is the port that is listening for connections
(`weblogic.system.ListenPort`)

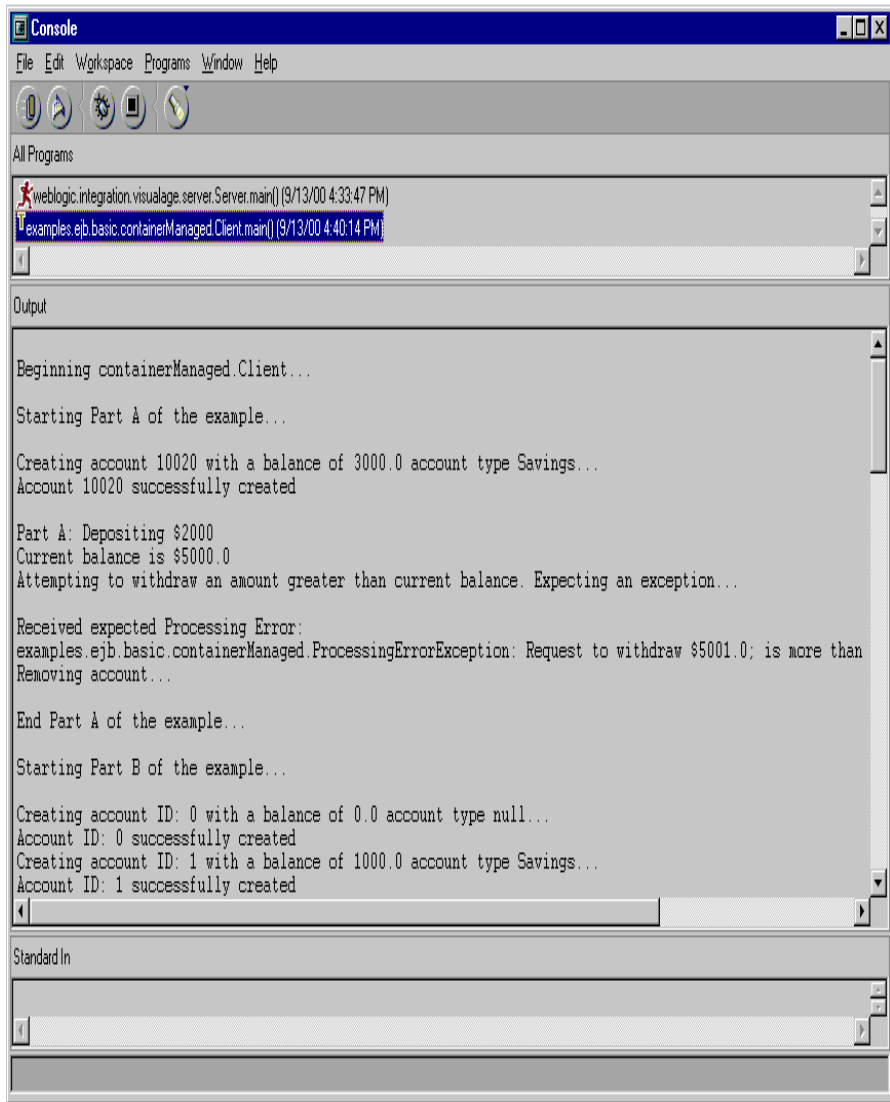
Running the Client Application

To run the client do one of the following:

1. Select the `Client` class in the `examples.ejb.basic.containerManaged` package and click on the Run button.
2. Right-click on the `Client` class in the `examples.ejb.basic.containerManaged` package and select Run→Run main.

When running the Client example, you should get output similar to this from the client application:

Figure 2-37 The VisualAge console showing messages from the client



Listing 2-4 shows the complete output to the VisualAge console:

Listing 2-4 Complete output to the VisualAge console

```
Beginning containerManaged.Client...
Starting Part A of the example...
Creating account 10020 with a balance of 3000.0 account type
Savings...
Account 10020 successfully created
Part A: Depositing $2000
Current balance is $5000.0
Attempting to withdraw an amount greater than current balance.
Expecting an exception...
Received expected Processing Error:
examples.ejb.basic.containerManaged.ProcessingErrorException:
Request to withdraw $5001.0; is more than balance $5000.0 in account
10020
Removing account...
End Part A of the example...

Starting Part B of the example...
Creating account ID: 0 with a balance of 0.0 account type null...
Account ID: 0 successfully created
Creating account ID: 1 with a balance of 1000.0 account type
Savings...
Account ID: 1 successfully created
Creating account ID: 2 with a balance of 2000.0 account type
Savings...
Account ID: 2 successfully created
Creating account ID: 3 with a balance of 3000.0 account type
Savings...
```


Account ID: 3 successfully created

Creating account ID: 4 with a balance of 4000.0 account type Savings...

Account ID: 4 successfully created

Creating account ID: 5 with a balance of 5000.0 account type null...

Account ID: 5 successfully created

Creating account ID: 6 with a balance of 6000.0 account type Savings...

Account ID: 6 successfully created

Creating account ID: 7 with a balance of 7000.0 account type Savings...

Account ID: 7 successfully created

Creating account ID: 8 with a balance of 8000.0 account type Savings...

Account ID: 8 successfully created

Creating account ID: 9 with a balance of 9000.0 account type Savings...

Account ID: 9 successfully created

Creating account ID: 10 with a balance of 10000.0 account type null...

Account ID: 10 successfully created

Creating account ID: 11 with a balance of 11000.0 account type Savings...

Account ID: 11 successfully created

Creating account ID: 12 with a balance of 12000.0 account type Savings...

Account ID: 12 successfully created

Creating account ID: 13 with a balance of 13000.0 account type Savings...

Account ID: 13 successfully created

Creating account ID: 14 with a balance of 14000.0 account type Savings...

```
Account ID: 14 successfully created

Creating account ID: 15 with a balance of 15000.0 account type
null...

Account ID: 15 successfully created

Creating account ID: 16 with a balance of 16000.0 account type
Savings...

Account ID: 16 successfully created

Creating account ID: 17 with a balance of 17000.0 account type
Savings...

Account ID: 17 successfully created

Creating account ID: 18 with a balance of 18000.0 account type
Savings...

Account ID: 18 successfully created

Creating account ID: 19 with a balance of 19000.0 account type
Savings...

Account ID: 19 successfully created

Account: :ID: 0 has a balance of 0.0
Account: :ID: 1 has a balance of 1000.0
Account: :ID: 2 has a balance of 2000.0
Account: :ID: 3 has a balance of 3000.0
Account: :ID: 4 has a balance of 4000.0
Account: :ID: 5 has a balance of 5000.0
Account: :ID: 6 has a balance of 6000.0
Account: :ID: 7 has a balance of 7000.0
Account: :ID: 8 has a balance of 8000.0
Account: :ID: 9 has a balance of 9000.0
Account: :ID: 10 has a balance of 10000.0
Account: :ID: 11 has a balance of 11000.0
Account: :ID: 12 has a balance of 12000.0
Account: :ID: 13 has a balance of 13000.0
```

Account: :ID: 14 has a balance of 14000.0

Account: :ID: 15 has a balance of 15000.0

Account: :ID: 16 has a balance of 16000.0

Account: :ID: 17 has a balance of 17000.0

Account: :ID: 18 has a balance of 18000.0

Account: :ID: 19 has a balance of 19000.0

Querying for accounts with a balance greater than 5000.0...

Account ID: 6; balance is \$6000.0

Account ID: 7; balance is \$7000.0

Account ID: 8; balance is \$8000.0

Account ID: 9; balance is \$9000.0

Account ID: 10; balance is \$10000.0

Account ID: 11; balance is \$11000.0

Account ID: 12; balance is \$12000.0

Account ID: 13; balance is \$13000.0

Account ID: 14; balance is \$14000.0

Account ID: 15; balance is \$15000.0

Account ID: 16; balance is \$16000.0

Account ID: 17; balance is \$17000.0

Account ID: 18; balance is \$18000.0

Account ID: 19; balance is \$19000.0

Querying for an account with zero balance...

Account ID: 0; balance is zero

Querying for accounts with a null account type

```
Account ID: 0; account type is null
Account ID: 5; account type is null
Account ID: 10; account type is null
Account ID: 15; account type is null
Removing beans...
End Part B of the example...
End containerManaged.Client...
```

Debugging the Client Application and the Server Object

For explanations about debugging the client and server, see “Debugging the Client Application and the Server Object” in the section “Developing, Deploying, Using, and Debugging a Stateless Session EJB” above.

3 Combining EJB with JMS and Servlet Technologies

Topics discussed in this chapter include:

- Configuring BEA WebLogic Server for JMS
- Configuring BEA WebLogic Server for the Servlet
- Running the Client Application
- Calling the Servlet from a Web Browser
- Exporting the Classes to the Production BEA WebLogic Server

Overview

Java Message Service (JMS) allows Java programs that share a messaging system to exchange messages. A messaging system accepts messages from *producer* clients and delivers them to *consumer* clients.

BEA WebLogic Server JMS implements the JavaSoft JMS specification, version 1.0.1.

BEA WebLogic Server JMS includes a fully-featured messaging system. This system can be configured by setting properties in the `weblogic.properties` file or in the BEA WebLogic Server console, or by setting them programmatically using the JMS interfaces.

You can use BEA WebLogic Server JMS with the other BEA WebLogic Server APIs and facilities, such as Enterprise JavaBeans (EJBs), JDBC connection pools, Servlets, and RMI. JMS operations can participate in transactions with other Java APIs that use the Java Transaction API.

In this chapter we will look at an application that combines three technologies:

- EJB (using the `StatelessSession` EJB created in the previous sections)
- JMS
- Servlet

In this application the Java servlet `TraderServlet` will send *buy* and *sell* messages to a JMS Topic. The `TraderReceive` client will receive the messages and invoke an EJB to process them.

To use this application you must have successfully built and deployed the `StatelessSession` EJB in the example in the previous chapter.

Configuring BEA WebLogic Server for JMS

The BEA WebLogic Server already has built-in defaults for JMS so you can use some of the default JMS features without doing any further special configuration. However, to use persistent messages, durable subscriptions, or to set up custom JMS applications, you have to do some or all of the following configuration tasks (see the BEA WebLogic Server *Administrator Guide* for details):

- Creating a Database for JMS
- Creating Database Tables for Transacted, Durable Subscribers
- Defining a JDBC Connection Pool for the JMS Database

- Defining JMS ConnectionFactories
- Defining JMS Topics and Queues

Once the necessary configuration has been done, JMS clients can begin sending and receiving messages through the JMS API.

In this section you will configure the server for JMS as required by the application in this chapter. You will create a JMS database, but the application does not require tables for transacted, durable subscribers. When you combine JMS and EJBs in an application that requires operations on both within the same transaction, you have to set up the application to use the same database connection pool. You will define the connection pool and the ConnectionFactory. You will define one Topic.

The `weblogic.properties` file included with the BEA WebLogic Server already has a section for BEA WebLogic Server JMS properties. In order to use this JMS application, you will have to include (uncomment) some of these properties before starting the BEA WebLogic Server.

Creating a Database for JMS

In this application we will use an Oracle database for JMS. The `WebLogic/classes/weblogic/jms/ddl` directory contains JMS DDL files for various types of databases. To use a different type of database you can copy and edit one of these files.

The JMS database in this application will contain five system tables used internally by JMS. To create the database tables use the BEA WebLogic Server console or the `utils.Schema` utility. For information about the `utils.Schema` utility see “Setting up the Oracle Database” on page 2-42.

The following example code shows a `utils.Schema` command that creates the JMS tables in an Oracle server named `demo`, with the username `scott` and password `tiger`:

```
c:\weblogic\jre1_2\jre\bin\java -classpathc:\weblogic\jre1_2\jre
\lib\rt.jar;c:\weblogic\classes;c:\weblogic\licenseutils.Schema
jdbc:weblogic:oracle:demoweblogic.jdbc.oci.Driver -u scott -p
tiger-verbose"c:\weblogic\classes\weblogic\jms\ddl\
jms_oracle.ddl"
```

Note: If you are using a different Java Virtual Machine, replace the `java` command and `rt.jar` with the new `java` command and `classpath`.

Listing 3-1 is the output from this command.

Listing 3-1 Output from interpreter

`utils.Schema` will use these parameters:

```
url: jdbc:weblogic:oracle:demo
driver: weblogic.jdbc.oci.Driver
dbserver: null
user: scott
password: tiger
SQL file: c:\weblogic\classes\weblogic\jms\ddl\jms_oracle.ddl
+++ WebLogic Native Layer for OCI 8.x (BETA-2)
DROP TABLE JMSDestination
```



```
java.sql.SQLException: ORA-00942: table or view does not exist
```

```
SQL Error Code: 942
```

```
SQL State:
```

```
DROP TABLE JMSConsumer
```

```
java.sql.SQLException: ORA-00942: table or view does not exist
```

```
SQL Error Code: 942
```

```
SQL State:
```

```
DROP TABLE JMSMessage
```

```
java.sql.SQLException: ORA-00942: table or view does not exist
```

```
SQL Error Code: 942
```

```
SQL State:
```

```
DROP TABLE JMSMessageQueue
```

```
java.sql.SQLException: ORA-00942: table or view does not exist
```

```
SQL Error Code: 942
```

```
SQL State:
```

```
DROP TABLE JMSTableId
```

```
java.sql.SQLException: ORA-00942: table or view does not exist
```

```
SQL Error Code: 942
```

```
SQL State:
```

```
CREATE TABLE JMSDestination (destId int, destType int, destName  
varchar(60))
```

```
CREATE TABLE JMSConsumer (consumerId int, clientName varchar(40),  
consumerName
```

```
varchar(40), destId int, selector varchar(100), noLocal  
NUMBER(1))
```

```
CREATE TABLE JMSMessage (messageId NUMBER(12), timeToLive int,  
destId int, state NUMBER(1), message LONG RAW)
```

```
CREATE TABLE JMSMessageQueue (consumerId int, messageId NUMBER(12),  
state int)
```

```
CREATE TABLE JMSTableId (tableName varchar(16), tableId NUMBER(12))
```

3 *Combining EJB with JMS and Servlet Technologies*

```
CREATE INDEX MSG_X ON JMSMessage (messageId)

CREATE INDEX MSGQ_X ON JMSMessageQueue (messageId)

INSERT INTO JMSTableId (tableName, tableId) VALUES
('JMSDestination', 1)

INSERT INTO JMSTableId (tableName, tableId) VALUES ('JMSConsumer',
1)

INSERT INTO JMSTableId (tableName, tableId) VALUES ('JMSMessage',
1)

INSERT INTO JMSTableId (tableName, tableId) VALUES ('JMSVersion',
500)

COMMIT
```

Defining a JDBC Connection Pool for the JMS Database

If you are using persistent messages with JMS, you have to define a JDBC connection pool in the `weblogic.properties` file to provide access to the JMS database.

To define the connection pool for this application, search for `WEBLOGIC JMS DEMO PROPERTIES` in the `weblogic.properties` file and replace the following property:

```
weblogic.jms.connectionPool=demoPool
```

To set up the pool, see “Configuring BEA WebLogic Server to Run the EJB” on page 2-54 of “Developing, Deploying, Using and Debugging a Container Managed Entity EJB”.

Defining JMS ConnectionFactories

ConnectionFactories allow JMS clients to create JMS connections. They can be configured to create connection pools with predefined attributes. The JMS specification classifies ConnectionFactories as *administered* objects. They are configured by the messaging system administrator and added to the JNDI namespace to allow access to JMS clients.

ConnectionFactories can be defined in the `weblogic.properties` file.

To define the ConnectionFactories for this application, search the `weblogic.properties` file for `WEBLOGIC JMS` and uncomment the following properties:

```
weblogic.jms.connectionFactoryName.trader=jms.connection.  
traderFactory
```

```
weblogic.jms.connectionFactoryArgs.trader=ClientID=traderReceive
```

Defining JMS Topics and Queues

The JMS specification classifies Queues, and Topics as *administered* objects. They are configured by the messaging system administrator and added to the JNDI namespace to allow access to JMS clients.

The JMS Queues and Topics that clients can access can be defined in the `weblogic.properties` file with the `weblogic.jms.queue` and `weblogic.jms.topic` properties.

This application does not use Queues.

To define the Topic that is required, search the `weblogic.properties` file for `WEBLOGIC_JMS` and uncomment the following property:

```
weblogic.jms.topic.exampleTopic=javax.jms.exampleTopic
```

Configuring BEA WebLogic Server for the Servlet

In this application the `TraderServlet` servlet will send *buy* and *sell* messages to a JMS Topic, to be received by the `TraderReceive` client which will invoke the EJB.

In this section you will register the servlet and include the path to the servlet in the servlet classpath. Both of these tasks will involve setting properties in the `weblogic.properties` file.

Registering the Servlet

In this application we use the Java servlet `TraderServlet`.

To register the servlet, search for `WEBLOGIC JMS DEMO PROPERTIES` in the `weblogic.properties` file and uncomment the following property:

```
weblogic.httpd.register.jmstrader=examples.jms.trader.  
TraderServlet
```

Setting the Servlet Classpath and Reloading Properties

In order to run and debug servlet projects in IBM VisualAge for Java, you need to specify the path for the directory where IBM VisualAge for Java stores the servlet classes. You can include more than one project in the servlet classpath. You may also include packages that are not in the IBM VisualAge for Java workspace. However, you will not be able to debug servlets that are not in the IBM VisualAge for Java workspace.

After modifying the classpath you have to reload the properties.

To add the servlet path to the servlet classpath, add (uncomment) the following property in the `weblogic.properties` file:

```
weblogic.httpd.servlet.classpath=\
    VisualAge/ide/project_resources/projectName
```

where:

VisualAge is the directory where IBM VisualAge for Java is installed

projectName is the name of the servlet project.

In this application, the property will be:

```
weblogic.httpd.servlet.classpath=\
    C:/weblogic/myserver/servletclasses
```

Note: For WebLogic Server Service Pack 8: If you have installed Service Pack 8, when running this JMS application the server will display an Out of Memory exception. To solve this problem, you must add the following line to the `weblogic.properties` file:

```
weblogic.jms.ignoreMemExhaustCheck=true
```

To reload the properties, uncomment the following property in the `weblogic.properties` file:

```
weblogic.httpd.servlet.reloadCheckSecs=1
```

Note: This assumes that you have installed IBM VisualAge for Java into the default location.

Running the Client Application

The `TraderReceive` client will receive the messages sent to the Topic by the servlet and invoke an EJB to process them.

This application uses the `StatelessSession` EJB (from the example in Chapter 2) which you must have successfully built and deployed in order to continue.

To run the client application:

1. Ensure that the following property is uncommented in the `c:\weblogic\weblogic.properties` file:

```
weblogic.ejb.deploy =\
    c:\weblogic\myserver\ejb_basic_statelessSession.jar
```
2. Start the BEA WebLogic Server (see “Verifying the EJB Deployment” on page 2-58).
3. Start IBM VisualAge for Java.
4. In the package `examples.jms.trader` under the WebLogic Examples Project, select the `TraderReceive (client)` class.
5. Right click on the `TraderReceive (client)` class and go to properties. On the Program tab, check that the Command line Arguments contain the following setting:

```
t3://hostname:port
```

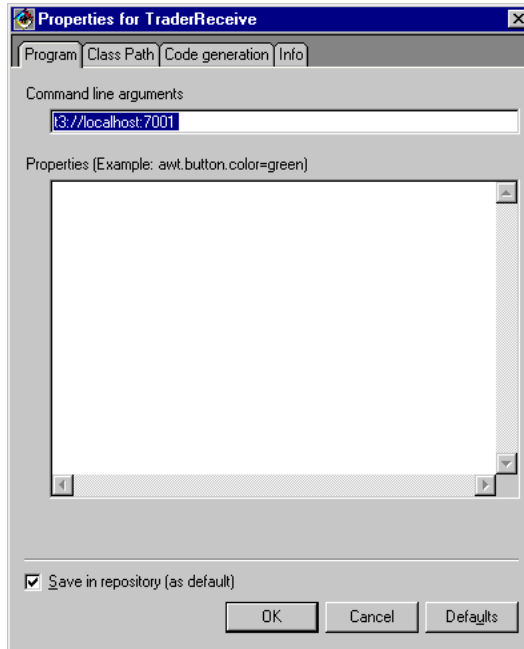
where:

`hostname` is the host name of the BEA WebLogic Server

`port` is the port where the server is listening for connections
(`weblogic.system.ListenPort`).

In this application, it will translate to: `t3://localhost:7001`.

Figure 3-1 The Command Line Arguments field in the Properties window



6. Select the Class Path tab and click the Edit button for the Project Path.
7. Ensure that all the WebLogic projects are checked.

8. Run the `TraderReceive` class.

Figure 3-2 and Figure 3-3 below show the messages that should appear on the console.

Figure 3-2 Messages from the server

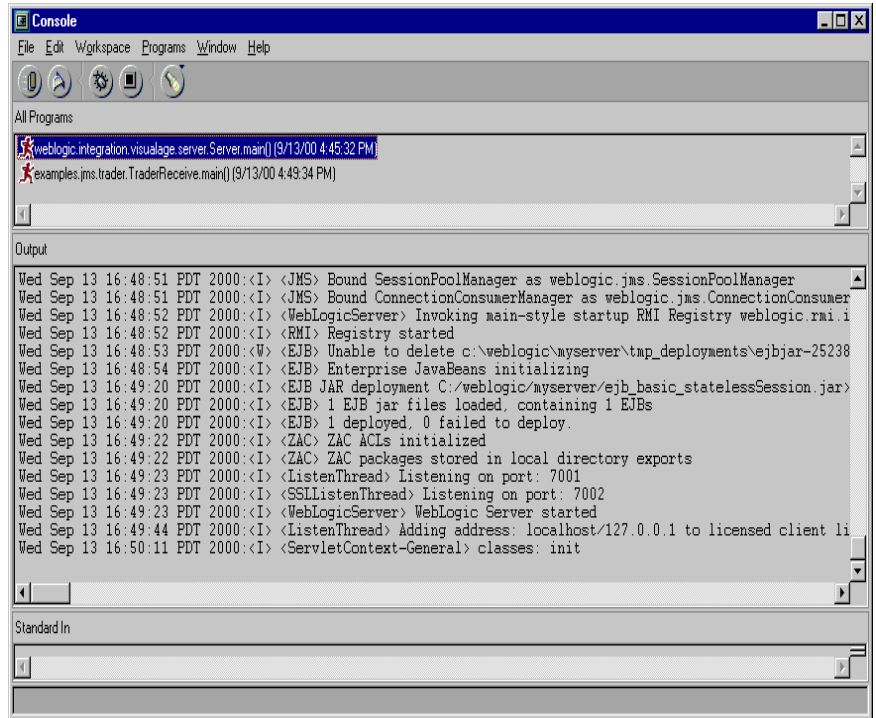
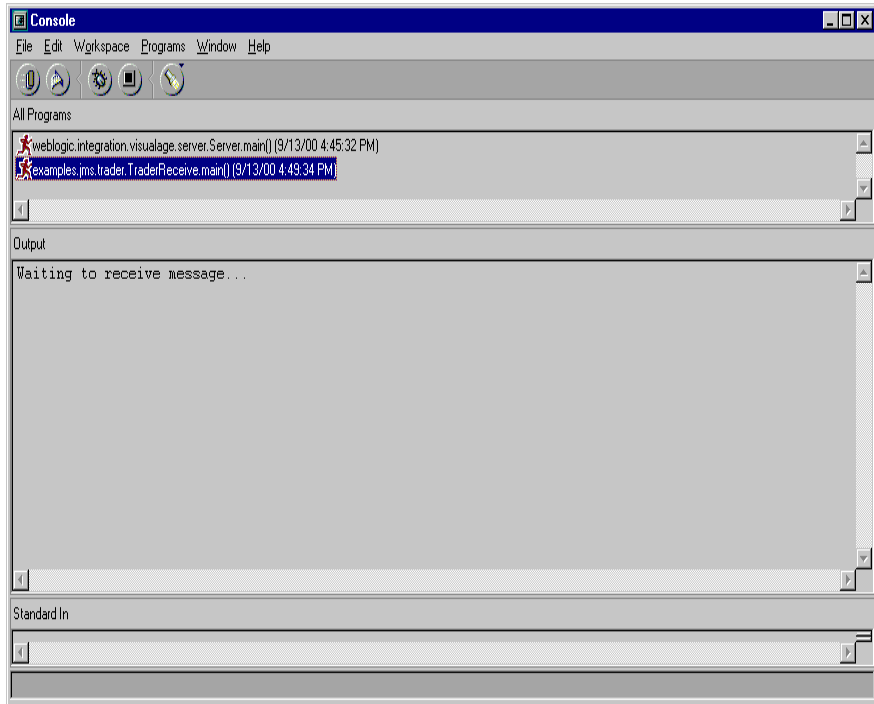


Figure 3-3 Message from the client, `TraderReceive`



Calling the Servlet from a Web Browser

Now that the client application is running, you can load the servlet into a web browser and send messages to the JMS Topic where the client will receive them.

To load the servlet into a web browser, request the following URL in the browser:

```
http://hostname:port/jmstrader
```

where:

hostname is the host name of the BEA WebLogic Server

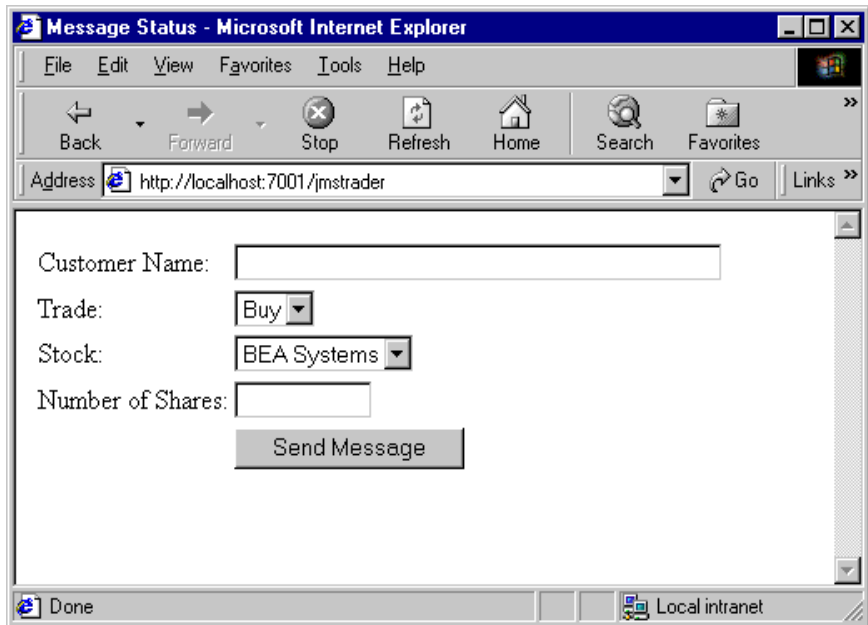
port is the port where the server is listening for connections
(`weblogic.system.ListenPort`)

In this application, it will translate to:

```
http://localhost:7001/jmstrader
```

Figure 3-4 below shows the form that appears in the browser.

Figure 3-4 The client application running



The screenshot shows a Microsoft Internet Explorer window titled "Message Status - Microsoft Internet Explorer". The address bar displays "http://localhost:7001/jmstrader". The main content area contains a web form with the following elements:

- Customer Name:
- Trade:
- Stock:
- Number of Shares:
-

The status bar at the bottom shows "Done" and "Local intranet".

To submit a trade request to the server, fill in the fields on the form and click the Send Message button.

Figure 3-5 Entering data into the fields

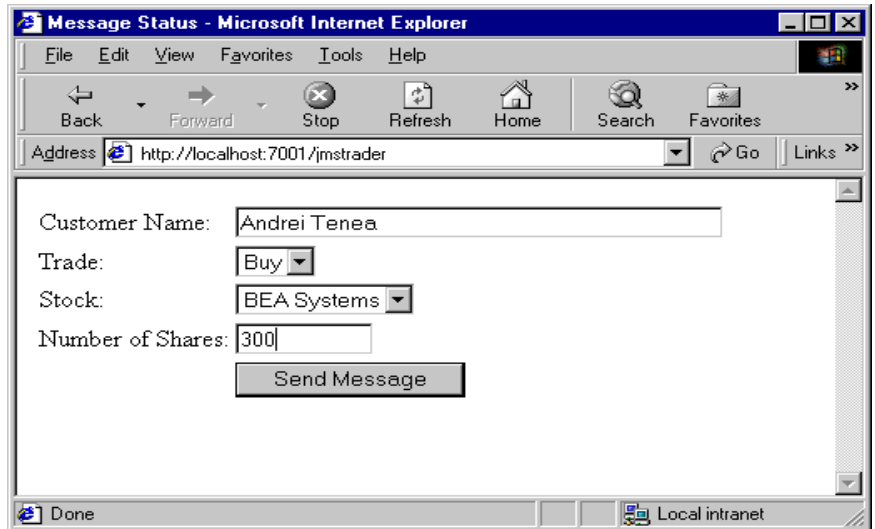
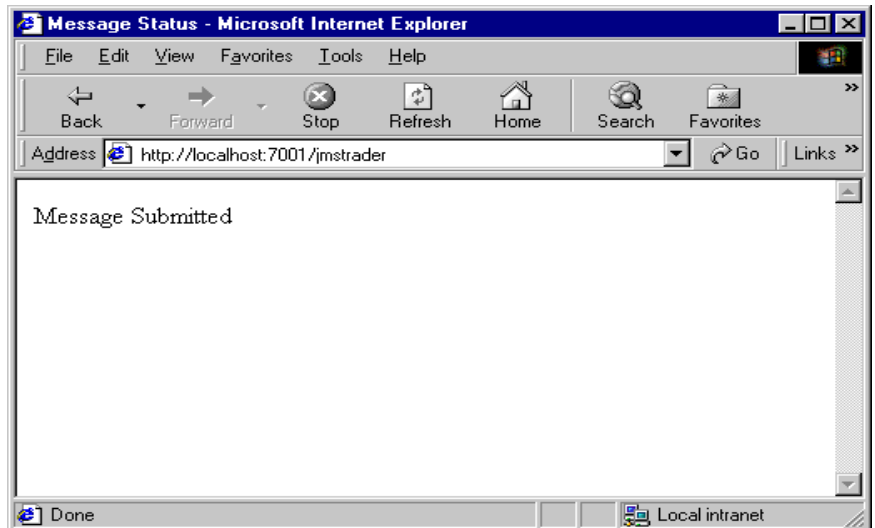
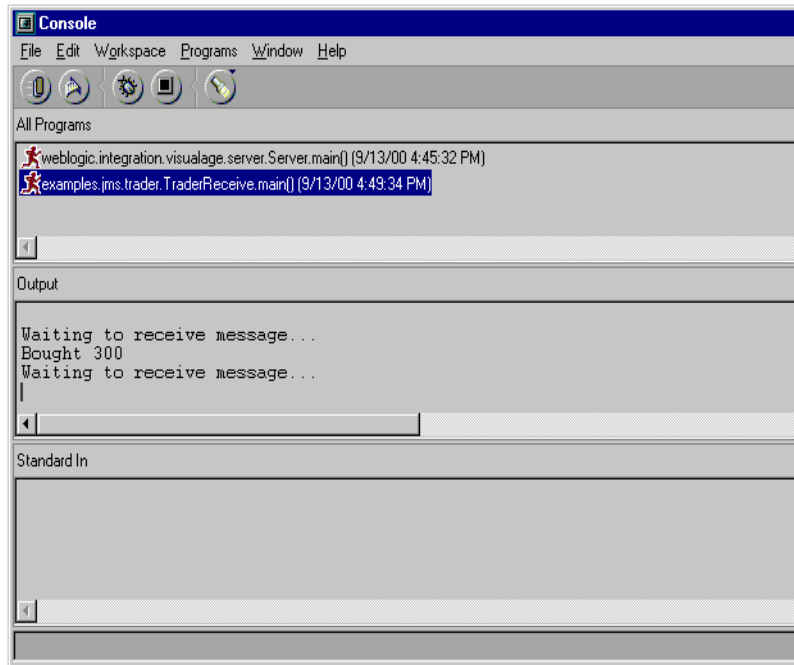


Figure 3-6 The message returned after submitting the data



The `TraderReceive` client displays messages as they are received from the Topic. The invoked EJB also displays messages in the console while processing the request.

Figure 3-7 The VisualAge console showing messages from `TraderReceive`



Exporting the Classes to the Production BEA WebLogic Server

When you have finished developing your BEA WebLogic Server application in IBM VisualAge for Java, you have to export it to a file system for use in a production environment. Refer to Chapter 5, “Exporting Classes.”

In this application we must export:

- the servlet class `TraderServlet.class` to
`WebLogic/myserver/servletclasses/examples/jms/trader`
- the client class `TraderReceive.class` to
`WebLogic/myserver/servletclasses/examples/jms/trader`

In order to debug the servlet, we set the servlet classpath to the IBM VisualAge for Java workspace by adding (uncommenting) the following property:

```
weblogic.httpd.servlet.classpath=\
C:/Program Files/IBM/Visual Age for
Java/ide/project_resources/Weblogic Examples
```

in the `weblogic.properties` file.

To use the production BEA WebLogic server only, modify the servlet classpath to:

```
weblogic.httpd.servlet.classpath=\
weblogic/myserver/servletclasses
```


4 Developing an Applet Application

Topics discussed in this chapter include:

- Setting Up the Database
- Running and Debugging the Applet in the IBM VisualAge for Java Environment
- Running the Applet in BEA WebLogic Server and a Web Browser

Overview

This chapter demonstrates how to run and debug an applet inside IBM VisualAge for Java IDE, how to deploy the applet into a production environment using BEA WebLogic server, and how to test the applet using a web browser.

PhoneBook1 is an applet that accesses a small database, containing names and addresses, via BEA WebLogic Server JDBC. The applet lists the entries in the `Customer` table and allows you to select any entry, displaying its details in separate fields.

This applet is one of the examples from the BEA WebLogic Server distribution, adapted to use an IDE (IBM VisualAge for Java) and an Oracle database. It was installed into the `WebLogic Examples` project by the Integration Kit Installer.

As a result of Java's security model, one cannot use a regular two-tier JDBC driver in an applet. We are using BEA WebLogic Server JDBC to connect to a pool driver on the BEA WebLogic Server. We then configure the pool driver to use an accompanying BEA WebLogic Server `jdbcDriver` for Oracle two-tier driver.

Setting Up the Database

The applet uses a connection pool called `demoPool`, which connects to an Oracle database.

In order to run this application you must set up the Oracle database and create and populate the table, `Customer`. For detailed instructions on setting up the database see “Setting up the Oracle Database” on page 2-42. Following these instructions, nothing more is required from the JDBC/DBpool point of view because the database created already contains the required `Customer` table.

If you want to use a different RDBMS (other than Oracle), you will have to create a database table named `Customer` with the fields `Custid`, `Name`, `Address`, `City`, `State`, `Zip`, `Area`, and `Phone`. You will also have to re-configure the `demoPool` connection pool for your database.

Running and Debugging the Applet in the IBM VisualAge for Java Environment

This section describes the steps needed to run the applet with the IBM VisualAge for Java Applet Viewer, which will then enable you to debug the applet.

Developing the Applet

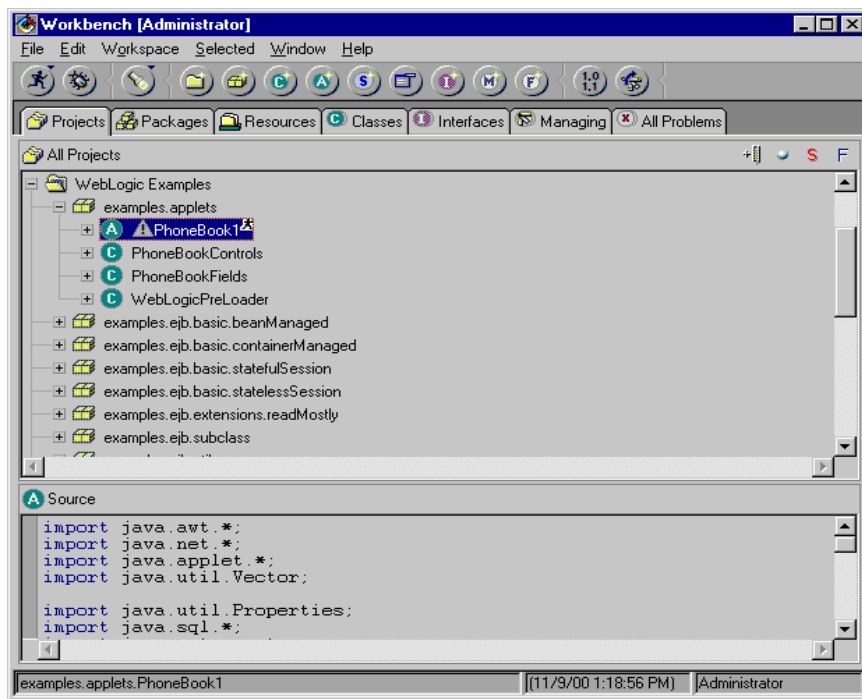
All the necessary classes have already been developed and installed in the IBM VisualAge for Java workspace in the `examples.applets` package which is in the `WebLogic Examples` project.

Notice that the `examples.applets` package already contains the four classes:

- Applet runnable class: `PhoneBook1`
- Helper classes:
 - `PhoneBookControls`
 - `PhoneBookFields`
 - `WebLogicPreloader`

In this tutorial we will not be using the `WebLogicPreloader` class which can be used to ensure that all of the necessary classes required by BEA WebLogic Server JDBC are downloaded by a separate thread.

Figure 4-1 The `examples.applets` package in the Workbench



In a real development situation, you will have to create or import the applet classes into a project/package inside IBM VisualAge for Java.

Verifying the Classpath

All the classes needed by the applet (other than the internal BEA WebLogic Server ones) have to reside in the project or package in which the applet class resides.

BEA WebLogic Server classes called by the applet class must be included in the applet's classpath.

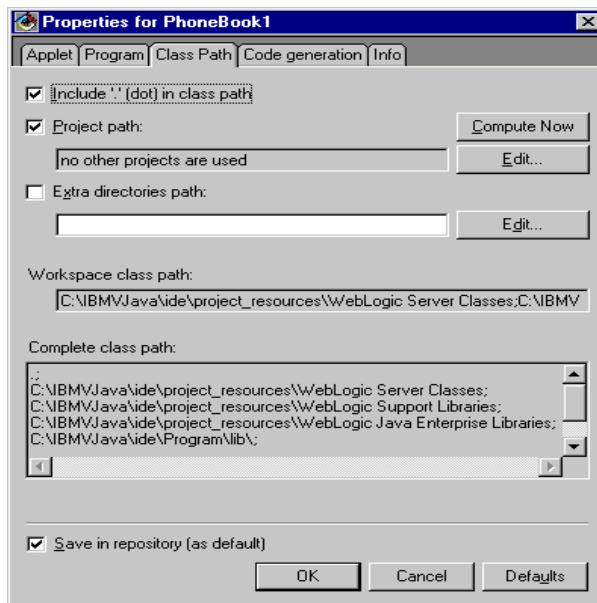
To verify the complete classpath for the applet:

1. Select the applet class, PhoneBook1.
2. From the menu bar, select Selected→Properties to open the Properties window.
3. Select the Class Path tab and inspect the Complete Class Path list.

It should include:

```
VisualAge\ide\project_resources\WebLogic Server Classes;  
VisualAge\ide\project_resources\WebLogic Support Libraries;  
VisualAge\ide\project_resources\WebLogic Java Enterprise  
Libraries;
```

Figure 4-2 The Class Path tab showing the Complete Classpath for PhoneBook1



To add any of these projects, click the Edit button for the Project Path field to open the Class Path window and check off the required WebLogic projects.

Using Attributes and Parameters

If the applet requires attributes or parameters that are normally specified in the HTML page, you must add them to the properties of the applet class.

To specify attributes or parameters normally specified in the HTML page:

1. In the Properties window for the applet class, `PhoneBook1`, go to the Applet tab.
2. Add the values for the attributes in the Attributes pane.

In this application we have already entered the values for the Width and Height attributes:

Width: 500

Height: 800

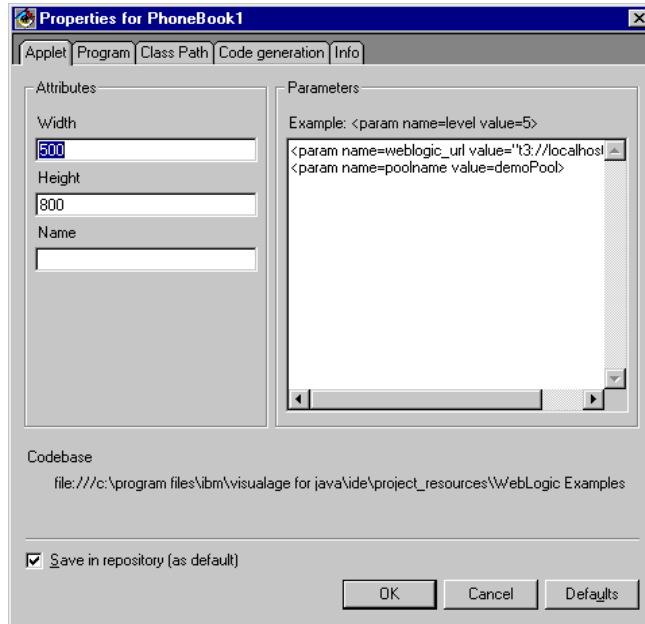
3. Add the parameters to the Parameters text box.

In this application we have already added parameters related to the `weblogic_url` and the `poolname`:

```
<param name=weblogic_url value="t3://localhost:7001">
```

```
<param name=poolname value=demoPool>
```

Figure 4-3 The Applet tab showing the attributes and parameters



As seen in Figure 4-3, IBM VisualAge for Java recognizes the applet Codebase as C:\Program Files\IBM\VisualAge for Java\IDE\project_resources\WebLogic Examples.

Running the Applet

Running the applet inside the IBM VisualAge for Java IDE enables you to debug it.

To run the applet with the IBM VisualAge for Java Applet Viewer:

1. Configure the server to run the container managed entity EJB (see “Configuring BEA WebLogic Server to Run the EJB” on page 2-54).

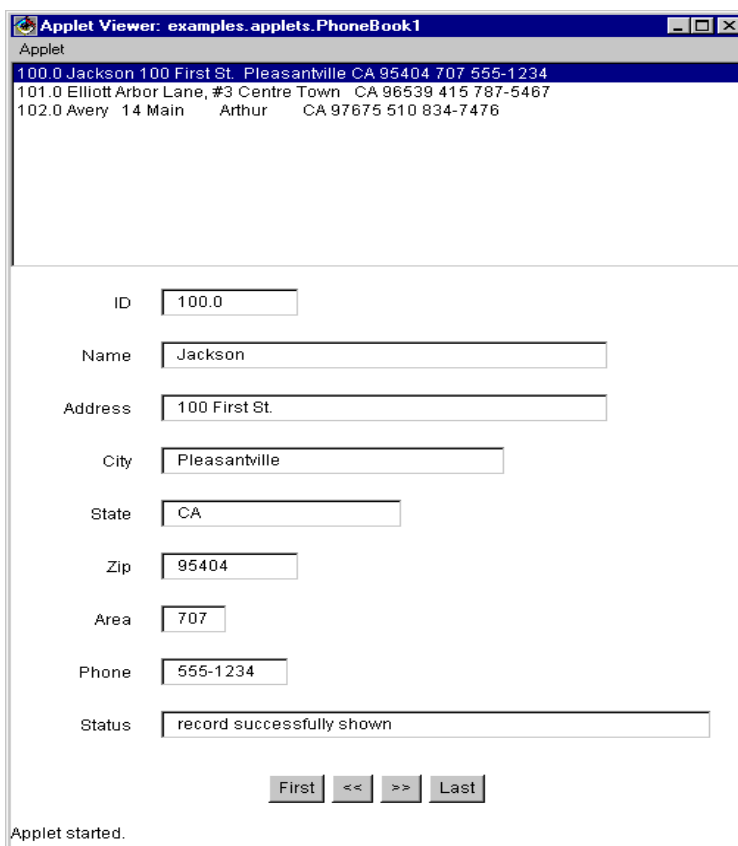
Ensure that the `weblogic.properties` file contains the following entry that deploys the EJB:

```
weblogic.ejb.deploy=\
C:/weblogic/myserver/ejb_basic_containerManaged.jar
```

2. Start the server so the applet can connect to it via JDBC (see “Verifying the EJB Deployment” on page 2-58).
3. In the Workbench select PhoneBook1 in the `examples.applets` package and from the menu bar, select Selected→Run→In Applet Viewer.

The applet should load, connect to the BEA WebLogic Server via JDBC, download the entries from the database, and allow you to view the details of each entry in the Applet Viewer by selecting a name from the list at the top of the Applet.

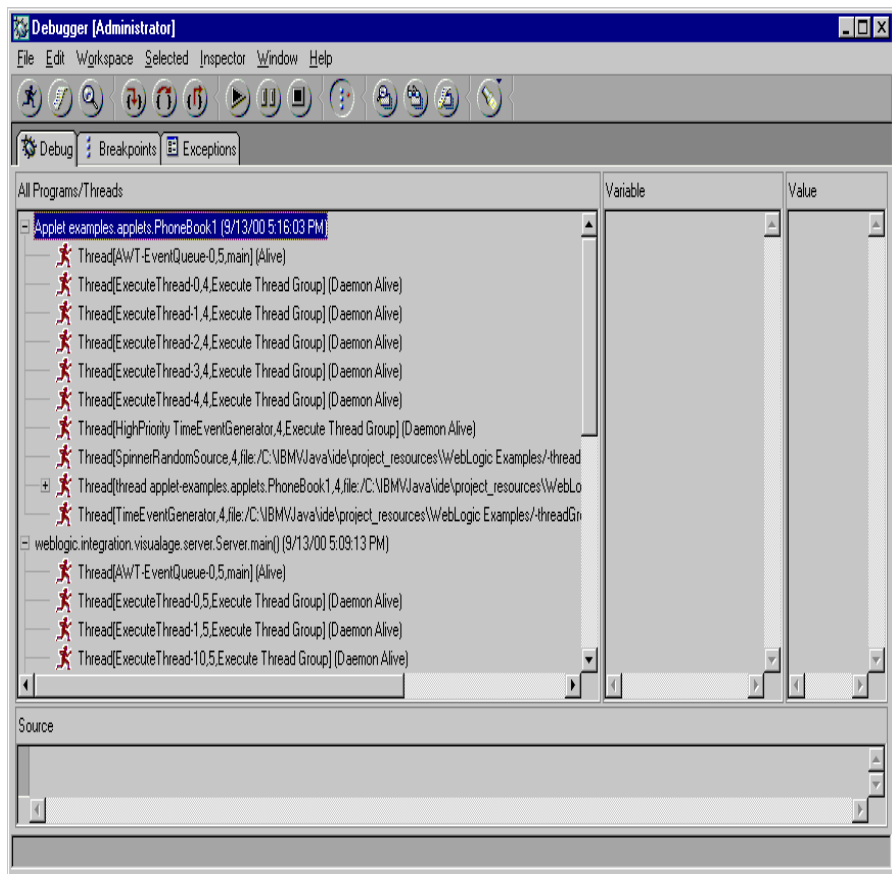
Figure 4-4 The Applet Viewer showing entries in the database



Debugging the Applet

While the applet is running you will see the debuggable applet thread in the Debug tab of the Debugger.

Figure 4-5 The Debug tab showing the applet thread

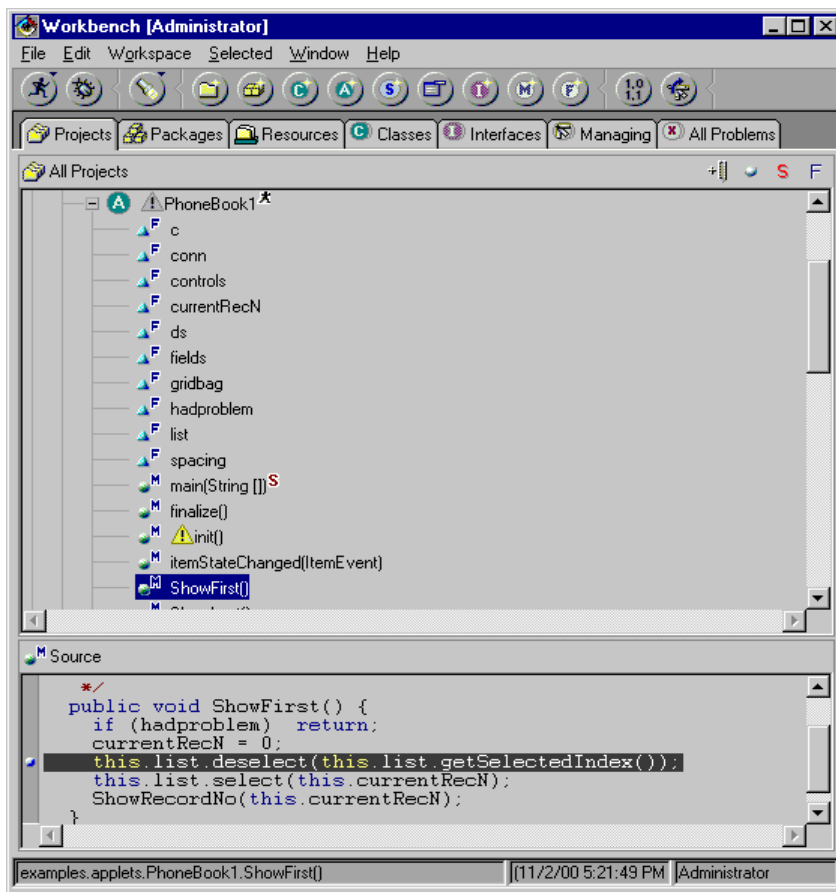


You can debug the applet if necessary (see “Debugging the Client Application and the Server Object” on page 2-25 for detailed instruction on debugging inside IBM VisualAge for Java, and IBM VisualAge for Java documentation about the Integrated Debugger).

To debug the PhoneBook1 applet:

1. In the Debug tab of the Debugger set a breakpoint in the `ShowFirst()` method of the `PhoneBook1` class (see Figure 4-6).

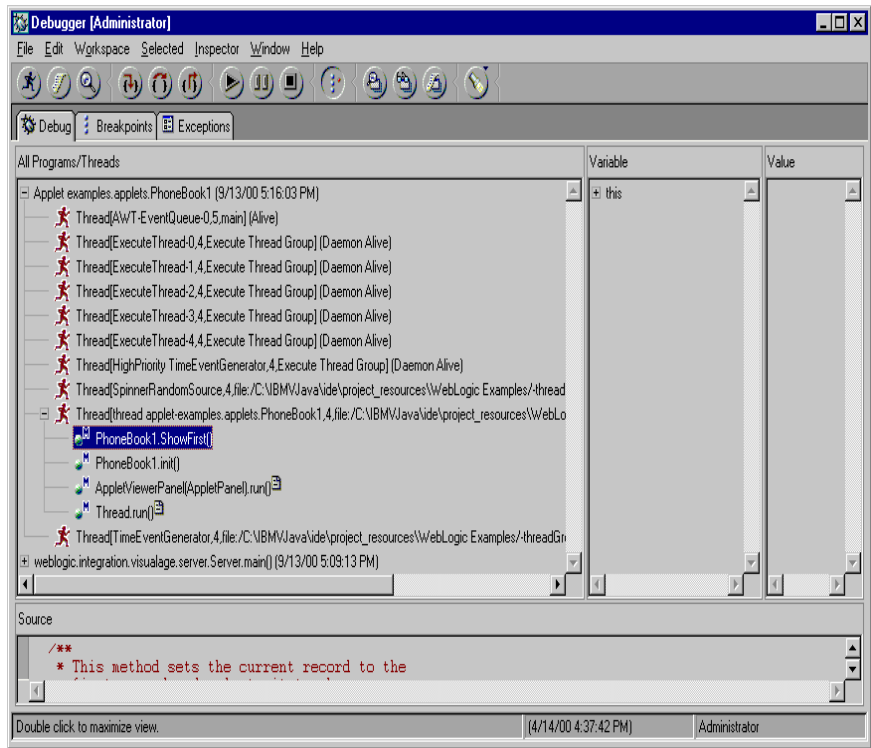
Figure 4-6 Setting the breakpoint in the `ShowFirst()` method



2. Reload the applet by selecting Applet→Reload in the Applet Viewer.

The applet initializes (its window is empty) and the debugger shows the suspended thread (among the other running threads started by the applet).

Figure 4-7 The Debugger showing the suspended thread



3. Resume the suspended thread.

The applet will continue running and the Applet Viewer will display the database as before.

Note: If you close the Applet Viewer, you shut down the applet and the applet threads will disappear from the Debugger window

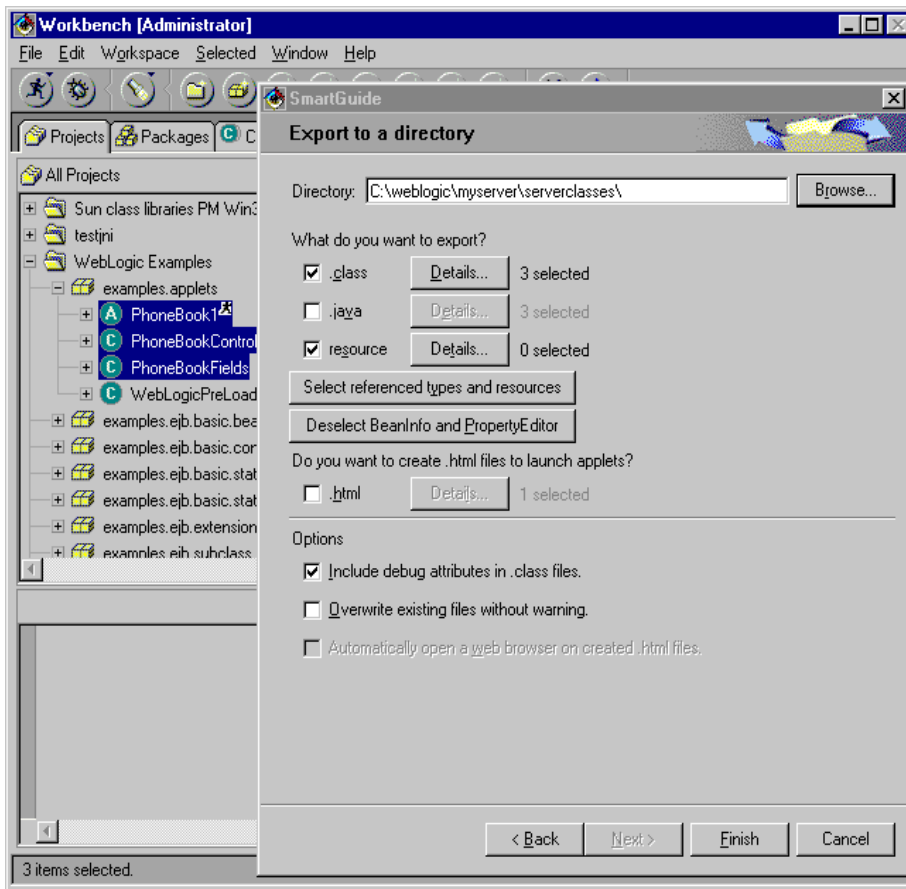
Running the Applet in BEA WebLogic Server and a Web Browser

This section describes the steps needed to run the applet in a web browser. While the applet is running in a web browser VM, not the IBM VisualAge for Java VM, you cannot debug it. You can, however, test the applet in its HTML context.

To run the applet in a web browser:

1. Export the applet classes to the *WebLogic\myserver\serverclasses* directory of the BEA WebLogic Server distribution as described in Chapter 5, “Exporting Classes.”

Figure 4-8 Exporting the applet classes to the WebLogic directory



2. Create the HTML file that runs and tests the applet and copy it to the server's document root directory.

The HTML file for this application, `phonebook1.html` is already created and you can find it in the folder `WebLogic\examples\applets`, where `WebLogic` is the BEA WebLogic Server installation folder.

The document root directory is where the BEA WebLogic Server searches for public HTML files. By default it is set to the directory *WebLogic\myserver\public_html* in your BEA WebLogic Server installation directory.

For this application, copy *PhoneBook1.html* to the *WebLogic\myserver\public_html* directory.

3. Run the server.

For production purposes, run the server from outside IBM VisualAge for Java (see the BEA WebLogic Server documentation for the configuration settings: <http://www.weblogic.com/docs51/examples/applets/Package-examples.applets.html>)

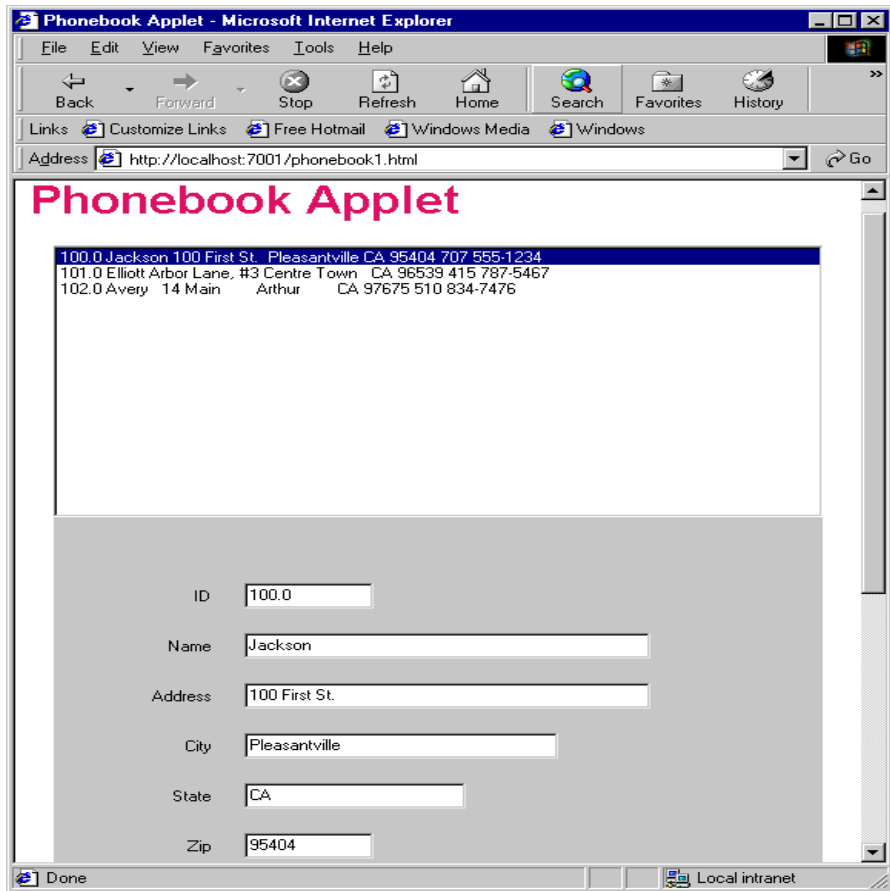
For testing purposes you can also start the server from inside IBM VisualAge for Java (see “Debugging the Client Application and the Server Object” on page 2-25).

4. In the web browser, request the URL:

`http://localhost:7001/phonebook1.html`

After some time needed to load all the necessary classes, the applet will appear.

Figure 4-9 The applet running in the web browser



Note: It has been observed that in certain implementations of Windows Internet Explorer the applet cannot be run in the browser.

You can find more details about using BEA WebLogic Server and Web browsers in the BEA WebLogic Server documentation.

5 Exporting Classes

Topics discussed in this chapter include:

- Exporting Classes to a Production BEA WebLogic Server

Overview

When you have finished developing your BEA WebLogic Server application in IBM VisualAge for Java, you will need to export it to a file system for use in a production environment.

Exporting Classes to a Production BEA WebLogic Server

When exporting the application to the production environment you do not have to export the EJB JAR files because they are exported as part of the development process. Other classes compiled within the IBM VisualAge for Java workspace during development do have to be exported.

The three directories where classes for BEA WebLogic Server applications are placed are:

- *WebLogic*/myserver/clientclasses for classes required by client applications
- *WebLogic*/myserver/serverclasses for classes required by server-side objects
- *WebLogic*/myserver/servletclasses for servlet classes

where *WebLogic* is the installation directory for BEA WebLogic Server.

Classes from your BEA WebLogic Server project in IBM VisualAge for Java should be exported to the appropriate directories. Classes that are shared by the client and server should be exported to both *WebLogic*/myserver/clientclasses and *WebLogic*/myserver/serverclasses.

You can export classes with their resources and/or source code. You can also export packages (or projects) with their included classes, resources and/or source code.

To export classes or packages:

1. Select the required classes or packages.
2. From the menu bar, select Selected—>Export to open the SmartGuide window.
3. Select Directory as the Export Destination and click Next.
4. Enter the appropriate directory in the Directory field.
5. Under the section What do you want to export?, check the appropriate boxes:
 - .class: for classes
 - .java: for the source code of the selected classes or resources
 - resource: for resources

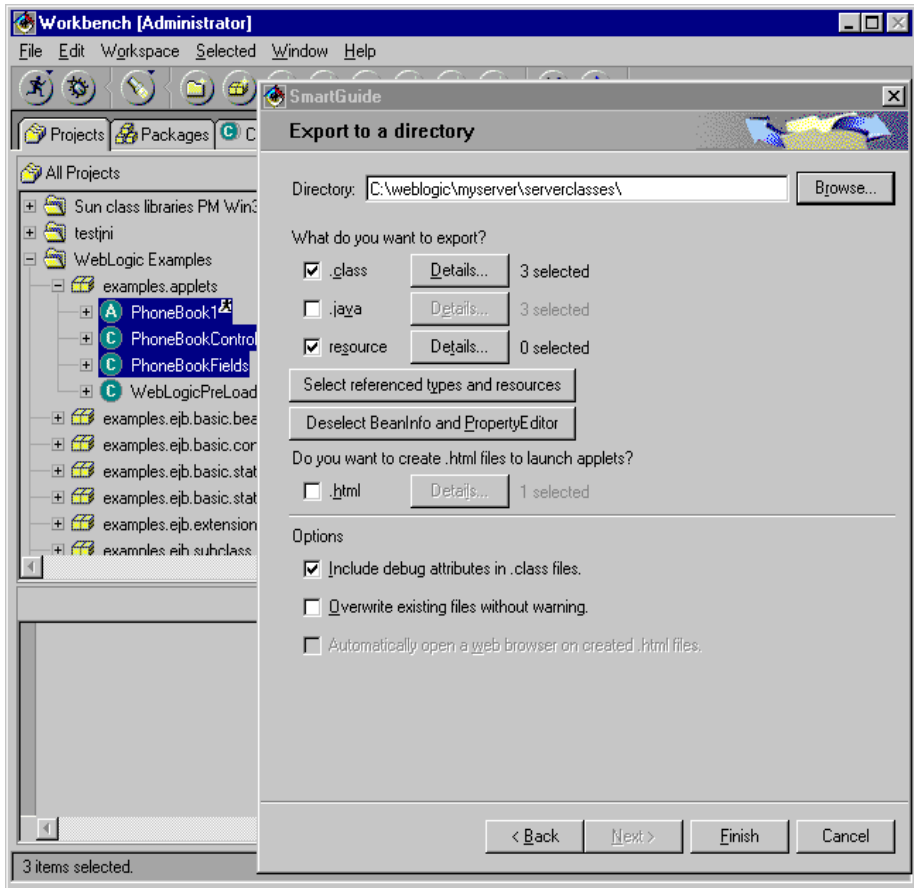
Notice that next to the Details button for each of these options you will be told how many are currently selected.

6. For each box you have checked, click the Details button to see or change the selected classes. If you are exporting a package, all the classes in the package will be selected. You can select or deselect classes.

Note: The system is not dynamic enough to reflect these changes on the Workbench.

In Figure 5-1, 3 classes have been selected which means that the source code for 3 classes has also been selected. However the .java box has not been checked, so the source code will not be exported.

Figure 5-1 The SmartGuide window showing that 3 classes have been selected



7. Select Finish.

The specified files will be exported to the specified directory.

A Tips and Troubleshooting

Topics discussed in this section include:

- Installing Service Packs
- Using EJB Dynamic Deployment
- Using Cloudscape Database

This section provides tips and discusses problems you may encounter while using the BEA WebLogic Integration Kit for IBM VisualAge for Java.

Installing Service Packs

Service packs are occasionally released for BEA WebLogic Server to provide a safe, easy and convenient way for users to incorporate resolved issues into their current release. For more information, see WebLogic FAQ on service packs.

To install a service pack into your BEA WebLogic Integration Kit for IBM VisualAge for Java:

1. Extract the service pack .zip file into a temporary directory.
2. Select the `webLogic Server Classes` project in the Projects tab of the VisualAge for Java Workbench.

3. From the menu bar, select Selected→Managed→Version.
4. Select Jar File as the import source, then select Next.
5. Click the Browse button next to the Filename field, and select the service pack JAR that you previously extracted into a temporary directory.
6. Under What type of file do you want to import?, choose .class and resource.
7. Select Finish. VisualAge will ask you, if want to create an edition of classes that are replaced by the service pack. Select Yes To All.

In order to keep track of the service packs that you have installed, you should version all of the classes that you have just installed with an appropriate name.

To version all of the classes that you have just installed:

1. Select the Show Edition Names button at the top of the VisualAge IDE.
2. Select each class that has just been installed as part of the service pack. To select multiple classes hold down the Ctrl key while you highlight the names of the appropriate classes. To locate all the classes that have been replaced, see the list of all classes included in the service pack that is provided in the service pack documentation.
3. From the Selected menu, choose Managed, then Version.
4. Select One Name. In the corresponding field, enter a name that is representative of the service pack that you have just installed, such as 451sp1.
5. Select OK.

Note: If you have installed WebLogic Server Service Pack 8, the server will display an Out of Memory exception when you run a JMS application. To solve this problem, you must add the following line to the `weblogic.properties` file:

```
weblogic.jms.ignoreMemExhaustCheck=true
```

Using EJB Dynamic Deployment

You cannot use the new dynamic EJB deployment while debugging server-side code in IBM VisualAge for Java.

Dynamic deployment uses classloaders in ways that are incompatible with the debugger classloader in VisualAge for Java. However, the debugger will reloads modified classes at run time, so you can do much of the work of developing and debugging server applications without restarting the server.

Using Cloudscape Database

The Integration Kit does not support the use of the Cloudscape database with IBM VisualAge for Java Version 3.5 and BEA WebLogic Server Version 5.1. Cloudscape database usage is unsupported because of problems that occur when Cloudscape is run in a VisualAge environment. This issue is currently being investigated by IBM Support (PMR 15142,519,000).

