# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>1</td>
</tr>
<tr>
<td>1. Getting Started</td>
<td>9</td>
</tr>
<tr>
<td>Software Requirements for the Tutorial</td>
<td>9</td>
</tr>
<tr>
<td>What You Need to Run the Forte for Java IDE</td>
<td>10</td>
</tr>
<tr>
<td>What You Need to Create and Run the Tutorial</td>
<td>10</td>
</tr>
<tr>
<td>Creating the Tutorial Database Tables</td>
<td>11</td>
</tr>
<tr>
<td>Starting Up the Java 2 SDK, Enterprise Edition Software</td>
<td>14</td>
</tr>
<tr>
<td>Starting the Forte for Java IDE</td>
<td>16</td>
</tr>
<tr>
<td>Starting the IDE on Solaris, UNIX, and Linux Environments</td>
<td>16</td>
</tr>
<tr>
<td>Starting the IDE on Microsoft Windows</td>
<td>16</td>
</tr>
<tr>
<td>Modifying the Session With Command-Line Switches</td>
<td>17</td>
</tr>
<tr>
<td>Specifying Your User Settings Directory</td>
<td>18</td>
</tr>
<tr>
<td>Setting up the IDE to Run the Reference Implementation</td>
<td>19</td>
</tr>
<tr>
<td>Understanding the Forte for Java Directory Structure</td>
<td>20</td>
</tr>
<tr>
<td>2. Introduction to the Tutorial</td>
<td>23</td>
</tr>
<tr>
<td>Functionality of the Tutorial Application</td>
<td>23</td>
</tr>
<tr>
<td>Application Scenarios</td>
<td>24</td>
</tr>
<tr>
<td>Application Functional Specification</td>
<td>25</td>
</tr>
</tbody>
</table>
Testing Entity Beans  62
  Creating a Test Client for an Entity Bean  62
  Testing an Entity Bean With the Test Client  66
Creating a Session Bean With the EJB Builder  70
  Coding a Session Bean's create Method  71
  Creating Business Methods to Get the Detail Data  73
  Adding EJB References  75
Testing the Session Bean  77
  Creating a Test Client for a Session Bean  77
  Using the Test Client to Test a Session Bean  78
Comments on Creating a Client  79

4. Creating the Tutorial Application’s Web Service  81
   Overview of the Tutorial’s Web Service  81
     The Web Service  82
     The XML Operations  82
     The Runtime Classes  83
     The Client Pages  83
   Creating XML Operations  84
     Creating XML Operations Based on Entity Bean Methods  84
     Creating XML Operations Based on Other Methods  89
   Creating a Web Service  91
   Creating a J2EE Application for the Web Service  92
   Deploying the Tutorial Application  93
   Testing the Generated Client Pages  94
5. Creating a Client for the Tutorial Application  101
   Developing the Tutorial’s Client  101
      Joining Development and Web Design With Forte for Java Tools  101
      Skipping the Forte ESP Toolkit Step for EatersDigest  102
   Adding the Modified JSP Pages to the Tutorial Application  103
   Running the Tutorial Application  104
   Understanding the Modifications  105
      Passing a Parameter Within an HTML Form  106
      Adding a Customer Review  108

Index  111
Figures

FIGURE 2-1 EatersDigest Application Architecture  29
FIGURE 2-2 EatersDigest Application Architecture With Manager Session Bean  31
FIGURE 3-1 Function of a Detail Class  40
FIGURE 5-1 Generated Restaurant List Page vs. the DGAllRestaurants Page  105
FIGURE 5-2 Generated View Customer Review Page vs.
DGAllCustomerreviewsByRestaurantName Page  106
FIGURE 5-3 Generated Input Form for AddCustomerReviewXMO  108
FIGURE 5-4 DGEnterCustomerReview Page  108
## Tables

| TABLE 1-1 | EatersDigest Database Tables | 11 |
| TABLE 1-2 | Restaurant Table Records | 12 |
| TABLE 1-3 | CustomerReview Table Records | 12 |
| TABLE 1-4 | "CustomerreviewEJBTable" Table Records | 12 |
| TABLE 1-5 | runide Command-Line Switches | 17 |
| TABLE 1-6 | Forte for Java Directory Structure | 20 |
| TABLE 1-7 | Directory Structure for the User Settings Directory | 21 |
Preface

Welcome to the Forte™ for Java™, Enterprise Edition tutorial. In this tutorial, you will learn to use the following features introduced in the Enterprise Edition:

- EJB™ 1.1 Builder—for creating, developing, and testing the Enterprise JavaBeans™ components of the tutorial application
- EJB module assembly—for assembling the EJB components into an EJB module, which you export into an EJB Java Archive (JAR) file
- Test application facility—for testing enterprise beans without having to create a client manually, using the J2EE Reference Implementation as the application server.
- Web Services module—for building a web service from the existing EJB components and generating JSP™ pages viewable from a web browser
- J2EE Reference Implementation module—for deploying and testing the tutorial application

You can create the tutorial application on the following platforms and operating systems:

- Solaris™ 8 SPARC™ Platform Edition
- Microsoft Windows 2000, SP2
- Microsoft Windows NT 4.0, SP6
- Red Hat Linux 6.2

All screen shots in this book are from the Windows NT version of the Forte for Java software. You should have no trouble translating the slight visual differences to other platforms. Although almost all procedures use the Forte for Java user interface, occasionally you might be instructed to enter a command at the command line. In such cases, examples are given with the prompt and syntax for a Microsoft Windows command window. For example:

    c:\>cd MyWorkDir\MyPackage
To translate for UNIX® or Linux environments, simply change the prompt and use forward slashes:

```bash
% cd MyWorkDir/MyPackage
```

---

## Before You Read This Book

This tutorial creates an application that conforms to the architecture documented in Java 2 Platform, Enterprise Edition Blueprints. If you want to learn how to use the features of Forte for Java, Enterprise Edition to create, develop, and deploy a J2EE-compliant application, you will benefit from working through this tutorial. Before starting, you should be familiar with the following subjects:

- Java programming language
- Enterprise JavaBeans concepts
- Java Servlet syntax
- JDBC™ enabled driver syntax
- JavaServer Pages syntax
- HTML syntax
- Relational database concepts (such as tables and keys)
- How to create tables with the chosen database software
- J2EE application assembly concepts
- J2EE application deployment concepts

This book requires a knowledge of J2EE concepts, as described in the following resources:

How This Book Is Organized

This manual is designed to be read from beginning to end. Each chapter in the tutorial builds upon the code developed in earlier chapters.

Chapter 1 describes the software requirements for the tutorial, explains how to start the Java 2 SDK, Enterprise Edition software, start the Forte for Java integrated development environment (IDE), set up the IDE to run the J2EE Reference Implementation, and create the tutorial database tables. It includes a descriptive list of the installed Forte for Java directories.

Chapter 2 describes the functionality and architecture of the tutorial application.

Chapter 3 provides step-by-step instructions for creating the EJB tier of the tutorial application. Describes how use the Forte for Java test application facility to test deploy each bean.

Chapter 4 describes how to use the Web Services module to create XML operations based on the application’s business methods, and automatically generate web services and test client for the application.

Chapter 5 explains how provided client pages were built upon the output generated from the Web Services module in Chapter 4.

Typographic Conventions

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>AaBbCc123</td>
<td>The names of commands, files, and directories; on-screen computer output</td>
<td>Edit your .login file. Use ls -a to list all files. % You have mail.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>What you type, when contrasted with on-screen computer output</td>
<td>% su</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Password:</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>Book titles, new words or terms, words to be emphasized</td>
<td>Read Chapter 6 in the User’s Guide. These are called class options. You must be superuser to do this.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>Command-line variable; replace with a real name or value</td>
<td>To delete a file, type rm filename.</td>
</tr>
</tbody>
</table>
Related Documentation

Forte for Java documentation includes books delivered in Acrobat Reader (PDF) format, online help, Readme files of example applications, and Javadoc™ documentation.

Documentation Available Online

The documents in this section are available from the Forte for Java portal, the docs.sun.com™ web site, and from Fatbrain.com, an Internet professional bookstore.


- Release Notes (PDF format)
  Available for each Forte for Java edition. Describe last-minute release changes and technical notes.

- Getting Started Guide (PDF format)
  Available for each Forte for Java edition. Describes how to install the Forte for Java product on each supported platform and includes other pertinent information, such as system requirements, upgrade instructions, web server and application server installation instructions, command-line switches, installed subdirectories, Javadoc setup, databases integration, and information on how to use the Update Center.

- The Forte for Java Programming Series (PDF format)
  This series provides in-depth information on how to use various Forte for Java features to develop well-formed J2EE applications.
    - Building Web Components - part no. 816-1410-10
      Describes how to build a web application as a J2EE web module using JSP pages, servlets, tag libraries, and supporting classes and files.
    - Programming Persistence - part no. 816-1411-10
      Describes support for different persistence programming models provided by Forte for Java: JDBC and Transparent Persistence.
Building Enterprise JavaBeans Components - part no. 816-1401-10
Describes how to build Enterprise JavaBeans components—session beans and entity beans with container-managed or bean-managed persistence—using the Forte for Java EJB Builder wizards and other graphical user interfaces.

Building Web Services - part no. 816-1400-10
Describes how to use the tools provided by the Web Services module to build web services. Web Services are application business services published as Extensible Markup Language (XML) documents delivered over HTTP connections.

Building JSP Pages That Use XML Data Services - part no. 816-1399-10
Describes how to use the Forte for Java Enterprise Service Presentation Toolkit to incorporate dynamic XML data in HTML.

Assembling and Executing J2EE Modules and Applications - part no. 816-1402-10
Describes how to assemble EJB modules and web modules into a J2EE application, and how to deploy and run a J2EE application.

Forte for Java tutorials (PDF format)
You can also find the completed tutorial applications in your user settings directory, under sampledir/tutorial.

Forte for Java, Community Edition Tutorial - part no. 816-1408-10

Forte for Java, Enterprise Edition Tutorial - part no. 816-1409-10
Provides step-by-step instructions for building an application using Enterprise JavaBeans components, the test application facility, and the Forte for Java Web Services technology.

Online Help
Online help is available inside the Forte for Java development environment. You can access it by pressing the help key (Help on Solaris, F1 on Microsoft Windows and Linux), or by choosing Help > Contents. Either action displays a list of help topics and a search facility.
Examples

Several examples, with accompanying Readme files, that illustrate a particular Forte for Java feature are available in the sampledir/examples subdirectory of your user settings directory. In addition, you can download Enterprise Edition-specific examples from the Forte for Java portal and unzip them into the examples directory. Completed tutorial applications—including the applications described in Forte for Java, Community Edition Tutorial and this document—are in the sampledir/tutorial directory.

Javadoc Documentation

Javadoc documentation is available within the IDE for many Forte for Java modules. Refer to the release notes for instructions on installing this documentation. When you start the IDE, you can access this Javadoc documentation within the Javadoc pane of the Explorer.

Accessing Sun Documentation Online

A broad selection of Sun system documentation is located at:

http://www.sun.com/products-n-solutions/hardware/docs

A complete set of Solaris documentation and many other titles are located at:

http://docs.sun.com

Ordering Sun Documentation

Fatbrain.com, an Internet professional bookstore, stocks select product documentation from Sun Microsystems, Inc.

For a list of documents and how to order them, visit the Sun Documentation Center on Fatbrain.com at:

http://www.fatbrain.com/documentation/sun
Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can email your comments to Sun at:

docfeedback@sun.com

Please include the part number (816-1409-10) of your document in the subject line of your email.
CHAPTER 1

Getting Started

This chapter explains what you need to do before starting the Forte for Java, Enterprise Edition tutorial. For your convenience, it duplicates some installation information from the *Getting Started Guide*.

The topics covered in this chapter are:

- “Software Requirements for the Tutorial,” which follows
- “Creating the Tutorial Database Tables” on page 11
- “Starting Up the Java 2 SDK, Enterprise Edition Software” on page 14
- “Starting the Forte for Java IDE” on page 16
- “Setting up the IDE to Run the Reference Implementation” on page 19
- “Understanding the Forte for Java Directory Structure” on page 20

Software Requirements for the Tutorial

This section describes how to prepare your system before starting the Forte for Java, Enterprise Edition tutorial. This means making sure you have everything required to run the Forte for Java integrated development environment (IDE), as well as what is required in addition to create and run the tutorial.

You can access general system requirements from the release notes or from the Forte for Java portal’s Documentation page

What You Need to Run the Forte for Java IDE

The Forte for Java IDE requires the Java Development Kit. When you install the IDE, the installer searches your system for the JDK™ software and will inform you if the correct version is installed or not. If the correct version is not installed, the installer informs you, then quits so that you can install the correct version, which you can download from the Forte for Java portal.

What You Need to Create and Run the Tutorial

You need the following items to create and run the tutorial. Some of these items are included in the default installation of Forte for Java, Community Edition.

- A web browser
  You need a web browser to view the tutorial application pages. This can be either Netscape Communicator™ or Microsoft’s Internet Explorer. Netscape Communicator, version 4.7, is available for download from the Forte for Java portal or the product CD.

- A web server—Tomcat, version 3.2
  To test the tutorial, you need a web server. This tutorial uses a version of Tomcat bundled with the IDE that provides the functionality of a web server for testing purposes.

- Database software—PointBase Network Server, version 3.5
  You need database software to create the tutorial’s database tables. PointBase is an install option when installing Forte for Java software. To verify that PointBase was installed, look for a pointbase directory under the Forte for Java home directory. If PointBase was not installed, you can rerun the installer to install it.

- J2EE Reference Implementation, version 1.2.2
  You need an application server to deploy the tutorial’s J2EE application. You must use the special version of the J2EE Reference Implementation that is included in the Java 2 SDK, Enterprise Edition, version 1.2.2, that you can download from the Forte for Java Download site or the product CD.
  You need to tell the IDE which application server you are using. How to do this is described in “Setting up the IDE to Run the Reference Implementation” on page 19.
Creating the Tutorial Database Tables

The Forte for Java, Enterprise Edition tutorial requires two database tables. Use the rest_pb.sql SQL script provided to create these tables. This is located in the
sampledir/tutorials/EatersDigest/SQLscript directory under your user
settings directory.

**Note** – The rest_pb.sql script actually creates two versions of the EatersDigest
tables: a normal database, which you use to create the EJB components and detail
classes, and a modified version, which follows the J2EE Reference Implementation
protocol of appending the string “EJBTable” to table names and quoting each table
and column name. Because you’ll already have this modified version of the tables in
your database when you test the EJB tier, you’ll set properties such that the RI is
prevented from creating and destroying its own version of your database tables each
time you deploy the application. If you were using another application server, you
would not need the quoted variations of the tables.

This rest_pb.sql script creates the database schemas shown in TABLE 1-1.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Columns</th>
<th>Primary Key</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restaurant</td>
<td>restaurantName</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cuisine</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>neighborhood</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>address</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>phone</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>description</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>rating</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CustomerReview</td>
<td>restaurantName</td>
<td>yes</td>
<td>Compound primary key with CustomerName; references Restaurant(restaurantName)</td>
</tr>
<tr>
<td></td>
<td>customerName</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>review</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Restaurant (and “RestaurantEJBTable”) tables contain the records shown in TABLE 1-2.

**TABLE 1-2 Restaurant Table Records**

<table>
<thead>
<tr>
<th>restaurant-Name</th>
<th>cuisine</th>
<th>neighborhood</th>
<th>address</th>
<th>phone</th>
<th>description</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>French Lemon</td>
<td>Mediterranean</td>
<td>Rockridge</td>
<td>1200 College Avenue</td>
<td>510 888</td>
<td>Very nice spot.</td>
<td>5</td>
</tr>
<tr>
<td>Bay Fox</td>
<td>Mediterranean</td>
<td>Piedmont</td>
<td>1200 Piedmont Avenue</td>
<td>510 888</td>
<td>Excellent</td>
<td>5</td>
</tr>
</tbody>
</table>

The CustomerReview table contains the records shown in TABLE 1-3.

**TABLE 1-3 CustomerReview Table Records**

<table>
<thead>
<tr>
<th>restaurantName</th>
<th>customerName</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>French Lemon</td>
<td>Fred</td>
<td>Nice flowers.</td>
</tr>
<tr>
<td>French Lemon</td>
<td>Fred</td>
<td>Excellent Service</td>
</tr>
</tbody>
</table>

(Just for variety) the “CustomerreviewEJBTable” table contains the records shown in TABLE 1-3.

**TABLE 1-4 “CustomerreviewEJBTable” Table Records**

<table>
<thead>
<tr>
<th>restaurantName</th>
<th>customerName</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>French Lemon</td>
<td>Ralph</td>
<td>Nice flowers.</td>
</tr>
<tr>
<td>French Lemon</td>
<td>Fred</td>
<td>Excellent Service</td>
</tr>
</tbody>
</table>

Install these tables in a PointBase database according to the following instructions.

1. **Start the PointBase Network Server.**
   - In Solaris or Linux environments: Run the `netserver.sh` file in the `forte4j-home/pointbase/network` directory.
   - On Microsoft Windows: Choose Start > Forte for Java EE > PointBase > Network Server > Server or double-click the `netserver.bat` file in the `forte4j-home/pointbase/network` directory.
2. Start the PointBase Console.

- In Solaris or Linux environments: Run the console.sh file in the forte4j-home/pointbase/client directory.
- On Microsoft Windows: Choose Start > Forte for Java EE > PointBase > Client Tools > Console or double-click the console.bat file in the forte4j-home/pointbase/client directory.

The Connect To Database dialog box appears, showing values for the PointBase driver to the default sample database.

Note – The Java 2 SDK, Enterprise Edition, version 1.2.2 (including the Reference Implementation) provided with this release is already configured to run a PointBase database named sample, which this tutorial uses. If you were to create a different PointBase database, you would have to add a specification for it to the jdbc.resources property in the J2EE_HOME/config/default.properties file. See the Getting Started Guide for details.

3. Click OK.

The PointBase Console is displayed.

4. Choose File > Open to display a file browser dialog box.

5. Use the file browser to find the rest_pb.sql file and click Open.

Look in your-work-dir/sampledir/tutorial/DiningGuide/Db. The contents of the rest_pb.sql file are copied to the SQL entry window.

On Solaris or Unix systems, your-work-dir is forte4j-home/ffjuser30.

6. Choose SQL > Execute All.

The message window confirms that the script was executed. (Ignore the initial messages beginning “Cannot find the table…” These appear because there are DROP statements for tables that have not been created yet. These DROP statements will be useful in the future if you want to rerun the script to initialize the tables.)

7. Test that you have created the table by clearing the SQL entry window (Window > Clear Input) and typing:

```sql
select * from Restaurant
```
8. Choose SQL > Execute.
Your console should display the Restaurant table.

9. Close the PointBase Console window.
Now, you’re ready to start up the J2EE software.

Starting Up the Java 2 SDK, Enterprise Edition Software

Before you start the Forte for Java IDE, you must first start the J2EE software.

1. Make sure the J2EE software is installed.
You can download a free copy of the 1.2.2 version of the J2EE software from the Forte for Java download page. This includes the J2EE Reference Implementation.

2. Make sure the J2EE_HOME environment variable is set to the J2EE root directory.
For example, set J2EE_HOME to c:\j2sdkee1.2.2.
3. **Make sure the JAVA_HOME environment variable is set to the JDK root directory.**
   For example, set JAVA_HOME to c:\jdk.

4. **Open a command window.**
   On Microsoft Windows, choose Start > Command Prompt.
   On Solaris-, Linux-, or other UNIX-based systems, open a terminal window.

5. **Navigate to the J2EE_HOME\bin directory and type j2ee -verbose.**
   You will see progress messages that the J2EE is starting. The last message should be, “J2EE server startup complete.”

6. **Verify that the J2EE is running by launching a web browser and navigating to the URL http://localhost:8000.**
   You should see the J2EE home page, as shown.

   ![J2EE 1.2.2 Default Home Page](image)
   
   You can now launch the Forte for Java IDE. Later, when you are ready to assemble and deploy, or simply to test, your application, you must tell the IDE that you are using this application server. See “Setting up the IDE to Run the Reference Implementation” on page 19.
Starting the Forte for Java IDE

You start the Forte for Java IDE by running the program executable, as described in the following sections, and more fully in the Getting Started Guide.

**Note** – The J2EE Reference Implementation must be installed on your system before you start the Forte for Java IDE.

Starting the IDE on Solaris, UNIX, and Linux Environments

After installation, a `runide.sh` script is in `forte4j-home/bin` directory. Launch this script by typing:

```
$ sh runide.sh
```

For ways to customize this script, see “Modifying the Session With Command-Line Switches” on page 17.

Starting the IDE on Microsoft Windows

After installation, a Forte for Java CE icon appears on your desktop. Double-click this icon to start the IDE. This icon is one of two available, depending on which mode you want to run the IDE in. The two modes available are represented in the `forte4j-home\bin` directory as the following executable files:

- `runidew.exe` launches the IDE without a console window. This launcher is used when you launch the IDE from the Start menu.
- `runide.exe` launches the IDE with a console window that includes standard error and standard output from the IDE. On the console, you can press Ctrl-Break to get a list of running threads or Ctrl-C to immediately terminate the program.
Alternatively, you can launch the IDE by choosing Start > Programs > Forte for Java EE > Forte for Java EE. Or you can run any of the executables from the command line. For example:

```
C:\> runidew.exe [switch]
```

Modifying the Session With Command-Line Switches

TABLE 1-5 describes the switches that you can use to modify how you launch the IDE. This information is also available from the Getting Started Guide, but is provided here for your convenience.

- **On Microsoft Windows systems**
  
  You can set options when running the IDE on the command line.

- **In Solaris, Linux, and other UNIX environments**
  
  You can modify the runide.sh file in the bin subdirectory of the installation directory, or you can create your own shell script that calls runide.sh with options.

**TABLE 1-5  runide Command-Line Switches**

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-classic</td>
<td>Uses the classic JVM.</td>
</tr>
<tr>
<td>-cp:p addl-classpath</td>
<td>Adds a class path to the beginning of the Forte for Java class path.</td>
</tr>
<tr>
<td>-cp:a addl-classpath</td>
<td>Adds a class path to the end of the Forte for Java class path.</td>
</tr>
<tr>
<td>-fontsize size</td>
<td>Sets the font size used in the GUI to the specified size.</td>
</tr>
<tr>
<td>-Jjvm-flags</td>
<td>Passes the specified flag directly to the JVM. (There is no space between -J and the argument.)</td>
</tr>
<tr>
<td>-jdkhome jdk-home-dir</td>
<td>Uses the specified Java 2 SDK instead of the default SDK.</td>
</tr>
<tr>
<td>-h or -help</td>
<td>Opens a GUI dialog box that lists the command-line options.</td>
</tr>
<tr>
<td>-hotspot or -classic or -client or -server or -native or -green</td>
<td>Uses the specified type of JVM.</td>
</tr>
</tbody>
</table>
Specifying Your User Settings Directory

By default, you run the Forte for Java software in multiuser mode, yet you store your individual projects, samples, and IDE settings in your own special directory. This enables individual developers to synchronize their development activities, while keeping their own personal work and preferences separate.

- In Solaris, UNIX, or Linux environments
  If you don’t explicitly specify a user settings directory with the `-userdir` command-line switch, user settings are located by default in `user-home/ffjuser30`.

- On Microsoft Windows systems
  At first launch of the Forte for Java IDE, you are prompted for the specification of this directory. Use a complete specification, for example, `C:\MyWork`.

  ![User Settings Directory Prompt](image)

  You can later specify a different user settings directory by using the `-userdir` command-line switch when launching the IDE.
Setting up the IDE to Run the Reference Implementation

To test your applications using the Reference Implementation, you must create a reference in the Server Registry of the Forte for Java IDE.

1. In the Forte for Java IDE, click the Explorer's Runtime tab.

2. Expand the Server Registry node and its Installed Servers subnode.
   You should see the J2EE Reference Implementation node.

3. Display the J2EE Reference Implementation node's properties.
   If the Properties window is open, simply select the J2EE RI node. Otherwise, right-click the J2EE RI node and choose Properties from the contextual menu.

4. In the Properties Window, change the value of the RIHome property to your J2EE_HOME directory.
   For example, type `c:\j2sdkee1.2.2`.

5. Right-click the J2EE Reference Implementation node and choose Add Server Instance.
   This adds a node that represents the server instance you started in Step 5, as shown.
The J2EE Reference Implementation is ready to accept deployed applications.

---

**Understanding the Forte for Java Directory Structure**

When you install the Forte for Java software, the subdirectories in TABLE 1-6 are included in your installation directory.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>beans</td>
<td>Contains JavaBeans™ components installed in the IDE.</td>
</tr>
<tr>
<td>bin</td>
<td>Includes Forte for Java launchers (as well as the ide.cfg file on Microsoft Windows installations).</td>
</tr>
<tr>
<td>docs</td>
<td>Contains the Forte for Java help files and other miscellaneous documentation. (Release notes are found under forte4j-home.)</td>
</tr>
<tr>
<td>iPlanet</td>
<td>Contains files used by the iPlanet plug-ins.</td>
</tr>
<tr>
<td>javadoc</td>
<td>The directory mounted by default in the IDE’s Javadoc repository. Both Javadoc provided with the IDE and user-created Javadoc are stored here.</td>
</tr>
<tr>
<td>lib</td>
<td>Contains JAR files that make up the IDE’s core implementation and the open APIs.</td>
</tr>
<tr>
<td>modules</td>
<td>Contains JAR files of Forte for Java modules.</td>
</tr>
<tr>
<td>pointbase</td>
<td>Contains the executables, classes, databases, and documentation for the PointBase Network Server database (if installed).</td>
</tr>
<tr>
<td>sources</td>
<td>Contains sources for libraries that might be redistributed with user applications.</td>
</tr>
<tr>
<td>system</td>
<td>Includes files and directories used by the IDE for special purposes.</td>
</tr>
<tr>
<td>teamware</td>
<td>Contains Forte for Java TeamWare module files (if installed).</td>
</tr>
</tbody>
</table>
When you launch the Forte for Java software in the default (multiuser) mode, the subdirectories in TABLE 1-7 are installed in your user settings directory. Most of them correspond to subdirectories in the Forte for Java home directory, and are used to hold your settings.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>beans</td>
<td>Contains user settings for JavaBeans components installed in the IDE.</td>
</tr>
<tr>
<td>javadoc</td>
<td>Contains user settings for Javadoc files installed in the IDE.</td>
</tr>
<tr>
<td>lib</td>
<td>Contains user settings for the system lib files.</td>
</tr>
<tr>
<td>modules</td>
<td>Contains modules downloaded from the Update Center.</td>
</tr>
<tr>
<td>sampledir</td>
<td>The directory mounted by default in the Filesystems pane of the Explorer. Objects you create in the IDE are saved here unless you mount other directories and use them instead.</td>
</tr>
<tr>
<td>sampledir/examples</td>
<td>Contains several example applications. If you download the Enterprise Edition-specific examples form the Forte for Java portal, you are instructed to put them under ~/sampledir/examples).</td>
</tr>
<tr>
<td>sampledir/tutorial</td>
<td>Contains several tutorial applications, including the EatersDigest tutorial described in this document and its database script.</td>
</tr>
<tr>
<td>system</td>
<td>Contains user settings for system files and directories. Among these are ide.log, which provides information useful when seeking technical support.</td>
</tr>
</tbody>
</table>
CHAPTER 2

Introduction to the Tutorial

In the process of creating the tutorial example application, you will learn how to build a simple J2EE application using Forte for Java, Enterprise Edition features.

This chapter describes the application you will build, first laying out its requirements, and then describing an architecture that fulfills the requirements. The final section describes how you use Forte for Java, Enterprise Edition features—the EJB Builder, the test application facility, and the Web Services module—to create the application.

This chapter is organized into the following sections:

- “Functionality of the Tutorial Application,” which follows
- “User’s View of the Tutorial Application” on page 25
- “Architecture of the Tutorial Application” on page 28
- “Overview of Tasks for Creating the Tutorial Application” on page 31

Functionality of the Tutorial Application

The tutorial application, EatersDigest, is a simple dining guide application that enables users to view a list of available restaurants and their features (name, cuisine type, neighborhood, address, phone number, a brief description, and a rating (1 - 5)), view a list of a selected restaurant’s customer reviews, and add a review to a restaurant’s record. The user uses a web browser to interact with the application’s interface as follows:

1. The user views a complete list of restaurants.
2. When the user finds an appealing restaurant, she can request a list of reviews submitted by customers for that restaurant.
3. The user can also write her own review of a restaurant and add it to the restaurant’s list of reviews.
Application Scenarios

The interaction of EatersDigest begins when the user points her web browser to the URL of a page that lists all the application’s restaurants, and ends when the user quits the application’s web site. The following scenarios describe possible interactions with the EatersDigest application, which illustrate interactions that happen within the application, and what its requirements are.

1. The user points her browser to the URL of the application’s Restaurant Listing page.
   The application displays the EatersDigest Restaurant Listing page, which displays a list of all restaurants, their names, cuisine type, location, phone number, a short review comment, and a rating from 1 to 5. Associated with each restaurant record are two buttons: View Customer Comments and Add Your Comment.

2. The user clicks a View Customer Comments button for a given restaurant.
   The application displays a All Customer Reviews By Restaurant Name page with a list of all the reviews submitted by customers for the restaurant. This page includes a Return to Restaurant List button.

3. The user clicks the Return to Restaurant List button on the Customer Review page.
   The application redisplays the Restaurant Listing page.

4. The user clicks the Add Your Comment button on the Restaurant Listing page.
   The application displays an Enter Customer Review page, consisting of a text entry area with three fields and a Submit button. The three fields are Restaurant Name (which displays the selected restaurant) and two text entry fields: Customer Name and Review.

5. The user types text into the Customer Name and Review fields of the Enter Customer Review page and clicks the Submit button.
   The application adds the customer’s name and review text to the CustomerReview database table, and redisplays the Restaurant Listing page.

6. The user clicks the View Customer Comments button on the previously selected restaurant to view her own review.
   The application redisplays the All Customer Reviews By Restaurant Name page showing all the reviews including the new one submitted by the user.
Application Functional Specification

The following items list the main functions for a user interface of an application that supports the application scenarios.

- A master view of all restaurant data through a displayed list
- A button on the master restaurant list page for retrieving all customer review data for a given restaurant
- A master view of all customer review data for a given restaurant
- A button on the master restaurant list page for adding a new review
- A button on the customer reviews-by-restaurant page for returning to the restaurant list page
- A text entry page for typing in a new customer name and new customer review for a displayed restaurant name
- A button on the text entry page for submitting the finished review data to the database

User’s View of the Tutorial Application

The user’s view of the application illustrates how the scenarios and the functional specification, described in “Functionality of the Tutorial Application” on page 23 are realized.

1. **Point a web browser to the application’s URL.**
   The EatersDigest Restaurant Listing JSP page is displayed, listing all the restaurants in the database.
This page displays the data from the Restaurant table you created in "Creating the Tutorial Database Tables" on page 11.

2. To view the customer reviews for a given restaurant, click the View Customer Comments button associated with that restaurant.

The All Customer Reviews By Restaurant Name page is displayed, listing all the reviews for the selected restaurant.
3. Return to the Restaurant Listing page by clicking the View Restaurant List button.

4. To add your own review to a restaurant, click the Add Your Comments button.
   The Enter Customer Review page is displayed, showing the name of the selected
   restaurant for which you can type in a name and review.
5. **Type in a customer name some text for the review and click the Submit button.**
   The text is added to the CustomerReview database table, and the Restaurant Listing page is redisplayed.

6. **To view your comment, click the same restaurant's View Customer Comments button.**
   The All Customer Reviews By Restaurant Name page is displayed.

![All Customer Reviews By Restaurant Name](image)

Continue to add and view comments.

7. **When you’re done, quit the application either by pointing your browser to a different web page or by quitting the browser.**

---

**Architecture of the Tutorial Application**

The heart of the tutorial application is the EJB Tier, which contains two entity type enterprise beans that represent the two EatersDigest databases (Restaurant and CustomerReview), and two detail classes that mirror the entity bean fields with getter and setter methods for each field. The detail classes are used to reduce the number of method calls to the entity beans when retrieving database data.

FIGURE 2-1 shows the EatersDigest application architecture.
In FIGURE 2-1, the connection between the application service components and the web service is depicted vaguely on purpose. This is because in this tutorial you are shown two methods for managing the interaction between the client and the entity beans. The first method, described in Chapter 3 (and depicted in FIGURE 2-2), uses a session enterprise bean to act as traffic manager. The second method, described in Chapter 4, uses the Web Services module functionality to create XML operations based on crucial application business methods, and then automatically generate a web service to manage them.

**Application Elements**

Briefly, the elements shown in FIGURE 2-1 are:

- An application service tier (an EJB tier)
  
  You build and test the EJB tier before you build anything else in the tutorial. It consists of:
  
  - Two entity enterprise beans that use container-managed persistence (CMP) to represent the two database tables of the application
  
  - Two detail classes to hold returned database records

  Chapter 3 version includes:
  
  - A stateless session enterprise bean to manage the requests from the client and to format the objects returned to the client.
Chapter 4 version includes:

- XML operations built upon the entity bean methods that fulfill the client’s requests.

- The web service tier

  Chapter 3 version—You generate a web module automatically when you test each enterprise bean with the Forte for Java test application facility. Servlets and JSP pages are automatically created in this web module that enable you to exercise each bean’s methods.

  Chapter 4 version—You generate a web service tier after you create XML operations based on selected methods in the EJB tier, then add references to these XML operations to the web service. Client JSP pages are automatically created that enable you to exercise the XML operations.

- The client

  The client component is a web browser that displays the application pages. In Chapter 5, you use provided client pages built upon the generated output pages created by the Web Services module in Chapter 4.

EJB Tier Details

The EJB tier of the EatersDigest application contains two entity-type enterprise beans and two detail classes. In Chapter 3, an additional component, a session bean, is used to manage the interaction between the client and the entity beans.

- Restaurant CMP EJB component

  The Restaurant bean is an entity bean that uses container-managed persistence (CMP) to represent the data of the Restaurant database table.

- Customerreview CMP EJB component

  Also a CMP-type entity bean, the Customerreview entity bean represents the data from the CustomerReview database table.

- RestaurantDetail class

  This component has the same fields as the Restaurant entity bean, plus getter and setter methods for each field for retrieving this data from the entity bean’s remote reference. Its constructor instantiates an object that represents the restaurant data. This object can then be formatted into a JSP or HTML page for the client to view.

- CustomerreviewDetail class

  This component serves the same function for the Customerreview entity bean that the RestaurantDetail class serves for the Restaurant entity bean.
DiningGuideManager session EJB component

This component is a stateless session bean that is used to manage the interaction between the client and the entity beans. It is introduced in Chapter 3 as a standard J2EE device for this purpose. It is not needed if you instead use the Web Services module to create a web service that uses XML operations based on the application’s business methods, as described in Chapter 4. With this session bean, the architecture of the application looks like FIGURE 2-2.

FIGURE 2-2  EatersDigest Application Architecture With Manager Session Bean

Overview of Tasks for Creating the Tutorial Application

The tutorial building process is divided into three chapters. In the first (Chapter 3), you create the EJB Tier, and use the IDE’s test application facility to test each enterprise bean as you work. In this chapter, you create a session bean to manage traffic, which is a common model if you were to create the web services and the client manually.

In the second chapter (Chapter 4), you learn that using the Web Services module to create a web service eliminates the need for the session bean you created previously. The Web Services module allows you to base your management of client-entity bean
traffic directly on the application’s business methods. You create XML operations for the critical business methods, and then let Web Services create the web service and basic client JSP pages for access to those operations.

In the final chapter (Chapter 5), you use provided client pages that are an improved version of the pages you generated in Chapter 4. The provided pages finish the application described in “User’s View of the Tutorial Application” on page 25.

Creating the EJB Components

In Chapter 3 you learn how to use Forte for Java features to:

■ Build entity and session beans quickly with the EJB Builder
■ Generate classes (with getter and setter methods) from a database schema
■ Use the test application facility to assemble a test J2EE application from enterprise beans
■ Add EJB references to a J2EE application
■ Deploy the test application to the J2EE Reference Implementation application server
■ Exercise enterprise bean methods from the test client page created by the test application facility.

Using the EJB Builder

The EJB Builder wizard automatically creates the various components that make up an enterprise bean, whether it’s a stateless or stateful session bean, or an entity bean with container-managed persistence (CMP) or bean-managed persistence (BMP). In Chapter 3, you create two CMP entity beans based on existing database tables, and a stateless session bean.

When you create the entity beans, you learn how to connect to a database during the creation process, and then generate an entity bean whose fields represent the table’s columns. The basic parts of the bean are generated into the Forte for Java Explorer with Java code already generated for the home interface, remote interface, bean class, and (if applicable) the primary key class. You learn how to edit and modify the bean properly by using the logical bean node, which represents the bean as a whole. You learn to add create, finder, and business methods using the EJB Builder’s GUI features.
Generating the Detail Classes

Although you don’t use Transparent Persistence in this tutorial, you can use its tools to streamline the creation of the detail classes. The detail classes must have the same fields as the entity beans. Transparent Persistence facilities automatically generate Java classes from database tables, resulting in classes that contain the database fields and getter and setter methods for each field. You can use this facility to create the detail classes without having to add persistence capability to the generated class.

Using the Test Application Facility

The Forte for Java IDE includes a facility for testing enterprise JavaBean components without your having to create a client for this purpose. This facility uses the J2EE Reference Implementation as the application server and deploys the enterprise bean as part of a J2EE application that includes a web module and client JSP pages. An HTML page coordinates these JSP pages so that, from a web browser, you can create an instance of the bean and then exercise its business methods.

You create test applications for all three of the enterprise beans separately. For the entity beans, the test application generates a J2EE application that contains a web module, which contains the automatically generated JSP pages for the client’s use from a web browser, and an EJB module for the entity bean. The session bean’s EJB module must also contain the EJB modules of the entity beans, because it calls methods on those entity beans. You add the entity bean references to the session bean’s EJB module using commands in the IDE.

When you test the session bean in a web browser, you can exercise all the application’s business methods. At the end of Chapter 3 are guidelines for using the test client apparatus to guide you if you want to create your own web service and client manually.

Creating the Tutorial’s Web Service

In Chapter 4 you learn how to use Forte for Java features to:

- Create the application’s response mechanism by wrapping the application’s business methods as XML operations
- Specify which entity business methods are to be called by each XML operation
- Combine multiple XML operations into a web service
- Create a J2EE application to contain the web service
- Generate the web service’s runtime classes and client pages
Creating XML Operations From Business Methods

Using functionality of the Web Services module, you create XML operations to represent responses to specific client requests. When a client passes a request, the web service executes an XML operation in response. An XML operation can call one or more methods on objects in the EJB tier, which in the EatersDigest application means the entity beans. In this manner, the web service replaces the managing function of the session bean you created in Chapter 3.

You create XML operations in the Forte for Java Explorer by using the New From Template wizard to specify the basic properties of the operation: name, package, method, and what the method returns. (If the method returns a collection, you specify the class the collection consists of.) Once created, you further define the operation in the Source Editor, which consists of a tree view of the XML operation, and windows that show XML format of the operation’s input and output.

In Chapter 4 you create four XML operations:

- **AllRestaurantsXMO**—Returns a collection of RestaurantDetail objects, one object for every restaurant in the database.
- **AllCustomerreviewsByRestaurantNameXMO**—Takes a restaurantName parameter and returns a collection of CustomerreviewDetail objects for the named restaurant.
- **AddCustomerReviewXMO**—Takes an input restaurantName, customerName, and review data and returns a CustomerreviewDetail object.
- **EnterCustomerReviewXMO**—Takes a string named element1, which it uses for the restaurantName field

The last XML is not based on a method from the application’s entity beans, but instead on a method from the Web Services library. It is used to pass a parameter.

Creating a Web Service

A web service is a logical entity you create in the Explorer using the New From Template wizard to define its name and package location. After you’ve created the web service, you add the XML operations to it. You can also add document files to the web service, and in Chapter 5 you add modified JSP pages to create the client.

Creating a J2EE Application for the Tutorial

In order to generate the web service’s runtime classes, you create a J2EE application and add the web service to it. This action makes the web service’s WAR and EJB JAR files available, so you can customize their properties. One property that you
customize is the Web Context property. You also add the web modules from the two entity beans that you created with the test application facility in Chapter 3. This completes the EatersDigest’s J2EE application, and you are ready to deploy it.

Deploying and Testing the Tutorial Application

You make sure that the J2EE Reference Implementation server is running, and then use an IDE command to deploy the EatersDigest application. This generates a JSP page for each XML operation and a welcome page that displays all the operations on one page. Also generated are classes for each XML operation and one for the web service.

The generated JSP pages contain input fields when an input parameter is required, and a Submit Query button, to execute the operation. You use these means to test each of the operations.

When Forte for Java Enterprise Service Presentation Toolkit is installed on the application’s web server (in this case, the J2EE Reference Implementation), developers can publish the XML operations to the Forte ESP registry from the Forte for Java IDE. This would allow web designers to use either Macromedia® DreamWeaver™ or Adobe® GoLive™, to which Forte ESP Toolkit plug-ins have been installed, to map the dynamic data of the XML operations to elements in their page layouts. The pages that Web Services generates can serve as a starting point for those pages.

You won’t need to install the Forte ESP Toolkit, because this tutorial does not cover Forte ESP development and does not step you through publishing the XML operations. (See “End Comments” on page 36 for information on where these steps are described.) However, you don’t need to publish XML operations, or even install the Forte ESP Toolkit, in order to take advantage of the Forte ESP tags at runtime.

Improving the Client Pages

In Chapter 5, you use four supplied JSP pages to create the tutorial client that functions as described in “User’s View of the Tutorial Application” on page 25. Because Forte ESP Toolkit has its own tutorial that demonstrates in detail how to modify the pages generated from the Web Services module, these steps are not specifically described in Chapter 5.

All you will need to do to finish the EatersDigest tutorial application is install the finished pages in the WebService/Documents directory, redeploy the EatersDigest application, and point your web browser to the application’s first page.

Two of the major modifications are described in detail, including how to pass a parameter when an HTML form is submitted and how to enter customer reviews.
End Comments

This tutorial application is designed to be a running application that illustrates the main features of Forte for Java, Enterprise Edition, while still brief enough for you to create in a short time (perhaps a day). This places certain restrictions on its scope, for example:

- There is no error handling
- There are no debugging procedures
- Publishing the web service’s XML operations is not described
- Using Forte ESP Toolkit to create the client is not described

A short tutorial document exists that describes how to publish XML operations and use Forte ESP to get dynamic data. This is the Hello Suite example that is included in the Enterprise Edition-specific examples that you can download from the Forte for Java portal. After you have unzipped the examples file into
your-user-dir/sampledir/examples, Hello Suite is found in
~/examples/ee/webservices/hellosuite. Supplement this information with detailed information from Building JSP Pages That Use XML Data Services.

Although the tutorial application described in this book is designed to be a simple application that you can complete quickly, you might want to import the entire application, view the source files, or copy and paste method code into methods you create. The EatersDigest application is accessible from within the IDE, in the DiningGuide folder under the your-user-dir/sampledir/tutorial file system.
Building the EJB Tier of the Tutorial Application

This chapter describes, step by step, how to create the EJB tier of the EatersDigest tutorial application. Along the way, you learn how to use the EJB Builder to create both entity and session beans, and the IDE’s test mechanism to test them with. The topics are organized as follows:

- “Overview of the Tutorial’s EJB Tier,” which follows
- “Setting Up Your System” on page 42
- “Creating Entity Beans With the EJB Builder” on page 42
- “Creating Detail Classes to View Entity Bean Data” on page 53
- “Testing Entity Beans” on page 62
- “Creating a Session Bean With the EJB Builder” on page 70
- “Testing the Session Bean” on page 77
- “Comments on Creating a Client” on page 79

By the end of this chapter, you will be able to run the whole EJB tier of the tutorial application as a deployed test application.

After you have created the EJB tier, you are free to create your own web services and client pages, or you can continue on to Chapter 4, to learn how to create the application’s web services using the Java™ Web Service Designer, and then to Chapter 5, to create the application’s client pages using the Forte for Java Enterprise Service Presentation Toolkit.

Overview of the Tutorial’s EJB Tier

In this chapter, you create the module that is the heart of the tutorial application, namely, its EJB tier. As you create each component, you test it using the IDE’s test application facility, which automatically creates a test web service and test client.
The EJB tier you’ll create will have:
- a Restaurant entity bean
- a CustomerReview entity bean
- a DiningGuideManager session bean
- a RestaurantDetail bean
- a CustomerReviewDetail bean

The Entity Beans

For more complete information about entity beans and how to create and program them, see Building Enterprise JavaBeans Components.

Entity beans provide a consistent interface to a set of shared data that defines a concept. In this tutorial, there are two concepts: restaurant and customer review. The Restaurant and CustomerReview entity beans that you’ll create represent the database tables you created in Chapter 1.

Entity beans can have either container-managed persistence (CMP) or bean-managed persistence (BMP). With a BMP entity bean, the developer must provide code for mapping the bean’s fields to the database table columns. With a CMP entity bean, the EJB execution environment manages persistence operations. In this tutorial, you’ll use CMP entity beans. Using the IDE’s EJB Builder wizard, you connect to the database, indicate which columns to map, and the wizard creates the entity beans mapped to the database.

The EJB Builder also creates the CMP entity bean’s framework, including the required home interface, remote interface, and bean class, as well as a logical node to organize and facilitate customizing the entity bean.

What you’ll have to do manually is add create methods for the entity beans to initialize the instance variables (the CMP fields). When you create a create method, the IDE automatically propagates the method to the home interface and a corresponding ejbCreate method to the bean class. When you add code to the create method to do the initializing work, the code is simultaneously added to the ejbCreate method, as well. You’ll also create a business method on each entity bean for testing purposes.
The Session Bean

For more complete information about session beans, see Building Enterprise JavaBeans Components.

Where entity beans represent shared data, session beans access data that spans concepts and is not shared. They can also manage the steps required to accomplish a particular task. Session beans can be stateful or stateless. A stateful session bean performs tasks on behalf of a client while maintaining a continued conversational state with the client. A stateless session bean doesn’t maintain a conversational state and is not dedicated to one client. Once a stateless bean has finished invoking a method for a client, it is available to service a request from a different client.

Client requests in the EatersDigest application include obtaining data on all the restaurants in the database, finding all the customer reviews for a given restaurant, and submitting a review for a given restaurant. These requests are not interrelated, and don’t require maintenance of a conversational state. For these reasons, the EatersDigest tutorial uses a stateless session bean to manage the different steps required for each request.

An action that the session bean performs repeatedly is building collections of restaurant and customer review records to satisfy a client’s request. This could be accomplished by adding getter and setter methods for each field onto the entity beans, but this would require calling a method for every field each time the session bean has to retrieve a row of the table. To reduce the number of method calls, this tutorial uses special helper classes, called detail classes, to hold the row data.

The Detail Classes

Detail classes have the same fields as the corresponding entity bean plus getter and setter methods for each field. When the session bean looks up an entity bean, it uses the corresponding detail class to create an instance of each remote reference returned by the entity bean. The session bean only has to call the detail class’s constructor to instantiate a row of data for viewing. In this way, the session bean can create a collection of row instances that can be formatted into an HTML page for the client to view.

FIGURE 3-1 shows graphically how the detail classes work.
FIGURE 3-1  Function of a Detail Class

1. Web container passes client’s request for all restaurant data to the DiningGuideManager session bean.

2. The Session bean calls the Restaurant entity bean’s findAll method to perform a lookup on the Restaurant entity bean.

3. The findAll method obtains all available remote references to the entity bean.

4. For each remote reference returned, the session bean calls the Restaurant bean’s getRestaurantDetail business method to fetch the RestaurantDetail class.

5. The getRestaurantDetail method returns a RestaurantDetail object, which is added to the collection.

6. Session bean returns a collection of all RestaurantDetail objects to the web container to format appropriately for the client to view.
Summary of Steps

The steps for creating the EJB tier are:

1. Create the entity beans.
   Create CMP entity bean skeletons with the EJB Builder wizard. Then add their create and finder methods and simple business methods for testing purposes.

2. Create detail classes from the database schemas.
   Capture a database schema and generate classes from its Restaurant and CustomerReview tables. These classes are automatically generated with getter and setter methods for each field.

3. Create get methods on the entity beans to get the detail classes.

4. Test the entity beans’ methods with the IDE’s test application facility.
   Viewing the automatically-generated test client in a web browser, exercise the create method to create an instance of the bean and make its business methods available. Then exercise the bean’s business methods.

5. Create the session bean.
   Create a stateless session bean skeleton with the EJB Builder, then modify the bean’s create method to perform a lookup on the entity beans. Then create getter methods for constructing collections of detail objects (from the detail classes) for each entity bean.

6. Use the test application facility to test the session bean.
   In the EJB module’s property sheet, add references to the CMP entity beans. Create a test application and add the EJB modules to the test application’s EJB module. Create an instance of the session bean using the test client and exercise its methods.
Setting Up Your System

Before you can begin work on the tutorial application, you must first perform the following steps (see Chapter 1 for details):

1. Start the PointBase network server.
2. Create the tutorial database tables.
3. Make sure a web browser is available on your system.
4. Start the J2EE server (the Java 2 SDK, Enterprise Edition containing the Reference Implementation).
6. Create an instance of the Reference Implementation in the Server Registry of the IDE.

Creating Entity Beans With the EJB Builder

Create two entity beans, Restaurant and Customerreview, to represent the two database tables you created in Chapter 1.

First, create a directory to mount as a Filesystem to contain the application. Next, create a package for the EJB Tier. Finally, create the entity beans within the package.

1. Somewhere on your file system, create a directory and name it DiningGuide.
   For example, you could create the DiningGuide directory under the user settings directory, MyWork.

2. In the Forte for Java IDE, use the File > Mount Filesystem command to find the DiningGuide directory and mount it.
   The c:\MyWork\DiningGuide directory is mounted in the Explorer window.

3. Right-click the new file system you just mounted and choose New Package.
   You’ll use this package to hold the EJB tier of the application.

4. Name the new package Data.
5. Right-click the new Data package and choose New > EJB > Entity Bean. The EJB Builder wizard appears, displaying the Entity EJB Type pane.

6. Select the Container Managed Persistence option.

7. Select the Select Database Table option (the default). The EJB Builder should look like this.

   ![EJB Builder wizard](image)

8. Click Next. This displays the Entity EJB Database Table pane.

9. Select the Select Database Table from Database Connection option and click Add Connection. The New Database Connection dialog box appears.

10. Enter the following values:
    
    Name: **PointBase Network Server**
    Driver: **com.pointbase.jdbc.jdbcUniversalDriver**
    Driver: **jdbc:pointbase://localhost/sample**
    User Name: **PUBLIC**
    Password: **PUBLIC** (displayed as asterisks)
The dialog box should look like this:

11. Click OK to accept the data and close the dialog box.

The new connection is displayed in the Entity EJB Database Table pane of the EJB Builder.

**Note** – If you did not get this result, it’s probably because you forgot to start the PointBase Network Server. See Step 1 of the procedure described in “Creating the Tutorial Database Tables” on page 11.

12. Open the new sample connection and the Tables folder and select the RESTAURANT table.
13. Click Next.

The Entity EJB Persistent Fields pane is displayed. You see a side-by-side display of the columns of the Restaurant database table and the corresponding persistent fields that the columns will be mapped to when the wizard creates the Restaurant entity bean. Accept all the default labels.

14. Click Next again.

The EJB Components page is displayed, listing the parts of the Restaurant bean that will be created. Notice that the EJB Builder wizard has automatically named the new entity bean with the same name as the database table.

15. Click Finish.

The new Restaurant entity bean and all its parts are created and displayed in the Explorer window.
Three of the parts are classes; the fourth part is the logical node, which is created to group all the elements of the enterprise bean together and facilitate working with them.

Now, create the Customerreview entity bean.


17. Select the Container Managed Persistence option.

18. Select the Select Database Table option.
   - If you have not exited the IDE since you created the Restaurant bean, the sample database is still connected. You can tell this by the icon . Go on to the next step.
   - If you have exited the IDE since you created the Restaurant bean, the sample database appears in the list, but its icon is “broken” ( ). To connect:
     a. In the list of databases, select the following item:
        jdbc:pointbase://localhost/sample[PUBLIC on PUBLIC]
     b. Click Connect to Database.
     c. In the Connect dialog box, type PUBLIC for the password and click OK.

19. Open the database node and the Tables folder, select the CUSTOMERREVIEW table and click Next.

20. Click Next on the Persistent Fields pane, and click Finish on the last pane (Components pane).
    The Customerreview entity bean is displayed in the Data package in the Explorer. Notice that there is an additional component: the CustomerreviewKey bean. This bean is automatically created when the entity bean has a composite primary key. (See TABLE 1-1 in Chapter 1 to confirm the composite primary key in this table.)
21. Open the `CustomerreviewKey` class, then open the `Constructors` node.

The EJB Builder has automatically created two constructors: a no-argument constructor, and an overloaded constructor that sets the two composite primary key fields.

Now, create the entity bean methods.
Creating create Methods for CMP Entity Beans

Create the create methods for both entity beans, adding parameters and code to initialize the fields of the beans' instances.

1. In the Forte for Java Explorer, right-click the Restaurant logical node (the bean icon)

2. Choose New Create Method from the contextual menu.

   The New Create Method dialog box appears.

3. Using the Add button, create seven new parameters, one for each column of the Restaurant table:
   - restaurantname (java.lang.String)
   - cuisine (java.lang.String)
   - neighborhood (java.lang.String)
   - address (java.lang.String)
   - phone (java.lang.String)
   - description (java.lang.String)
   - rating (java.lang.Integer)

   **Note** – The order you create these parameters becomes important when you test the bean with the test application facility. Please create them in the order given here.

   Keep the two exceptions that are created by default.

4. Click OK.

   This propagates a create method under the RestaurantHome interface, and an ejbCreate method under the Restaurant bean class (RestaurantEJB). A related ejbPostCreate method is also added to the bean class.

5. Open the Restaurant logical node and the Create Methods folder, then double-click the create method.

   This opens the Text Editor right to the.ejbCreate method of the bean.
6. Add the following code to the body of the ejbCreate method to initialize the fields of the bean instance:

```java
    if (restaurantname == null) {
        throw new CreateException("The restaurant name is required.");
    }
    this.restaurantname = restaurantname;
    this.cuisine = cuisine;
    this.neighborhood = neighborhood;
    this.address = address;
    this.phone = phone;
    this.description = description;
    this.rating = rating;
    return null;
}
```

When the ejbCreate method is called, it creates a new record in the database, based on the container-managed fields of this bean.

Now create the Customerreview's create method.

7. Right-click the Customerreview logical node (the bean icon 🍔) and choose New Create Method.

8. Use the Add button to create three parameters, one for each column of the CustomerReview table:
   - restaurantname (java.lang.String)
   - customername (java.lang.String)
   - review (java.lang.String)

   **Note** – As in Step 3, above, please create these parameters in this order.

9. Click OK.

10. Open the Customerreview logical node and the Create Methods folder, then double-click the create method.
    This opens the Text Editor right to the ejbCreate method of the bean.
11. Add the following code to the body of the `ejbCreate` method to initialize the fields of the bean instance:

```java
public CustomerreviewKey ejbCreate(java.lang.String restaurantname, java.lang.String customername, java.lang.String review) throws CreateException {
    if ((restaurantname == null) || (customername == null)) {
        throw new CreateException("Both the restaurant name and customer name are required.");
    }
    this.restaurantname = restaurantname;
    this.customername = customername;
    this.review = review;

    return null;
}
```

When the `ejbCreate` method is called, it creates a new record in the database, based on the container-managed fields of this bean.

Now, create finder methods on both entity beans that will locate all or selected instances of each bean in the context.

Creating finder Methods on the Entity Beans

Create a `findAll` method on the `Restaurant` bean, to locate all Restaurant data. Also create a `findByRestaurantName` on the `Customerreview` bean to locate Customerreview data for a given restaurant.

1. Right-click the `Restaurant` logical node and choose New Finder Method.

2. In the New Finder Method dialog box, enter the following values:
   - Name: `findAll`
   - Return type: `java.util.Collection`
   - Accept the default exceptions.

3. Click OK.
   The new `findAll` method is created in the `Home` interface of the `Restaurant` bean.

Now, create a finder method on the `Customerreview` bean.

4. Right-click the `Customerreview` logical node and choose New Finder Method.
5. In the New Finder Method dialog box, enter the following values:
   Name: findByNameRestaurantName
   Return type: java.util.Collection

6. Add a parameter with the following values:
   Field name: restaurantname
   Type: java.lang.String
   Accept the default exceptions.

7. Click OK.
   The new findByNameRestaurantName method is created in the Home interface of the Customerreview bean.

8. Open the property sheet for the Customerreview.findByNameRestaurantName method.
   If the Properties window is not already open, right-click the method and choose Properties from the contextual menu.

9. Click the J2EE RI tab of the property sheet.

10. Click the value field for the SQL statement, then click the browse (...) button to view the whole SQL statement.
    The WHERE clause is incomplete.

11. Modify the statement as follows:

```
    SELECT "customername", "restaurantname" FROM 
    "CustomerreviewEJBTable" WHERE "restaurantname" = ?1
```

Which numeral you use is determined by the position of the restaurantname parameter in the finder method (there’s only one in this case, so this is “1”).

**Creating Business Methods for Testing Purposes**

Create a business method for each entity bean that returns a value of one of its parameters, so that you can test the beans later.

1. Right-click the Restaurant logical node and choose New Business Method.
   The New Business Method dialog box is displayed.

2. Type these values:
   Name: getRating
   Return Type: java.lang.Integer
3. Click OK.
   This propagates a getRating method declaration under the Restaurant remote
   interface, and under the RestaurantEJB class.

4. Open the Restaurant logical node, open the Business Methods folder, then
double-click the getRating method.

5. Add the following code to the body of the getRating method:

   ```java
   public java.lang.Integer getRating() {
       return rating;
   }
   ```

   When the getRating method is used, it returns the value in the rating column of a
   selected restaurant record.

6. Right-click the Data package and choose Build All from the contextual menu.
   The entire package should compile without errors. Now you are ready to verify this
   entity bean.
   Next, create a similar method for the Customerreview bean.

7. Right-click the Customerreview logical node and choose New Business Method.

8. In the New Business Method dialog box, type these values:
   Name: getReview
   Return Type: java.lang.String

9. Click OK.
   This propagates a getReview method declaration under the Customerreview
   remote interface, and under the CustomerreviewEJB class.

10. Open the Customerreview logical node, open the Business Methods folder,
then double-click the getReview method.

11. Add the following code to the body of the getReview method:

   ```java
   public java.lang.String getReview() {
       return review;
   }
   ```
Compiling Entity Beans

Now you can compile the entity beans.

1. **Right-click the Data package and choose Compile All or right-click each logical bean and choose Compile EJB Classes.**
   
   Both beans should compile without errors.

2. **Check that the icons in the Explorer no longer have “uncompiled” marks.**

![Explorer screenshot](image)

Creating Detail Classes to View Entity Bean Data

As discussed in “The Detail Classes” on page 39, this tutorial uses special classes, called detail classes, as a mechanism for holding row data for viewing and reducing method calls to the entity beans. These classes must have the same fields as the corresponding entity bean, access methods for each field, and a constructor that sets each field. You could create these classes by hand, but the Forte for Java IDE includes a facility that does most of this for you.

You can use the mechanism that generates persistence-capable Java classes for Transparent Persistence to generate the detail classes. You first import the database schema, then generate the Java classes from it, without using the persistence-generating features.
Generating Classes From a Database Schema

To generate the detail classes:

1. In the Forte for Java Explorer, right-click the Data package and choose New > Databases > Database Schema.
   The New Object Name pane of the New From Template wizard appears.

2. Name the new schema DiningSchema and click Next.
   A pane for the connection information appears.

3. Check the Existing Connection option and select the connection for the sample database from the list.
   If you have restarted the IDE since you created the entity beans, a dialog appears for logging into the connection. Type PUBLIC and click OK to dismiss the dialog box.

4. Click Next.
   A pane appears that is used for selecting the tables you want to capture. You want only the CUSTOMERREVIEW and RESTAURANT tables.

5. Select the CUSTOMERREVIEW and RESTAURANT tables (use the Control key for multiple selection) and click the Add button.
   These two tables are listed in the Selected Tables and Views window, as shown.
6. **Click Finish.**

   A progress window appears and when it closes, the new database schema is displayed in the Forte for Java Explorer window.

7. **Right-click the DiningSchema node ( ) and choose Generate Java from the contextual menu.**

   The choose Target Location pane of the Generate Java wizard appears.
8. Select the Data package.

9. Click Next.

10. On the Customize Options pane, set the following options:

    Make Generated Classes Persistence-Capable: unchecked
    Make Generated Classes Implement java.io.Serializable: checked
    Add Java "transient" Modifier to: No Fields
    For Each Foreign Key Generate: No Relationship Fields

    The wizard page should look like this:
11. **Click Next.**
The Table Selection pane of the wizard appears.

12. **Click the Add All Tables button.**
The RESTAURANT and CUSTOMERREVIEW tables move to the Selected Tables and Views window of the wizard. Now, edit the Java Class labels to create the detail class names.

13. **Click on Restaurant label under Java Class, change the name to RestaurantDetail, and press Enter.**

14. **Similarly, change Customerreview to CustomerreviewDetail and press Enter.**

15. **Click Next.**
The wizard displays a summary of the tables it will generate. These should be CustomerreviewDetail and RestaurantDetail.
16. **Click Generate.**

The IDE displays a feedback window showing which classes were generated.

![Feedback window](image)

17. **Close the window.**

The `RestaurantDetail` and `CustomerreviewDetail` classes appear in the Explorer. Now you are ready to edit the detail classes.

## Editing the Generated Classes

The main modification to make to the generated classes is to edit their constructors to instantiate each field. But first, you have to adjust the generated type of the `rating` field.

### Adjusting the Type for the `rating` Field

The `rating` field of the `RestaurantDetail` class has a different type than the same field of the `Restaurant` entity bean. The `Restaurant.rating` field is an Integer, whereas, due to a rule of conversion for Transparent Persistence that does not concern this case, the `RestaurantDetail.rating` field is a Short. Therefore, you must manually change the type of that field to an Integer, to match the `Restaurant` entity bean’s field. You must make the same change wherever this field is cited in other methods.

1. **In the Forte for Java Explorer, select the `rating` field of the `RestaurantDetail` class.**

2. **Open the property sheet for the `rating` field.**

   If the Properties window is open, simply select the field.

   If the Properties window is not open, select the field and choose Properties from the contextual window.
3. **For the Type option**, type `java.lang.Integer`. Notice that the type for this field automatically changes in the text editor.

4. **Change the data type in the `getRating` and `setRating` methods** as follows:

   ```java
   public java.lang.Integer getRating() {
       return rating;
   }
   public void setRating(java.lang.Integer rating) {
       this.rating = rating;
   }
   ``

5. **And finally, change the data type in the `readObject` method**:

   ```java
   private void readObject(java.io.ObjectInputStream in) throws java.io.Exceptio, ClassNotFoundException {
       restaurantname = (String)in.readObject();
       cuisine = (String)in.readObject();
       neighborhood = (String)in.readObject();
       address = (String)in.readObject();
       phone = (String)in.readObject();
       description = (String)in.readObject();
       rating = (java.lang.Integer)in.readObject();
   }
   ``

**Creating the Constructors**

To create the constructors for the detail classes that instantiate the class fields:

1. **Expand the RestaurantDetail bean**, right-click the `RestaurantDetail` class node, and choose New > Constructor.

   The New Constructor wizard appears.

2. **Add the following method parameters and click OK**:

   ```java
   java.lang.String restaurantname
   java.lang.String cuisine
   java.lang.String neighborhood
   java.lang.String address
   java.lang.String phone
   java.lang.String description
   java.lang.Integer rating
   ```
3. Add the following code to the body of the RestaurantDetail constructor to initialize the fields:

```java
public RestaurantDetail(java.lang.String restaurantname,
    java.lang.String cuisine, java.lang.String neighborhood,
    java.lang.String address, java.lang.String phone,
    java.lang.String description, java.lang.Integer rating){
    System.out.println("Creating new RestaurantDetail");
    this.restaurantname = restaurantname;
    this.cuisine = cuisine;
    this.neighborhood = neighborhood;
    this.address = neighborhood;
    this.phone = phone;
    this.description = description;
    this.rating = rating;
}
```

4. Similarly, add a constructor to the CustomerreviewDetail class with the following parameters:

```java
public CustomerreviewDetail(java.lang.String restaurantname,
    java.lang.String customername, java.lang.String review){
    System.out.println("Creating new CustomerreviewDetail");
    this.restaurantname = restaurantname;
    this.customername = customername;
    this.review = review;
}
```

5. Add the following code to the body of the CustomerreviewDetail constructor to initialize the fields:

```java
public CustomerreviewDetail(java.lang.String restaurantname,
    java.lang.String customername, java.lang.String review){
    System.out.println("Creating new CustomerreviewDetail");
    this.restaurantname = restaurantname;
    this.customername = customername;
    this.review = review;
}
```

6. Right-click the Data package and choose Compile All.

   The package should compile without errors.

   Now, create `get` methods on the entity beans to retrieve instances of the detail classes.
Creating get Methods on the Entity Beans

Create methods on each entity bean that returns an instance of its corresponding detail class.

1. In the Forte for Java Explorer, right-click the Restaurant (EJB) logical node and choose New Business Method.

2. In the New Business Method dialog box, type getRestaurantDetail in the Name field.

3. For the return type, use the Browse button to select the RestaurantDetail class. Data.RestaurantDetail appears in the Return Type field.

4. Click OK to create the method.

5. Open the method in the text editor and add the following code:

```java
public Data.RestaurantDetail getRestaurantDetail() {
    return new RestaurantDetail(restaurantname, cuisine, neighborhood, address, phone, description, rating);
}
```

6. Similarly, add a business method to the Customerreview entity bean with the following values:

   Name: getCustomerreviewDetail
   Return Type: Data.CustomerreviewDetail

7. Open the method in the text editor and add the following code:

```java
public Data.CustomerreviewDetail getCustomerreviewDetail() {
    return new CustomerreviewDetail(restaurantname, customername, review);
}
```

8. Right-click the Data package and choose Compile All.
   The entire package should compile without errors.

You have finished creating the entity beans of the tutorial application and their detail class helpers. The next step is to test the beans.
Testing Entity Beans

The Forte for Java IDE includes a mechanism for testing enterprise beans without having to create your own client. This feature uses the J2EE Reference Implementation as the application server and deploys your enterprise bean as part of an application that uses JavaServer Page technology. The facility creates a page viewable in a web browser where you can create instances of the bean and exercise its create and business methods.

You’ll use this test mechanism to exercise the Restaurant bean’s create and getRating methods.

Note – The SQL statements generated for the test client are subject to the protocol of the Reference Implementation, namely, that all table and column names are enclosed in quotation marks, and the string “EJBTable” is added to table names. When you ask a bean instance in the test application to look for a table named Restaurant, the application, because of this RI protocol, causes the bean instance to look for a table named “RestaurantEJBTable” instead. This is why the rest_pb.sql created such a table (see “Creating the Tutorial Database Tables” on page 11).

Creating a Test Client for an Entity Bean

First add PointBase information to the Reference Implementation properties of the Restaurant bean so that the test application can find the database and log into it. Then create the test client.

1. Select the Restaurant logical node and examine its Properties window.

   If the Properties window is not already open (it is usually under the Explorer window), then right-click Restaurant (EJB) and choose Properties from the contextual menu. If you want the Properties window to stay up, choose Window > Windows > Browsing > Properties Window.

2. Select the J2EE RI tab of the Properties window.

   Note – If there is no J2EE RI tab on the Properties window, it’s because there’s no instance of the Reference Implementation in the Server Registry. See “Setting up the IDE to Run the Reference Implementation” on page 19 for the procedure.
3. Specify the database connection by typing these values on the following properties:

Data Source JNDI Name: jdbc/Pointbase
Data Source Password: PUBLIC
Data Source UserName: PUBLIC

**Note** – Make sure to spell “Pointbase” with an initial capital only. This is the way it is specified in the J2EE_HOME/config/default.properties file, and must be exactly the same for this property. Also, you must press Enter after you type in the password. Using the mouse to move to another field will not set this value. (This is only true of the Data Source Password field.)

4. Turn off the option to create and drop the table automatically by selecting False on these properties:

Create table when deployed: false
Drop table when undeployed: false

In “Creating the Tutorial Database Tables” on page 11, you used the database script to create the “TablenameEJBTable” version of the tables, which RI would otherwise create and drop whenever you deploy or undeploy the application. If you set these fields to true, you would need to create new table data each time you deploy.

If you’re curious about the rest of the properties on the J2EE RI tab, press your Help key while focussed on this tab to access context-sensitive help information.

5. Save your work with File > Save All.

6. Right-click the Restaurant logical node and choose New EJB Test Application.

The EJB Test Application wizard appears. For context-sensitive help information on this wizard, press your Help key.
7. Accept all default values. 
   The wizard’s window should look like this:

![Create a new EJB Test Application](image)

8. Click OK. 
   A progress monitor appears briefly. 

9. View the resulting test objects in the Explorer. 
   The Test Application operation has created an EJB module (containing the Restaurant entity bean), a web module (containing a number of JSP pages and helper Java classes), and a J2EE application (containing the EJB module and web module).
The J2EE application created by the IDE consists of a web module and an EJB module, which you can see by opening the Restaurant_TestApp:

10. Select the EJB module (Restaurant_EJBModule) in the Explorer and display its property sheet.
    If the Properties Window is not already open, choose View > Properties.

11. Click on the browse button (…) of the Transaction Settings input field.
    You may have to scroll down the properties list to find this property.

12. Make sure the Trans-attribute value is Required, and click OK.

13. Right-click the Restaurant_TestApp J2EE application node and choose Deploy from the contextual menu.
You can read the progress of deployment in the IDE’s output window, on the RI Application tab. The text there should show that a web client named Restaurant_TestApp was deployed. This is the client you use to test the Restaurant entity bean.

To verify that the application is deployed, open the J2EE command window. The last statement should be “Application Restaurant_TestApp deployed.”

**Tip** – If your deployment failed, check whether you have set the IDE to run the J2EE Reference Implementation properly, especially whether the RIHome property is set to your J2EE_HOME value. See Step 4 of the procedures under “Setting up the IDE to Run the Reference Implementation” on page 19. Set it correctly, then redeploy.

Now, test the Restaurant entity bean.

**Testing an Entity Bean With the Test Client**

On the test client’s web page, use the create method of the Restaurant bean’s home interface to create an instance of the bean; then test a business method (in this case, getRating) on that instance.

Point your web browser to the test application to start the application.

1. **From your operating system, open a web browser and point it to the following URL:**

   http://localhost:8000/Restaurant_TestApp

   Your browser displays the test client.
Chapter 3  Building the EJB Tier of the Tutorial Application

2. **Create an instance of the Restaurant bean by invoking the create method.**

   Under the heading “Invoke Methods on Data.RestaurantHome” is the create method with seven fields under it. The fields are not named, but you can deduce what they are by their order, which is the same order you created them in (see Step 3 under “Creating create Methods for CMP Entity Beans” on page 48).

   **Note** – To see quickly what the parameters are, find the definition of the Restaurant.create method in the text editor; the fields are in the same order as in the definition.

   **Tip** – If you want the parameters to appear in a different order, right-click the Restaurant.create method node in the Explorer window and choose Customize. In the Customizer dialog box, rearrange the parameters by selecting and clicking the Up and Down buttons. Then redeploy the test application by right-clicking its node in the Explorer and choosing Deploy.

   Type any data you like into the fields, for example (your field order may be different):

   **Invoke Methods on Data.RestaurantHome**

   Data.RestaurantHome

   ![Invoke Methods on Data.RestaurantHome](image)

   3. **Click the Invoke button next to the create method.**

   The deployed test application adds the records you created to the test database. The new Restaurant instance is listed by its restaurantname value in the upper left, and new data objects are listed in the upper right, as shown.
4. Test the `findAll` method of the `Restaurant` bean by clicking the Invoke button next to it.

The results area should look like this:

<table>
<thead>
<tr>
<th>Results of the Last Method Invocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's House of Fish</td>
</tr>
<tr>
<td>Method Invoked: <code>create</code></td>
</tr>
<tr>
<td>Parameters:</td>
</tr>
<tr>
<td>Joe's House of Fish</td>
</tr>
<tr>
<td>American</td>
</tr>
<tr>
<td>Alameda Island</td>
</tr>
<tr>
<td>1234maner Sq Loop</td>
</tr>
<tr>
<td>510-222-3333</td>
</tr>
<tr>
<td>Interesting variety</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Notice that three items were returned. This demonstrates that the new database record you created in Step 3 was added to the two you created in Chapter 1.
5. **Test the `findByPrimaryKey` method by typing in Bay Fox and clicking the Invoke button next to the method.**
   The results area should show that the Bay Fox record was returned.
   Now, test the entity bean’s business methods.

6. **Select the instance for Joe’s House of Fish listed under `Data.RestaurantHome` in the instances list (upper left).**
   The `getRating` method is now listed under the Invoke Methods area.

7. **Click the Invoke button next to the `getRating` method.**
   The results of this action are listed in the Results area and should look like this:

<table>
<thead>
<tr>
<th>Results of the Last Method Invocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>Method Invoked: <code>getRating()</code></td>
</tr>
<tr>
<td>Parameters:</td>
</tr>
<tr>
<td><code>null</code></td>
</tr>
</tbody>
</table>

   This demonstrates that you have created a new record in the database and used the `getRating` method to retrieve the value of one of its fields.

   Continue testing by selecting created objects and invoking their methods. For example, if you select one of the `Data.RestaurantDetail` objects, you can invoke its getter methods to view its data, or its setter methods to write new data to the database.

8. **When you are done testing, you can stop the test client by pointing your web browser at another URL or by exiting the browser.**
   You will use the Restaurant_TestApp in Chapter 4. You will also need a Customerreview_TestApp in Chapter 4.

9. **Create a test client for the Customerreview entity bean and test it redoing all the steps appropriately for the Customerreview bean, starting with “Creating a Test Client for an Entity Bean” on page 62.**

   **Note** – Even though it’s repetitive, you must not skip Step 9, because you’ll need the Customerreview_EJBMModule in Chapter 4.

   If you ever want to delete the test application, unmount the web application first, then delete the J2EE application, the web module, and the EJB module under the Data package.
   You are now ready to create the session bean.
Creating a Session Bean With the EJB Builder

Create a stateless session bean to manage the conversation between the client and the entity beans.

1. In the Forte for Java Explorer, right-click the Data package and choose New > EJB > Session Bean.
   The EJB Builder wizard appears, displaying the Session EJB Type pane.

2. Specify the following values:
   State: Stateless
   Transaction Type: Container Managed Transaction
   Use with Transparent Persistence: unchecked

3. Click Next.
   The EJB Components pane of the wizard appears, listing all the components that will be created for this session bean.

4. In the Name field, type DiningGuideManager.
   Notice that the names of all the components change as you type this name.

5. Click Finish.
   The new DiningGuideManager session bean is displayed in the Explorer.
Coding a Session Bean’s `create` Method

The `create` method was created when you created the `DiningGuideManager` session bean, so you only need to modify it.

Create methods of stateless session beans don’t have arguments because session beans don’t maintain an ongoing state that needs to be initialized. The `create` method of this session bean starts by creating an `InitialContext`, which it then uses to get the required remote references.

1. **Open the** `DiningGuideManager`’s `create` method **in the text editor**.

2. **Begin coding the method with a JNDI lookup for a remote reference to the** `RestaurantHome` **interface.**

```java
public void ejbCreate(){
    System.out.println("Entering DiningGuideManagerEJB.ejbCreate()");
    Context c = null;
    Object result = null;

    if (this.myRestaurantHome == null) {
        try {
            c = new InitialContext();
            result = c.lookup("Restaurant");
            myRestaurantHome = (RestaurantHome)javax.rmi.PortableRemoteObject.narrow (result,
                               RestaurantHome.class);
        } catch (Exception e) {System.out.println("Error: "+ e); }
    }
}
```
3. Under the preceding code, add a similar JNDI lookup for the CustomerreviewHome interface.

```java
Context crc = null;
Object crresult = null;

if (this.myCustomerreviewHome == null) {
    try {
        crc = new InitialContext();
        result = crc.lookup("Customerreview");
        myCustomerreviewHome =
            (CustomerreviewHome)javax.rmi.PortableRemoteObject.narrow(result,
                CustomerreviewHome.class);
    } catch (Exception e) {
        System.out.println("Error: "+ e);
    }
}
```

Add this statement under the other import statements at the top of the file. You must import javax.naming because it contains the lookup method you just used.

```java
import javax.ejb.CreateException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.SessionSynchronization;
import javax.naming.*;
```

5. Finally, declare the myRestaurantHome and myCustomerreviewHome fields.
Add these declarations to the description of the DiningGuideManagerEJB session bean above the ejbcreate method.

```java
public class DiningGuideManagerEJB implements SessionBean {
    private SessionContext Context;
    private RestaurantHome myRestaurantHome;
    private CustomerreviewHome myCustomerreviewHome;

    Next, create the DiningGuideManager’s business methods.
Creating Business Methods to Get the Detail Data

The DiningGuideManager bean needs a method that retrieves all Restaurant data when it receives a request from the client to see the list of restaurants. It needs another method to retrieve Customerreview data for a specific restaurant when the client requests a list of customer reviews. Create the getAllRestaurants and getCustomerreviewsByRestaurant methods to do these things.

1. Right-click the DiningGuideManager logical node and choose New Business Method.

2. In the New Business Method dialog box, type in the following values:
   - Name: getAllRestaurants
   - Return Type: java.util.Vector

3. Click OK.

The method shell is created in the DiningGuideManager session bean's business methods.
4. Open the method in the text editor and add the following code:

```java
public java.util.Vector getAllRestaurants() {
    System.out.println("Entering DiningGuideManagerEJB.getAllRestaurants()");
    java.util.Vector restaurantList = new java.util.Vector();
    try {
        java.util.Collection rl = myRestaurantHome.findAll();
        if (rl == null) { restaurantList = null; } 
        else {
            RestaurantDetail rd;
            java.util.Iterator rli = rl.iterator();
            while ( rli.hasNext() ) {
                rd = ((Restaurant)rli.next()).getRestaurantDetail();
                System.out.println(rd.getRestaurantname());
                System.out.println(rd.getRating());
                restaurantList.addElement(rd);
            }
        }
    } catch (Exception e) {
        System.out.println("Error in DiningGuideManagerEJB.getAllRestaurants():" + e);
    }
    System.out.println("Leaving DiningGuideManagerEJB.getAllRestaurants()");
    return restaurantList;
}
```

This code gets an instance of RestaurantDetail for each remote reference of the Restaurant bean in the context, adds it to a vector called restaurantList and returns this vector.

Now, create a similar method to get a list of customer reviews.

5. Right-click the DiningGuideManager logical node and choose New Business Method.

6. In the New Business Method dialog box, type in the following values:
   Name: getCustomerreviewsByRestaurant
   Return Type: java.util.Vector

7. Add the following parameter:
   Field Name: restaurantname
   Type: java.lang.String

8. Click OK.
   The method shell is created in the DiningGuideManager session bean.
9. Find the method in the text editor and add the following code:

```java
public java.util.Vector getCustomerreviewsByRestaurant(java.lang.String restaurantname) {
    System.out.println("Entering DiningGuideManagerEJB.getCustomerreviewsByRestaurant()");
    java.util.Vector reviewList = new java.util.Vector();
    try {
        java.util.Collection rl = myCustomerreviewHome.findByRestaurantName(restaurantname);
        if (rl == null) { reviewList = null; }
        else {
            CustomerreviewDetail crd;
            java.util.Iterator rli = rl.iterator();
            while ( rli.hasNext() ) {
                crd =((Customerreview)rli.next()).getCustomerreviewDetail();
                System.out.println(crd.getRestaurantname());
                System.out.println(crd.getCustomername());
                System.out.println(crd.getReview());
                reviewList.addElement(crd);
            }
        }
    } catch (Exception e) {
        System.out.println("Error in DiningGuideManagerEJB.getCustomerreviewsByRestaurant(): "+ e);
    }
    System.out.println("Leaving DiningGuideManagerEJB.getCustomerreviewsByRestaurant()");
    return reviewList;
}
```

Similar to the getAllRestaurants code, this method retrieves an instance of CustomerreviewDetail for each remote reference of the Customerreview bean in the context, adds it to a vector called reviewList and returns this vector.

**Adding EJB References**

When you deploy a session bean, the bean’s properties must contain references to any entity beans whose methods the session bean calls. You add them to the session bean now, because you can’t add them after the bean once it has been assembled into an EJB module.

1. **In the Forte for Java Explorer, select the DiningGuideManager bean’s logical node.**
2. Display the bean’s property sheet.
   If the Properties window is not already visible, right-click the bean and choose
   Properties from the contextual menu.

3. Click the References tab of the property window.

4. Click the EJB References field and then the browser (...) button.
   The EJB References dialog box appears.

5. Click the Add button.
   The Add EJB Reference dialog box appears.

6. Enter the following values for the Restaurant entity bean and click OK:
   Reference Name: Restaurant
   Description: (you can leave blank)
   Referenced EJB Name: Restaurant
   Type: Entity
   Home Interface: Data.RestaurantHome
   Remote Interface: Data.Restaurant

   **Tip** – If you use the Browse button to find the Referenced EJB Name, the Home and
   Remote Interface fields are automatically filled.

7. Similarly, add a reference to the Customerreview entity bean.
   The EJB References dialog box should look like this:

8. Choose File > Save All.
   You have now completed the EJB Tier of the tutorial application and are ready to test
   it. As when you tested the entity beans, the IDE’s test application facility creates a
   web tier and JSP pages that can be read by a client in a browser.
Testing the Session Bean

Use the IDE’s test application facility to test the `DiningGuideManager` session bean. This will test the whole EJB tier, because the session bean’s methods provide access to methods on all the other components of the tier.

Creating a Test Client for a Session Bean

Create a test application from the `DiningGuideManager` bean. Then add the two entity beans to the EJB module.

**Note** – Make sure that PointBase information has been added to the J2EE RI properties of the `Restaurant` and `Customerreview` beans. Refer to Step 1 through Step 5 under “Creating a Test Client for an Entity Bean” on page 62.

1. **Right-click the DiningGuideManager logical node and choose New EJB Test Application.**
   The EJB Test Application wizard appears.

2. **Accept all default values and click OK.**
   A progress monitor appears briefly.

3. **View the resulting test objects in the Explorer.**
   The Test Application operation has created the `DiningGuideManager_EJBModule`, the `DiningGuideManager_WebModule` (and also mounted it), and the `DiningGuideManager_TestApp`, which is the J2EE application that contains the EJB module and web module.
   The EJB module contains only the `DiningGuideManager` bean, so you must add the two entity beans.

4. **Right-click the EJB module (DiningGuideManager_EJBModule) and choose Add EJB.**
   The Add EJB to EJB Module window appears.

5. **Find the Restaurant entity bean, select it, and click OK.**

6. **Repeat Step 4 and Step 5 for the Customerreview entity bean.**
   The `DiningGuideManager_EJBModule` should look like this:
7. Choose File > Save All

8. Right-click the DiningGuideManager_TestApp J2EE application node and choose Deploy from the contextual menu.

You can read the progress of deployment in the IDE’s output window, on the RI Application tab. The text there should show that a web client named DiningGuideManager_TestApp was deployed.

To verify that the application is deployed, open the J2EE command window. The last statement should be “Application DiningGuideManager_TestApp deployed.”

Now, test the DiningGuideManager session bean.

Using the Test Client to Test a Session Bean

On the test client’s web page, create an instance of the DiningGuideManager session bean by exercising the create method; then test the business methods (getRating) on that instance.

Point your web browser to the test application to start the application.

1. From your operating system, open a web browser and point it to the following URL:

   http://localhost:8000/DiningGuideManager_TestApp

   Your browser displays the test client, with DiningGuideManagerHome listed as the only instance in the instance list (upper left).

2. Create an instance of the DiningGuideManager session bean by invoking the DiningGuideManagerHome’s create method.

   The Data.DiningGuideManager[x] instance appears in the instance list. Now you can test the bean’s getter methods.


   The getAllRestaurants and getCustomerreviewsByRestaurant methods are made available.
4. **Click the Invoke button on the getAllRestaurants method.**

   If you created Joe’s House of Fish in the database (in “Testing an Entity Bean With the Test Client” on page 66), a vector of size 3 appears in the list of created objects (upper right), and the results of the method invocation should look as shown. If you didn’t create this record, your results might be different.

<table>
<thead>
<tr>
<th>Results of the Last Method Invocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data.RestaurantDetail@4da86b, Data.RestaurantDetail@6c14c0, Data.RestaurantDetail@468059</td>
</tr>
</tbody>
</table>

   Method Invoked: getAllRestaurants ()
   Parameters: none

5. **Type in Bay Fox in the field for the getCustomerReviewsByRestaurant method and click the Invoke button.**

   No CustomerReviewDetail records should be returned, because there are no customer review comments in the database (see TABLE 1-4 in Chapter 1). Now try the French Lemon record.

6. **Type in French Lemon in the same field and invoke the method.**

   Two CustomerReviewDetail records should be returned:

<table>
<thead>
<tr>
<th>Results of the Last Method Invocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data.CustomerReviewDetail@61469c, Data.CustomerReviewDetail@62bda7</td>
</tr>
</tbody>
</table>

   Method Invoked: getCustomerReviewByRestaurant (java.lang.String)
   Parameters:
   French Lemon

7. **When you are done testing, stop the test client by pointing your web browser at another URL or by exiting the browser.**

---

**Comments on Creating a Client**

Congratulations, you have successfully completed the EJB tier of the EatersDigest application. You are ready to go on to Chapter 4, to use the Forte for Java IDE’s Web Services module to create web services for the application, and then on to Chapter 5 to use the Forte ESP Toolkit to create the pages for your client.

You may, however, wish to create your own web services and client, in which case, the Forte for Java test application can offer some guidelines.
Web services that access a session bean like the DiningGuideManager bean must include a servlet and JSP pages with lookup methods for obtaining the Home interfaces and Home objects of the entity beans in the EJB tier. The web module created by the test application facility offers examples of the required code.

Lookup method examples are found in the EjbInvoker class under the web module. Specifically, look for this class under the WEB-INF/Classes/com/sun/forte4j/j2ee/ejbtest/webtest directory.

For example, the following methods offer good example lookup code:

- EjbInvoker.getHomeObject
- EjbInvoker.getHomeInterface
- EjbInvoker.resolveEjb
Creating the Tutorial Application’s Web Service

This chapter teaches you how to use the Forte for Java Web Services module to create web services for the EatersDigest application.

This chapter covers the following topics:
- “Overview of the Tutorial’s Web Service,” which follows
- “Creating XML Operations” on page 84
- “Creating a Web Service” on page 91
- “Creating a J2EE Application for the Web Service” on page 92
- “Deploying the Tutorial Application” on page 93
- “Testing the Generated Client Pages” on page 94

Overview of the Tutorial’s Web Service

In this chapter, you will create the EatersDigest application’s EJB tier as a web service. As part of this procedure you will explicitly create a number of components and generate some others.

You’ll explicitly create one logical web service and four XML operations:
- the DiningGuideWebService web service
- the AllRestaurantsXMO XML operation
- the AllCustomerReviewsByRestaurantNameXMO XML operation
- the EnterCustomerReviewXMO XML operation
- the AddCustomerReviewXMO XML operation
You’ll generate runtime classes—one class for each XML operation and one class for the web service:

- DiningGuideWebService.class
- AllRestaurantsXMO.class
- AllCustomerreviewsByRestaurantNameXMO.class
- EnterCustomerReviewXMO.class
- AddCustomerReviewXMO.class

You’ll generate client pages for testing the web service—four JSP pages (one for each XML operation) and one HTML welcome page:

- AllRestaurantsXMO.jsp
- AllCustomerreviewsByRestaurantNameXMO.jsp
- EnterCustomerReviewXMO.jsp
- AddCustomerReviewXMO.jsp
- DiningGuideWebService.html

You’ll also explicitly create a J2EE application, to which you will add your web service and the EJB module you created in the previous chapter.

The Web Service

For more complete information about web services and how to create and program them, see "Building Web Services."

The web service you create in the IDE is a logical entity that represents and allows you to program your web service. You develop your web service’s functionality by creating XML operations and adding them to the logical web service. You also add any document files the web service needs to reference, such as JSP pages, HTML files, image files, and so forth.

The XML Operations

For more complete information about XML operations and how to create and program them, see "Building Web Components."

XML operations are logical entities that represent and allow you to program your web service’s response mechanism. Each XML operation you create represents the response to a specific client request. When the web service receives a client request, it executes a single XML operation in response.

You program each XML operation to call some number of methods on objects in the EJB tier. In the tutorial web service, the XML operations call methods on the application’s entity beans.
XML operations accept client requests in the form of an XML document (the XML input document). When an XML operation receives a request, it maps elements of the XML input document to parameters of the methods you have programmed the XML operation to call. The XML operation then calls the methods and formats their return values as an XML document and returns the document (the XML output document).

The Source Editor enables you to program your XML operations without hand coding the XML documents or the method calls.

The Runtime Classes

When you have finished programming your web service, you generate its runtime classes. You don’t work directly on the runtime classes, but you will see them generated in the package containing the XML operations and the logical web service.

The Client Pages

When you generate your web service's runtime classes, client pages are also generated in the logical web service's Documents directory. You'll use these client pages for testing the web service. You can also use them as a starting point or a guide for developing a full-featured client.

One JSP page is generated corresponding to each XML operation. One welcome HTML page is also generated for the entire web service.

The welcome page contains one HTML form corresponding to each of the generated JSP pages. During testing, you use these HTML forms to request their corresponding JSP pages. The HTML forms contain input fields for you to provide parameter values for JSP pages that require parameters. Chapter 5 offers guidelines in this direction.

When you request one of these JSP pages from the welcome page:

1. The JSP page sends a corresponding request to the web service. It sends its request in the form of an XML document, which it creates using the ESP tag library provided with the Forte for Java distribution.
2. The web service executes the requested XML operation.
3. The XML operation processes the request and returns data in an XML output document.
4. The ESP tag library extracts the data from the XML output document, and the JSP page formats the data for display in the web browser.
Creating XML Operations

Before you create the web service, you must first create the XML operations. You will create operations that are based on methods of the application's two entity beans:

- **AllRestaurantsXMO**—based on the `Data.RestaurantHome.findAll` method
  Also calls `Restaurant.getRestaurantDetail`. Returns a collection of `RestaurantDetail` objects, one object for every restaurant in the database.

- **AllCustomerreviewsByRestaurantNameXMO**—based on the `Data.CustomerreviewHome.findByRestaurantName` method
  All calls `Customerreview.getCustomerreviewDetail`. Takes a `restaurantname` parameter and returns a collection of `CustomerreviewDetail` objects for the named restaurant.

- **AddCustomerReviewXMO**—based on the `Data.CustomerreviewHome.create` method
  Takes an input `restaurantname`, `customername`, and `review` data and returns a `CustomerreviewDetail` object.

You must also create a fourth XML operation, which is not based on a method in the `Data` package, but on a Web Services design utility method. This method will enable the application to pass in and return the same parameter. The application will use this operation to pass a restaurant name first to the Customer Review data entry page, and then, along with three other parameters, to the JSP page that adds the Customer Review to the database. The XML operation for this is:

- **EnterCustomerReviewXMO**—based on the `com.sun.forte4j.webdesigner.utilities.WebServicesDesignerUtilities.Identity` method
  Takes a string named `element1`, which it uses for the `restaurantname` field.

Creating XML Operations Based on Entity Bean Methods

Create a new package named `WebService` under the `DiningGuide` directory. Then use the Web Services wizard to create three XML operations within it.
Creating the WebService Package

1. In the Forte for Java Explorer, right-click the mounted DiningGuide Filesystem and choose New Package.
   The New Package dialog box is displayed.

2. Type WebService for the name and click OK.
   The new WebService package appears under the DiningGuide directory.

Creating the AllRestaurantsXMO Operation

1. Right-click the new WebService package and choose New > Web Services > XML Operation.
   The XML Operation pane of the New From Template wizard appears.

2. Enter the following values:
   - Name: AllRestaurantsXMO
   - Package: WebService
     This should be the default, but if it is not specified, use the Browse button to find this package.
   - Method: Data.RestaurantHome.findAll
     Use the Browse button to find this method from the logical bean, as shown. If you don't, the XMO editor will not display it correctly.

   - Method returns a collection of this class: Data.Restaurant
     This is the remote interface of the Restaurant entity bean. As with the Method, above, use the Browse button to find the remote interface from the logical bean, as shown.
When you’re done, the New XML Operation wizard should look like this:

3. Click Finish.
   A new XML operation node and its contents appears in the Explorer and the Web Services Source Editor displays the operation.

4. In the Data Source pane of the Web Services Source Editor, expand all the nodes to display the contents of the new operation.
   The Web Services Source Editor should look like this:
5. Delete the getRating method (right-click it and choose Delete).

6. Right-click the Returns: Data.RestaurantDetail node (under getRestaurantDetail) and choose Expand.

This action adds all the RestaurantDetail fields to the operation. The fields are displayed in the XML Output Format pane of the Source Editor. Notice that the labels for the fields have been converted from all lowercase (in the bean) to initial capitals. This is important to remember when referring to them later in code.
Creating the AllCustomerreviewsByRestaurantNameXMO Operation

1. Using Step 1 through Step 3 from the previous procedure, create an XML operation with the following values:
   - Name: AllCustomerreviewsByRestaurantNameXMO
   - Package: WebService
   - Method: Data.CustomerreviewHome.findByRestaurantName
   - Method returns a collection of this class: Data.Customerreview

2. Open the AllCustomerreviewsByRestaurantNameXMO operation in the Source Editor.
   - It may open automatically, but if not, open it by double-clicking the operation in the Explorer.

3. On the Data Source pane of the Source Editor, expand all the nodes under Methods.

4. Delete the getReview method.

5. Add the Customerreview fields to the operation by right-clicking Returns: Data.CustomerreviewDetail and choosing Expand.

Creating the AddCustomerReviewXMO Operation

This operation doesn’t return a collection, so create it as follows:

1. Similar to the other procedures, create an XML operation with the following values:
   - Name: AddCustomerReviewXMO
   - Package: WebService
   - Method: Data.CustomerreviewHome.create
   - Method returns a collection of this class: (Leave blank)

2. Open the AddCustomerReviewXMO operation in the Source Editor.

3. On the Data Source pane of the Source Editor, expand these nodes under Methods:

4. Delete the getReview method.

5. Do not expand the Returns: Data.CustomerreviewDetail node.
   - You do not need the fields for this XML operation.
   - Now, create the last XML operation, which is not based on a method you created.
Creating XML Operations Based on Other Methods

The tutorial uses a method from a Web Services library for passing and returning the same parameter. First mount the library JAR file that contains the bean whose method you need. Then create the XML operation as before.

Mounting the Web Services Module JAR File

Mount the webservices-util.jar file from the Web Services module.

1. **Right-click** Filesystems in the Forte for Java Explorer and choose Mount JAR. The Mount JAR File dialog box appears.
2. **Browse to** FORTE4J_HOME/modules.
3. **Scroll down until you find** webservices-util.jar and select it.
4. **Click Mount.** The webservices-util.jar file is mounted in the Explorer.

Creating the EnterCustomerReviewXMO Operation

Create an XML operation based on the Identity method.

1. **Right-click** the WebService package and choose New > Web Services > XML Operation.
2. **In the wizard, enter the following values:**
   - **Name:** EnterCustomerReviewXMO
   - **Package:** WebService
   - **Method:** com.sun.forte4j.webdesigner.utilities.WebServicesDesignerUtilities.Identity
   - **Method returns a collection of this class:** (leave blank)
When you select the **Identity** method the Select Method window should look like this:

3. Click Finish.
   
The new EnterCustomerReviewXMO operation appears in the Explorer.

4. **Open the EnterCustomerReviewXMO operation in the Source Editor.**
   
   It may open automatically, but if not, double-click the operation in the Explorer.

5. On the Data Source pane of the Source Editor, expand the **Methods** node.

6. **Open the Identity method’s properties.**
   
   If the Properties Window is not already visible, right-click the method and choose Properties.

7. **Change the Tag Name value to restaurantname.**
   
   Notice that the field’s name has changed in the XML Output Format pane of the Source Editor.
   
   You are now finished creating the XML operations. Next, create the web service.
Creating a Web Service

Create the DiningGuideWebService webservice and add references to it to the XML operations you just created.

1. In the IDE’s Explorer, right-click the WebService package and choose New > Web Services > Web Service.
   The New Web Service wizard window appears.

2. Enter the following values:
   Name: DiningGuideWebService
   Package: WebService

3. Click Finish.
   The new DiningGuideWebService web service appears under the WebService package in the Explorer.

4. Right-click the DiningGuideWebService node and choose Add Reference.
   A browse window appears displaying the mounted directories. Open the DiningGuide/WebService node.

5. Select the four XML operations you just created.
   Use the Ctrl key to select multiple items.

6. Click OK.

7. Right-click the DiningGuideWebService node and choose Generate/Compile Java File.
   When the operation is finished, the word “Finished” appears in the IDE’s output window and many new items have been created under the WebService node, including a Documents folder.
This concludes creating the web service, and you are ready to create a J2EE application for it.

Creating a J2EE Application for the Web Service

Now create a J2EE application that contains the EJB modules and the web service module. Name the application EatersDigestApp and add the DiningGuideWebService module, the Restaurant_EJBModule, and the Customerreview_EJBModule from the Data package to it. Then specify a web context for the application.

1. In the IDE’s Explorer, right-click the WebService package and choose New > J2EE > Application.
   The New J2EE Application wizard appears.

2. Type EatersDigestApp for the application’s name and click Finish.
   The new J2EE application appears under the WebService directory.

3. Right-click the EatersDigestApp application and choose Add Module.
   The Add Module to Application window appears.
4. **Find and select the** DiningGuideWebService **node and click OK.**

5. **Open the EatersDigestApp application.**

   Notice that both the DiningGuideWebService’s WAR and EJB JAR files have been added to the application.

6. **Right-click the EatersDigestApp application again and choose Add Module.**

7. **In the browse window, open the Data package and select the following modules and click OK:**

   Restaurant_EJBModule
   Customerreview_EJBModule

   (Use Ctrl-click to select multiple items.)

   **Note** – If both of these modules are not in the Data package, you must create them and then add them. Refer to “Creating a Test Client for an Entity Bean” on page 62.

Now, give the new J2EE application a web context.

8. **Open the property sheet of the** DiningGuideWebService_War **file inside the EatersDigestApp application.**

   If the Properties window isn’t still open, right-click the file and choose Properties.

9. **Type** EatersDigestContext **in the Web Context field.**

10. **Choose File > Save All.**

    You are finished creating the EatersDigestApp application and are now ready to deploy it.

---

**Deploying the Tutorial Application**

Before you deploy the tutorial application, make sure the IDE is set up to run the J2EE Reference Implementation. For information, refer to “Setting up the IDE to Run the Reference Implementation” on page 19.

Deploying the application will take longer than when you deployed with the test application facility (see “Testing the Session Bean” on page 77). Have patience.
In the IDE’s Explorer, right-click the EatersDigestApp application and choose Deploy.

A progress monitor appears, showing the progress of the operation.

You can read the progress of deployment in the IDE’s output window, on the RI Application tab. At the end, the text there should show that deployment of EatersDigestApp has completed.

To verify that the application is deployed, open the J2EE command window. The last statement should be “Application EatersDigestApp deployed.”

---

Testing the Generated Client Pages

1. From your operating system, open your web browser and point it to the following URL:

   http://localhost:8000/EatersDigestContext

   Your browser displays the generated welcome page. (In the Explorer, this is the DiningGuideWebService HTML page in the Documents folder.) As described in “The Client Pages” on page 83, the Web Services module generates one JSP page for each XML operation and one welcome page that contains the HTML form that allows you to test each of the generated JSP pages. The welcome page for the EatersDigest tutorial is shown as follows.
This page allows you to test whether each of the operations works as expected.

2. Test the AllCustomerreviewsByRestaurantNameXMO by typing French Lemon in the text field and clicking the Submit Query button.

The application should return the data from the database and display it in the generated JSP page for this operation, as shown.
3. Use the Back button on your browser to return to the welcome page.

4. Test the AddCustomerReviewXMO operation by typing French Lemon in the restaurantname field, and whatever you want in the other two fields.

   For example:

   ![AddCustomerReviewXMO form]

5. Click the Submit Query button.

   The application displays the page generated for this operation with only the operation label on it. (You will improve on this in Chapter 5, in “Adding a Customer Review” on page 108). To see if the record was written to the database, go on to the next step.
6. Return to the welcome page and repeat Step 2 (get all the customer reviews for the French Lemon restaurant).

   The application redisplays the new data added to the old data.

7. Use the Back button on your browser to return to the welcome page.

8. Test the AllRestaurantsXMO operation by clicking its Submit Query button on the welcome page.
9. Go back to the welcome page and type anything in the EnterCustomerReviewXMO operation's page.

For example:

```
EnterCustomerReviewXMO

Input Name: Name
Submit Query
```
10. Click Submit Query.

The resulting page should show the same text you entered, for example:

This seemingly uninteresting page will be useful later when you need to pass parameters to and from a JSP page.

Congratulations, you have successfully created a web service for the EatersDigest tutorial. In Chapter 5, you will use some provided HTML pages that use the JSP pages generated for the web service to create the finished client.
Creating a Client for the Tutorial Application

This chapter shows you how to run the EatersDigest application using JSP pages that are derived from the JSP pages you generated in Chapter 4, but modified for aesthetics and ease of use. The modified JSP pages are provided in the "/sampledir/tutorial/DiningGuide/CustomizedPages" directory.

This chapter covers these topics:

■ “Developing the Tutorial’s Client,” which follows
■ “Adding the Modified JSP Pages to the Tutorial Application” on page 103
■ “Running the Tutorial Application” on page 104
■ “Understanding the Modifications” on page 105

Developing the Tutorial’s Client

In a normal development process, application developers are developing the “back end” of the application (the EJB tier and the web services) while web designers are somewhat independently developing the application’s “front end” (the client’s web pages), using placeholders for method calls and references to dynamic data. At some point, the web designers must add the real calls and references to the client pages.

Joining Development and Web Design With Forte for Java Tools

The Forte for Java mechanisms for facilitating joining the application’s back end to its front are the Web Services module and the Forte for Java Enterprise Service Presentation Toolkit (the Forte ESP Toolkit).
Developers use Web Services to create XML operations that call one or more methods from the application services tier. Using the Web Services GUI, developers can generate, for each operation, a test client JSP page that contains a Forte ESP tag to call the operation. You have seen how this is done in Chapter 4.

When Forte ESP Toolkit is installed on the application’s web server, developers can publish those XML operations to the Forte ESP registry from the Forte for Java IDE.

Once the application’s XML operations are published to the Forte ESP registry, web designers, using design tools that have Forte ESP extensions installed, can graphically map the dynamic data of the operations to elements in their page layouts. At runtime, the extensions insert custom tags in the JSP pages to access the data dynamically.

Forte ESP Toolkit extensions are available for Macromedia® Dreamweaver™ and Adobe® GoLive™. You can download Forte ESP Toolkit and free trial versions of Dreamweaver and GoLive from the Forte for Java portal.

**Skipping the Forte ESP Toolkit Step for EatersDigest**

Forte ESP Toolkit has its own tutorial, also using a restaurant theme. The setup files for the Forte ESP tutorial are part of the Forte ESP Toolkit software, and the instructions for inserting the dynamic data—including detailed examples of how the Forte ESP tags and XML and XSLT coding is automatically generated—are found in the accompanying Forte ESP user guide, *Building JSP Pages That Use XML Data Services*. For a short tutorial that steps you through the process of publishing XML operations and developing a Dreamweaver client that uses them, see the Hello Suite example within the Enterprise Edition-specific examples you can download from the Forte for Java portal.

For the EatersDigest tutorial, imagine that web designers have been using Dreamweaver or GoLive applications to develop the client web pages, using techniques similar to the ones described in the Forte ESP tutorial. Imagine also that you had installed Forte ESP Toolkit, and had published the four XML operations you created in Chapter 4 into the Forte ESP registry, so that the web designers could use them in their web pages. (Instructions for publishing in the ESP registry are given in *Building Web Services*.)

All you will need to do to finish the EatersDigest tutorial application is install the finished pages in the WebService/Documents directory, redeploy the EatersDigest application, and point your web browser to the application’s first page.
Adding the Modified JSP Pages to the Tutorial Application

The four modified client pages are found in the CustomizedPages subdirectory of the ~/sampledir/tutorial/DiningGuide directory under your user settings directory. In the Forte for Java IDE, mount the sampledir directory, copy the files to the web service’s Documents directory, and then redeploy the application.

1. In the Forte for Java Explorer, mount the your-user-dir/sampledir directory.
2. Expand the ~/sampledir/tutorial/DiningGuide/CustomizedPages directory.
   You’ll see four JSP pages.
3. Select the four JSP pages and choose Edit > Copy.
4. Right-click the Documents node under DiningGuide/WebService and choose Paste > Copy.
5. Right-click the EatersDigestApp node and choose Deploy.
   When the deploy process finishes successfully, go to “Running the Tutorial Application” on page 104.
Running the Tutorial Application

Run the EatersDigest application by pointing your browser to the deployed application.

1. Make sure that J2EE is running.
   See “Starting Up the Java 2 SDK, Enterprise Edition Software” on page 14, for information.

2. Point your browser to the DGAllRestaurants.jsp in the EatersDigestContext directory.
   For example, type http://localhost:8000/EatersDigestContext/DGAllRestaurants.jsp.
   The Restaurant Listing page should appear.

3. Play around with the features, as described in “User’s View of the Tutorial Application” on page 25.

4. Quit the application either by exiting the browser or pointing it at a different URL.
Understanding the Modifications

By comparing the customized JSP pages with those generated by Web Services, you can see the techniques that were used to create presentation and flow while using dynamic data. This section explains how this was done with the EatersDigest application. For a more thorough description of details, see the Forte ESP tutorial, described in Building JSP Pages That Use XML Data Services.

All of the customized pages are based on the files you generated with Web Services in Chapter 4. In both versions of the files, the application’s methods-turned-XML operations are referenced within a code block of Forte ESP tags: `<forte:esp> body </forte:esp>`. Sometimes other Forte ESP tags occur within the block, as well. At runtime, Java code in the Forte ESP tag library is called to process the tag and the body of the tag. These tags are processed by the JSP runtime system, so you don’t need to have the Forte ESP Toolkit installed to process them.

All of the pages have undergone standard HTML changes, for example:

- Titles and table column labels improved
- Table borders eliminated
- The list of restaurants table formatting change

Observe these when comparing the two restaurant listing pages:

![FIGURE 5-1  Generated Restaurant List Page vs. the DGAllRestaurants Page](image)
They are also the main changes between the two View Customer Review pages.

Two customizations are worth examination: passing a parameter within an HTML form and adding a customer review.

**Passing a Parameter Within an HTML Form**

The Restaurant Listing page (DGAllRestaurants) provides a good illustration of how a parameter from an XML result set can be passed when an HTML form is submitted. This is done for both buttons that appear on this page, the View Customer Comments button and the Add Your Comments button. All of this is illustrated in the code segment that follows.

The code segment begins with the Forte ESP tag generated by Web Services for the XML operation that retrieves the collection of restaurant detail objects. This is followed by an XSL statement to generate a display for each restaurant object returned. Within this display, forms are defined for each of the two buttons.

The first form defines the View Customer Comments button. The form calls the DGA1CustomerreviewsByRestaurantName page, which is based on the AllCustomerreviewsByRestaurantNameXMO. The first highlighted (bolded) area in the code segment shows how the Forte ESP `<forte:inputtext>` tag is used to pass the restaurantname parameter from the XML result set of the AllRestaurantsXMO in order to retrieve the correct list of customer comments.
The second form defines the Add Your Comments button. This form calls the DGEnterCustomerReview page, based on the Identity method (see “Creating XML Operations Based on Other Methods” on page 89) that passes the element1 parameter. Remember that this XML operation was generated to enable the application to pass in and return the same parameter.

The highlighted (bolded) area within this form shows how the <forte:inputtext> tag is used to pass the element1 parameter.

```xml
<forte:esp name="TestAllRestaurantsXMO" sourceURL=
"local::AllRestaurantsXMO:"resultSet="AllRestaurantsXMO/Collection-
Restaurant/Restaurant/RestaurantDetail:Address, Cuisine, Description,
Neighborhood, Phone, Rating, Restaurantname">
<table width="99%" border="0">
  <xsl:for-each select="AllRestaurantsXMO/Collection-
Restaurant/Restaurant/RestaurantDetail">
    <tr>
      <td><b>Restaurantname</b></td>
      <td><forte:expand value="{Restaurantname}"/>Restaurantname</td>
    </tr>
  </xsl:for-each>
</forte:esp>

<form method="get" action="DGAllCustomerReviewsByRestaurantName.jsp">
  <forte:inputtext value="{Restaurantname}"/>
  <input type="hidden" name="restaurantname" value="restaurantHolder" />
  <input type="submit" name="ViewCommentsButton" value="View Customer Comments"/>
</form>

<form method="get" action="DGEnterCustomerReview.jsp">
  <forte:inputtext value="{Restaurantname}"/>
  <input type="hidden" name="element1" value="restaurantHolder"/>
  <input type="submit" name="AddCommentsButton" value="Add Your Comments"/>
</form>
```
Adding a Customer Review

To add a customer review, the application uses the DGEnterCustomerReview page, based on the EnterCustomerReviewXMO page, to gather the data. DGEnterCustomerReview calls DGAddCustomerreview, which uses the AddCustomerReviewXMO operation to perform the actual insert to the database.

Consider the input form that the generated welcome page displayed for the AddCustomerReviewXMO operation:

FIGURE 5-3  Generated Input Form for AddCustomerReviewXMO

If you then examine the customized page (DGEnterCustomerReview) for entering customer review input, you can see that the AddCustomerReviewXMO input form was used.

FIGURE 5-4  DGEnterCustomerReview Page

You can also see that the EnterCustomerReviewXMO operation has allowed the selected restaurant’s restaurantname parameter to be filled into the Restaurant Name field.
When you examine the code for the DGEnterCustomerReview page, you’ll see how this was done.

```html
<forte:esp name="EnterCustomerReviewXMO" inline="true" sourceURL="local::EnterCustomerReviewXMO:?element1={?element1}" resultSets="EnterCustomerReviewXMO:restaurantname">
<form method="get" action="DGAddCustomerReview.jsp">
  <TABLE>
    <TR>
      <TD>
        <TABLE>
          <TR>
            <TH>Restaurant Name</TH>
            <TD>
              <forte:inputtext value="EnterCustomerReviewXMO/restaurantname">
                <input type="text" name="restaurantname" value="restaurantHolder" />
              </forte:inputtext>
            </TD>
          </TR>
          <TR>
            <TH>Customer Name</TH>
            <TD>
              <input type="text" size="20" name="customername" />
            </TD>
          </TR>
          <TR>
            <TH>Review</TH>
            <TD>
              <input type="text" size="20" name="review" />
            </TD>
          </TR>
        </TABLE>
      </TD>
      <TD>
        <input type="submit" value="Submit" />
      </TD>
    </TR>
  </TABLE>
</form>
</forte:esp>
```
Lastly, the DGAddCustomerReview page inserts the data into the database by calling AddCustomerReviewXMO and then displaying all the restaurants.

```xml
<forte:esp name="AddCustomerReviewXMO" inline="true" sourceURL=
"local::AddCustomerReviewXMO:?restaurantname=
{?restaurantname}?customername={?customername}?review=
{?review}" resultSet="AddCustomerReviewXMO:Customerreview">
  <table width="99%" border="1">
    <tr>
      <td><b>Customerreview</b></td>
    </tr>
    <xsl:for-each select="AddCustomerReviewXMO">
      <tr>
        <td><forte:expand value="" 
          value="" 
          value="">Customerreview</forte:expand>
        </td>
      </tr>
    </xsl:for-each>
  </table>
</forte:esp>
```
Index

A
AddCustomerReviewXMO
   creating, 88
   in customized pages, 108
   testing, 96
All Customer Reviews By Restaurant Name page, 26
AllCustomerReviewsByRestaurantNameXMO
   creating, 88
   in customized pages, 106
   testing, 95
AllRestaurantsXMO
   creating, 85
   in customized pages, 105, 106
   testing, 97

B
beans directory, description, 20
bin directory, description, 20
business methods
   Customerreview
      getCustomerreviewDetail, 61
      getReview, 52
   DiningGuideManager
      getAllRestaurants, 73, 79
      getCustomerreviewsByRestaurant, 74, 79
   Restaurant
      getRating, 51, 52, 69
      getRestaurantDetail, 61
      Restaurant.getRestaurantDetail, 40

C
commands
   Add Module, 92
   Add Reference, 91
   Deploy, 94, 103
   Generate Java, 55
   Generate/Compile Java File, 91
   Mount Filesystem, 42
   New Business Method, 51
   New Constructor, 59
   New Create Method, 48
   New Database Schema, 54
   New EJB Entity Bean, 43
   New Finder Method, 50
   New J2EE Application, 92
   New Web Service, 91
   New XML Operation, 85
   Paste Copy, 103
constructors
   CustomerreviewDetail, 60
   CustomerreviewKey, 47
   RestaurantDetail, 59
create methods
   Customerreview.create, 49, 67
   DiningGuideManager.create, 71 to 72, 78
   JNDI lookup code in, 71
   Restaurant.create, 48, 67
Customerreview entity bean
   create method, creating, 49
   creating, 42 to 53
   getReview method, creating, 52
getReview method, deleting from XML operation, 88
testing, 69
CustomerReview table, description, 11
CustomerReview_TestApp, creating, 69
customized pages
adding a customer review, 108
adding to the tutorial application, 103
DGAddCustomerReview, 110
DGAllCustomerreviewsByRestaurantName, 106
DGAllRestaurants, 104, 105
DGEnterCustomerReview, 108
passing a parameter in an HTML form, 106

database schemas
capturing, 54
connecting to entity beans, 43
database tables
creating, 11 to 14
test form for Reference Implementation, 11
detail classes
description, 33, 39
ingoing, 58 to 60
creating, 54 to 58
DiningGuideManager session bean
create method, coding, 71 to 72
creating, 70 to 76
getAllRestaurants method, 73 to 74, 79
getCustomerreviewsByRestaurant method, 74 to 75, 79
testing, 78 to 79
DiningGuideManager_TestApp
bean methods, testing, 78 to 79
creating, 77 to 78
DiningGuideWebService webservice,
creating, 91
docs directory, description, 20

e
EatersDigest tutorial
and the Forte ESP tutorial, 102
application scenarios, 24

Index 112  Forte for Java, Enterprise Edition Tutorial  •  August 2001
connecting to a database schema, 43
create methods, creating, 48
create methods, testing, 66
creating, 42 to 52
finder methods, creating, 50
finder methods, testing, 68
primary key class, 46
primary key class's constructors, 47
testing, 62 to 69
example applications, location, 6, 21

F
ffjuser30, UNIX default user settings file, 18
finder methods
Customerreview.findByRestaurantName, 50, 68
Restaurant.findAll, 40, 50, 68
testing, 68
Forte ESP Toolkit
inputtext tag, 106 to 110
the ESP tutorial, 102
using ESP tags, 105
using with Web Services, 101 to 102
Forte for Java IDE
command-line switches, 17
descriptions of subdirectories, 20
requirements, 10
starting on Solaris, Linux, and other UNIX software, 16
starting on Windows, 16

I
ide.log file, location, 21
Identity method, 90
inputtext tag, using, 106 to 110
iPlanet directory, description, 20

J
J2EE applications
creating, 92
deploying, 93
EatersDigestApp, 92
J2EE Reference Implementation (RI)
configuring for a database, 13
properties on entity beans, 62
required version, 10
setting up the IDE to run, 19
Java 2 SDK, Enterprise Edition (J2EE)
required version, 10
starting, 14
Javadoc
directory, 20
using in the IDE, 6
JNDI lookup code, 71

L
lib directory, description, 20

M
modules directory, description, 20

N
Netscape, supported version, 10

P
pages
All Customer Reviews By Restaurant Name, 26
Enter Customer Review, 27
Restaurant Listing, 25
parameters
adjusting type for rating, 58
changing order of, 67
order in test client, 67
passing within an HTML form, 106
PointBase
and the Reference Implementation, 13
home directory, 20
installing a database table, 12
supported version, 10
Reference Implementation. See J2EE Reference Implementation

rest_pb.sql database script, 11

Restaurant entity bean
  create method, 48, 67
  creating, 42 to 53
  findAll method, 40
  getRating method, 51, 69
  getRating method, deleting from XML operation, 87
  getRestaurantDetail method, 40
  testing, 66 to 69

Restaurant Listing page, 25

Restaurant table, description, 11

Restaurant_TestApp
  bean methods, testing, 66 to 69
  creating, 62 to 66

runide.sh. See Forte for Java IDE, starting on Solaris

sampledir directory, description, 21

session beans
  business methods, creating, 73 to 75
  create method, modifying, 71
  create method, testing, 78
  creating, 70 to 76
  EJB references, adding, 75 to 76
  testing, 77 to 79

sources directory, description, 20

system directory, description, 20

teamware directory, description, 20

test application facility
  adding entity beans to the EJB module, 77
  entity beans, testing, 66 to 69
  session bean, testing, 77 to 79
  test client, creating, 62 to 66, 77 to 78
  test client, using, 66 to 69, 78 to 79
  using in EatersDigest, 33
  test applications

Customerreview_TestApp, 69
DiningGuideManager_TestApp, 77
Restaurant_TestApp, 63

testing enterprise beans
  business methods, testing, 69
  create method, testing, 67
  finder methods, testing, 68
  results in IDE's output window, 66, 78, 94
  results in J2EE command window, 66, 78, 94
  test page, 66, 78

user settings directory
  specifying at initial launch, 18
  specifying with a command-line switch, 18
  UNIX default, 18

web browsers, supported versions, 10

web servers, supported versions, 10

Web Services module
  about generated client pages, 83
  about generated runtime classes, 83
  about the generated web service, 82
  about XML operations, 82
  adding references to XML operations, 91
  creating, 91

Documents directory, 35, 102
  using with Forte ESP Toolkit, 101 to 102

webservices-util.jar file, 89

work directory. See user settings directory

XML operations
  AddCustomerReviewXMO, 88, 108, 110
  adding to a web service, 91
  AllCustomerReviewsByRestaurantNameXM, 88, 106
  AllRestaurantsXMO, 85, 105, 106
  creating, 84 to 90
  description, 82
  EnterCustomerReviewXMO, 89, 108