



Sun™ ONE Studio 5 Web Application Tutorial

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

Part No. 817-2320-10
June 2003 Revision A

Send comments about this document to: docfeedback@sun.com

Copyright © 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Sun, Sun Microsystems, the Sun logo, Forte, Java, NetBeans, iPlanet, docs.sun.com, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

Federal Acquisitions: Commercial Software - Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à <http://www.sun.com/patents> et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats - Unis et dans les autres pays.

Ce produit est un document protégé par un copyright et distribué avec des licences qui est en restreignent l'utilisation, la copie, la distribution et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Sun, Sun Microsystems, le logo Sun, Forte, Java, NetBeans, iPlanet, docs.sun.com, et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

UNIX est une marque enregistrée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company Ltd.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISÉE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.



Contents

Before You Begin 11

1. Getting Started 19

Obtaining and Installing the Required Software 19

Starting the Software 21

Starting the IDE 21

Starting the Application Server 21

Confirming Sun ONE Application Server 7 as the Default Server 27

Creating the Tutorial Database Table 27

2. Introduction to CDShopCart 31

Functionality of the Tutorial Application 31

Application Scenarios 32

Application Functional Specification 33

User's View of the Tutorial Application 33

Architecture of the Tutorial Application 37

Application Elements 38

Service Component Details 39

Overview of Tasks for Creating the Tutorial Application 40

Creating a Web Module 40

Using the JSTL Tag Libraries	41
Creating the Supporting Elements	41
Testing the Application	42
End Comments	42
3. Creating the CDShopCart Application	43
Creating a Web Module	43
What Is a Sun ONE Studio 5 Web Module?	44
Creating the CDShopCart Web Module	44
Setting the Web Application's Context Root	46
Using JSP Tags to Fetch and Display Database Data	47
What Is a JSP Tag?	47
Using JSTL Tags	48
Creating the CD Catalog List Page	51
Creating the Shopping Cart Page and Supporting Elements	58
Creating the CartLineItem Bean	59
Creating the Cart Bean	65
Creating the Shopping Cart Page	68
Testing the Shopping Cart Page	72
Creating the Three Message Pages	73
Creating the Empty Cart Page	74
Creating the Place Order Page	75
Creating the Cancel Order Page	77
Testing the Three Message Pages	79
A. CDShopCart Source Files	81
ProductList.jsp Source	82
CartLineItem Bean Source	84
CartLineItemBeanInfo Source	87

Cart Bean Source	91
ShopCart.jsp Source	94
EmptyCart.jsp Source	96
PlaceOrder.jsp Source	96
CancelOrder.jsp Source	97
B. CDSShopCart Database Scripts	99
Script for a PointBase Database	100
Script for an Oracle Database	101
Script for a Microsoft SQLServer Database	102
Script for an IBM DB2 Database	103
C. Creating the Tutorial with an Oracle Database	105
Connecting the IDE to an Oracle Database	106
Enabling the Oracle Type 4 JDBC Driver	106
Connecting the IDE to the Oracle Server	107
Creating the Database Tables	108
Viewing the Database Tables in the IDE	109
Creating the CDSShopCart Application with an Oracle Database	110
Index	111

Figures

FIGURE 2-1	Architecture of the CDShopCart Application	38
FIGURE 3-1	CD Catalog List Page	51
FIGURE 3-2	Shopping Cart Page	58
FIGURE 3-3	Empty Cart Page	74
FIGURE 3-4	Place Order Page	76
FIGURE 3-5	Cancel Order Page	78

Tables

TABLE 1-1	Admin Server Property Values	23
TABLE 1-2	CDCatalog Database Table	30
TABLE 1-3	CD Table Records	30

Before You Begin

Welcome to the Sun™ ONE Studio 5 Web Application tutorial. In this tutorial, you will learn how to use the features introduced in Sun ONE Studio 5, Standard Edition, namely:

- Support for web applications that use Java™ Servlet and JavaServer Pages™ (JSP™) technology
- Database access using the JSP Standard Tag Library (JSTL) reference implementation from the Jakarta Project
- Deploying and execution to the Sun ONE Application Server—for testing the tutorial application

See the release notes for a list of environments in which you can create the examples in this book. The release notes are available on this web page:

<http://forte.sun.com/ffj/documentation/index.html>

Screen shots vary slightly from one platform to another. Although almost all procedures use the interface of the Sun ONE Studio 5 software, occasionally you might be instructed to enter a command at the command line. Here too, there are slight differences from one platform to another. For example, a Microsoft Windows command might look like this:

```
c:>cd MyWorkDir\MyPackage
```

A UNIX command might look like this:

```
% cd MyWorkDir/MyPackage
```

Before You Read This Book

This tutorial creates a simple web application that interacts with a database and displays dynamically generated content. The design and architecture conforms to the Java 2 Platform, Enterprise Edition (J2EE™) Blueprints. If you want to learn how to use the features of Sun ONE Studio 5, Standard Edition to build the components of a web application, you will benefit from working through this tutorial.

Before starting this tutorial, you should be familiar with the following subjects:

- Java programming language
- Java Servlet syntax
- JDBC™ enabled driver syntax
- JavaServer Pages syntax
- HTML syntax
- JSP Standard Tag Library (JSTL) from the Jakarta Project
- Relational database concepts (such as tables and keys)
- How to use the chosen database

This book requires a knowledge of Java Servlet and JavaServer Pages concepts, including web applications. The following resources define these concepts:

- *Java Servlet Specification Version 2.3*
<http://java.sun.com/products/servlet/download.html#specs>
- *JavaServer Pages Specification Version 1.2*
<http://java.sun.com/products/jsp/download.html#specs>

For the JSTL tutorial, as well as links to other useful information, see:

<http://jakarta.apache.org/taglibs/tutorial.html>

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document and does not endorse and is not responsible or liable for any content, advertising, products, or other materials on or available from such sites or resources. Sun will not be responsible or liable for any damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services available on or through any such sites or 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 CDShopCart tutorial, explains how to start the Sun ONE Studio 5 integrated development environment (IDE), and how to start the Sun ONE Application Server web server. This chapter also describes how to create the tutorial database table.

Chapter 2 describes the architecture of the CDShopCart application.

Chapter 3 provides step-by-step instructions for creating the CDShopCart application, a simple online shopping cart application for the purchase of music CDs.

Appendix A provides complete source files for the tutorial application.

Appendix B provides database script files for the tutorial application.

Appendix C describes how to adapt the tutorial to create and run the application using an Oracle database.

Typographic Conventions

Typeface	Meaning	Examples
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.cvspass</code> file. Use <code>DIR</code> to list all files. Search is complete.
AaBbCc123	What you type, when contrasted with on-screen computer output	> login Password:
<i>AaBbCc123</i>	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> save your changes.
<code>AaBbCc123</code>	Command-line variable; replace with a real name or value	To delete a file, type <code>DEL filename</code> .

Related Documentation

Sun ONE Studio 5 documentation includes books delivered in Acrobat Reader (PDF) format, release notes, online help, readme files for example applications, and Javadoc™ documentation.

Documentation Available Online

The documents described in this section are available from the `docs.sun.com`SM web site and from the documentation page of the Sun ONE Studio Developer Resources portal at <http://forte.sun.com/ffj/documentation>.

The `docs.sun.com` web site (<http://docs.sun.com>) enables you to read, print, and buy Sun Microsystems manuals through the Internet. If you cannot find a manual, see the documentation index that is installed with the product on your local system or network.

- Release notes (HTML format)

Available for each Sun ONE Studio 5 edition. Describe last-minute release changes and technical notes.

 - *Sun ONE Studio 5, Standard Edition Release Notes* - part no. 817-2337-10
- Getting Started guides (PDF format)

Describe how to install the Sun ONE Studio 5 integrated development environment (IDE) on each supported platform and include other pertinent information, such as system requirements, upgrade instructions, application server information, command-line switches, installed subdirectories, database integration, and information on how to use the Update Center.

 - *Sun ONE Studio 5, Standard Edition Getting Started Guide* - part no. 817-2318-10
 - *Sun ONE Studio 4, Mobile Edition Getting Started Guide* - part no. 817-1145-10
- Sun ONE Studio 5 Programming series (PDF format)

This series provides in-depth information on how to use various Sun ONE Studio 5 features to develop well-formed J2EE applications.

 - *Building Web Components* - part no. 817-2334-10

Describes how to build a web application as a J2EE web module using JSP pages, servlets, tag libraries, and supporting classes and files.
 - *Building J2EE Applications* - part no. 817-2327-10

Describes how to assemble EJB modules and web modules into a J2EE application and how to deploy and run a J2EE application.
 - *Building Enterprise JavaBeans Components* - part no. 817-2330-10

Describes how to build EJB components (session beans, message-driven beans, and entity beans with container-managed persistence or bean-managed persistence) using the Sun ONE Studio 5 EJB Builder wizard and other components of the IDE.
 - *Building Web Services* - part no. 817-2324-10

Describes how to use the Sun ONE Studio 5 IDE to build web services, to make web services available to others through a UDDI registry, and to generate web service clients from a local web service or a UDDI registry.
 - *Using Java DataBase Connectivity* - part no. 817-2332-10

Describes how to use the JDBC productivity enhancement tools of the Sun ONE Studio 5 IDE, including how to use them to create a JDBC application.
- Sun ONE Studio 5 tutorials (PDF format)

These tutorials demonstrate how to use the major features of Sun ONE Studio 5, Standard Edition:

 - *Sun ONE Studio 5 Web Application Tutorial* - part no. 817-2320-10

Provides step-by-step instructions for building a simple J2EE web application.

- *Sun ONE Studio 5 J2EE Application Tutorial* - part no. 817-2322-10
Provides step-by-step instructions for building an application using EJB components and web services technology.
- *Sun ONE Studio 4, Mobile Edition Tutorial* - part no. 816-7873-10
Provides step-by-step instructions for building a simple application for a wireless device, such as a cellular phone or personal digital assistant (PDA). The application you build is compliant with the Java 2 Platform, Micro Edition (J2ME™ platform) and conforms to the Mobile Information Device Profile (MIDP) and Connected, Limited Device Configuration (CLDC).

You can also find the completed tutorial applications at:

<http://forte.sun.com/ffj/documentation/tutorialsandexamples.html>

Online Help

Online help is available in the Sun ONE Studio 5 IDE. You can open help by pressing the help key (F1 in Microsoft Windows and Linux environments, Help key in the Solaris environment), or by choosing Help → Contents. Either action displays a list of help topics and a search facility.

Examples

You can download examples that illustrate a particular Sun ONE Studio 5 feature, as well as completed tutorial applications, from the Sun ONE Studio Developer Resources portal at:

<http://forte.sun.com/ffj/documentation/tutorialsandexamples.html>

The site includes the applications that are used in this document.

Javadoc Documentation

Javadoc documentation is available within the IDE for many Sun ONE Studio 5 modules. Refer to the release notes for instructions on installing this documentation.

Documentation in Accessible Formats

The documentation is provided in accessible formats that are readable by assistive technologies for users with disabilities. You can find accessible versions of documentation as described in the following table.

Type of Documentation	Format and Location of Accessible Version
Books and tutorials	HTML at http://docs.sun.com
Mini-tutorials	HTML at http://forte.sun.com/ffj/tutorialsandexamples.html
Integrated example readmes	HTML in the example subdirectories of <i>sIstudio-install-directory/examples</i>
Release notes	HTML at http://docs.sun.com

Contacting Sun Technical Support

If you have technical questions about this product that are not answered in this document, go to:

<http://www.sun.com/service/contacting>

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. Email your comments to Sun at this address:

docfeedback@sun.com

Please include the part number (817-2320-10) of your document in the subject line of your email.

Getting Started

This chapter explains what you must do before starting the Sun ONE Studio 5 Web Application tutorial. The topics covered in this chapter are:

- “Obtaining and Installing the Required Software,” which follows
- “Starting the Software” on page 21
- “Creating the Tutorial Database Table” on page 27

Note – There are several references in this book to the *CDShopCart application files*. These files include a completed version of the tutorial application, a readme file describing how to run the completed application, and SQL scripts for creating the required database table. These files are compressed into a zip file you can download from the Sun ONE Studio 5 Developer Resources portal at <http://forte.sun.com/ffj/documentation/tutorialsandexamples.html>

Obtaining and Installing the Required Software

The following items are required to create and run the tutorial.

- Sun ONE Studio 5, Standard Edition software, which includes:
 - The Sun ONE Studio 5, Standard Edition integrated development environment (IDE)
 - The Sun ONE Application Server 7 software

The Sun ONE Studio 5, Standard Edition installer installs both products unless it detects a supported version of the Sun ONE Application Server 7 already installed. The Solaris™ 9 Update 2 Operating Environment, for example, includes an installation of Sun ONE Application Server 7.

You can obtain the Sun ONE Studio 5, Standard Edition software from:

- The Sun ONE Studio 5, Standard Edition CD
- The Sun ONE Portal (<http://www.sun.com/software/sundev/jde/>)
- The Sun ONE Developer Resources portal (<http://forte.sun.com/ffj/>)
- Java™ 2 Software Development Kit (the J2SE™ SDK), version 1.4.1_02 or higher

The Sun ONE Studio 5, Standard Edition installer will not run if you do not have the J2SE SDK on your system. If you do have a J2SE SDK on your system, the installer will start and then verify whether the J2SE SDK you are using is a version required by the IDE and the Sun ONE Application Server 7 for your platform. If you do not have the required version, the installer will quit, displaying a message that you must install the correct version before proceeding. You can obtain the J2SE SDK from the same locations as the IDE.

- PointBase Network Server database software

This tutorial uses the PointBase database. PointBase is installed with the Sun ONE Studio 5, Standard Edition software, in the subdirectory that contains the Sun ONE Application Server 7 software. If your IDE and application server were installed separately, your application server may or may not include PointBase software. If it does not, you must download and install PointBase software manually. Instructions are provided in the *Sun ONE Application Server 7 Getting Started Guide*. Alternatively, Appendix C describes how to create the tutorial application with an Oracle database.

- The SQL script that creates the tutorial database table

The tutorial SQL scripts are described in Appendix B. Files of these SQL scripts are included in the application files of the CDSShopCart tutorial, available from the Sun ONE Studio 5 Developer Resources portal. “Creating the Tutorial Database Table” on page 27 describes how to install the tutorial database table in a PointBase database. Appendix C describes how to install the tutorial tables in an Oracle database.

- A web server

The tutorial is a web application, which requires a web server. This tutorial uses the web server component of Sun ONE Application Server 7.

- 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.

You can access general system requirements from the release notes or from the Sun ONE Studio 5 Developer Resources portal’s Documentation page at <http://forte.sun.com/ffj/documentation/>.

Starting the Software

This section describes how to start the Sun ONE Studio 5 IDE and Sun ONE Application Server 7 after the software has been installed.

Starting the IDE

There are several ways to start the Sun ONE Studio 5 IDE. Only one is described here. For more options, see the *Sun ONE Studio 5, Standard Edition Getting Started Guide*.

To start the IDE:

- **Start the Sun ONE Studio 5 IDE by running the program executable.**
 - On Microsoft Windows, choose Start → Programs → Sun Microsystems → Sun ONE Studio 5 SE → Sun ONE Studio 5 SE
 - On Solaris, UNIX, and Linux environments, run the `runide.sh` script in a terminal window, as follows:

```
$ sh $studio-install-directory/bin/runide.sh
```

The `$studio-install-directory` variable stands for the IDE's home directory, which is by default `$HOME/studio5_se` (UNIX standard user) or `/opt/studio5_se` (UNIX superuser).

Starting the Application Server

Before starting this section, you must have write access to an application server domain. The default domain is created during installation and requires superuser privileges (administrator privileges on Microsoft Windows systems or root privileges in Solaris or Linux environments) to access. If your userid has superuser privileges, you can start the application server using all default settings, as described in the next section. Standard users (without superuser privileges) must use the procedures described in “Starting the Admin Server (Standard User)” on page 23.

Starting the Admin Server (Superuser)

If you have started the admin server previously, confirm whether it is running. See “Confirming Sun ONE Application Server 7 as the Default Server” on page 27 for information. If this is the first time you have started the admin server, start with this section.

To start the admin server:

- 1. In the IDE, select the Runtime tab of the Explorer.**

The Runtime pane of the Explorer displays the Server Registry node, among other nodes. This node contains subnodes for all the installed web and application servers, and a node showing which servers are the default servers.

- 2. Select the Server Registry node.**

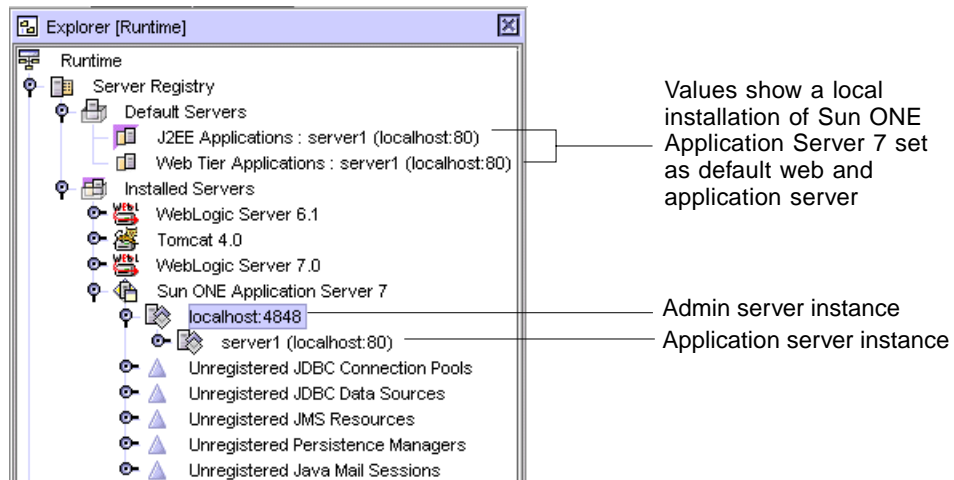
A query window pops up, asking whether you want to start the admin server. This refers to the default domain’s admin server, which can only be run by a privileged user.

- 3. Click OK to start the default admin server.**

The IDE starts the default admin server and configures Sun ONE Application Server 7 as the IDE’s default application server.

- 4. Expand the Server Registry node, the Installed Servers node, and the Sun ONE Application Server 7 node.**

The Server Registry in the Explorer looks like this:



Now, start the server instance, as described in “Starting the Application Server Instance” on page 26.

Starting the Admin Server (Standard User)

If your userid does not have superuser privileges, a superuser must create a domain for you before you start this section. The procedures are described in the *Sun ONE Studio 5, Standard Edition Getting Started Guide*.

If you have started the IDE previously, created an admin server, and started it, confirm now whether it is running by using the procedures described in “Confirming Sun ONE Application Server 7 as the Default Server” on page 27. If this is the first time you have started the admin server, start with this section.

Before starting the procedures described in this section, you will need values of several properties of your domain. Your administrator can provide these for you. Use the following table to record your values to these properties.:

TABLE 1-1 Admin Server Property Values

Admin Server Properties	Your Value
Admin Server Host	
Admin Server Port	
User Name	
User Password	
Domain	

To start the admin server:

1. In the IDE, select the Runtime tab of the Explorer.

The Runtime pane of the Explorer displays the Server Registry node, among other nodes. This node contains subnodes for all the installed web and application servers, and a node showing which servers are the default servers.

2. Select the Server Registry node.

A query window pops up, asking whether you want to start the admin server. This refers to the admin server of the default domain, which can only be run by a privileged user.

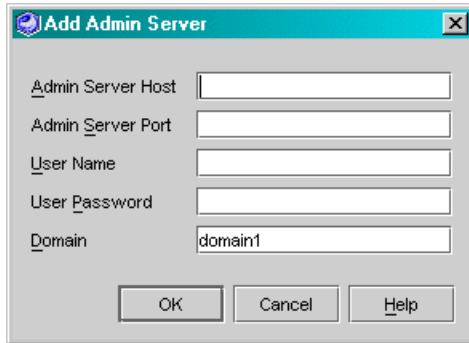
If you click OK, this action creates and starts an admin server that you cannot use. Click Cancel.

3. Add your admin server to the IDE.

a. Expand the Server Registry node and the Installed Servers node.

b. Right-click the Sun ONE Application Server 7 node and choose Add Admin Server.

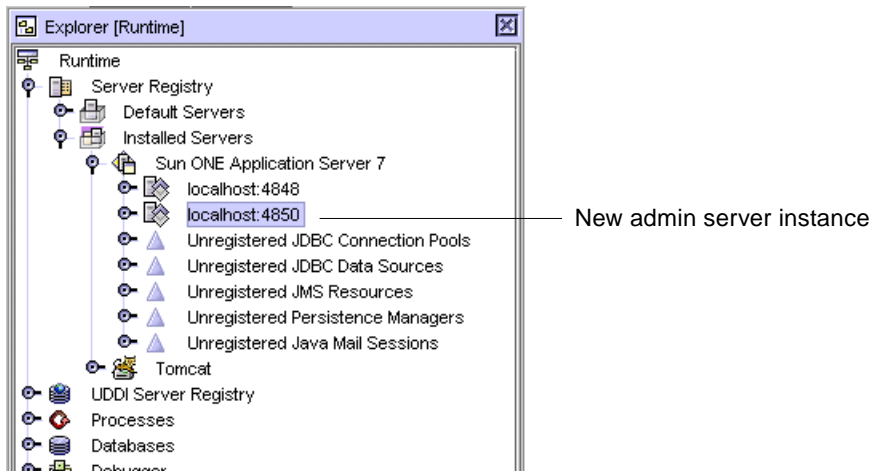
The Add Admin Server dialog box is displayed.



c. Type in your values from TABLE 1-1 and click OK.

If an error message appears, stating that the IDE could not locate the admin server, but will start it if it is local, click OK to close the error window. A progress window appears, showing the admin server process starting up.

A new admin server node is generated in the Explorer. In the following screen shot, the new admin server's host is localhost and port number is 4850.



4. Create an application server instance.

a. Right-click the new admin server node and choose Create a Server Instance.

The Enter Server Instance Values dialog box is displayed.

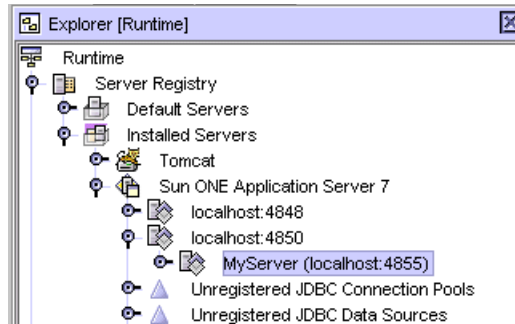
b. Type in a name and port number.

For example, you could type `MyServer` and `4855`.

Note – On Solaris and Linux systems, port numbers below 1024 are reserved. Use a port number above 1023. On all systems, do not use the port number used by the default application server, which is 80.

c. Click OK.

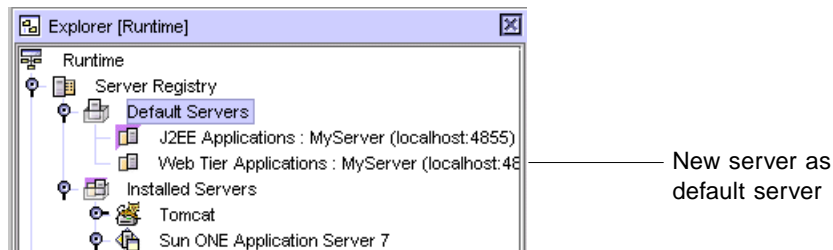
This action starts the admin server, which you can verify by messages in the output window and status bar. The new server instance is created in the IDE.



5. Set the default application and web server by right-clicking the new server instance and choosing Set As Default.

6. Expand the Default Servers node to verify this action.

The default servers for J2EE applications and web tier applications show the new server as the default.



Now start the server instance, as described in the next section.

Starting the Application Server Instance

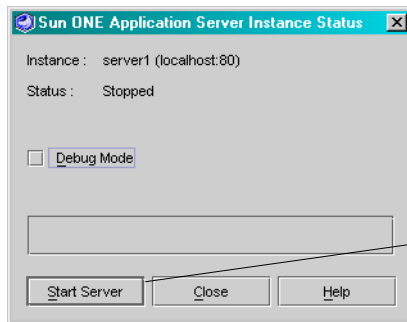
When you are test deploying and deploying applications during development, the IDE starts the application server instance automatically as long as the admin server is running. In this section, you start the application server instance manually in order to perform some operations described later in this chapter.

All users start the server instance as follows:

1. **Right-click the application server node and choose Status.**

Note – If this node is not displayed, select the admin server instance node and choose Refresh.

The Sun ONE Application Server Instance Status dialog box is displayed, as shown (your instance label may be different).



Start Server button

2. **Click the Start Server button.**

(If the dialog box has a Stop Server button, the server is already running.)

On Microsoft Windows systems, a command window appears, displaying progress messages.

The server is started when the Server Instance Status window displays Status: Running.

3. **Click Close on the instance status dialog box.**

Now, proceed to “Creating the Tutorial Database Table” on page 27.

Confirming Sun ONE Application Server 7 as the Default Server

If you have started Sun ONE Application Server 7 before, this is how you confirm that it is still the default server:

1. In the IDE, select the Runtime tab of the Explorer.

2. Expand the Server Registry node and its Default Servers subnode.

If the Web Tier Applications node's label is *server-instance (server-hostname: server-port-number)*, as shown, then Sun ONE Application Server 7 is the default web server. Go to the next section. Otherwise, continue with the next step.



Value shows a local installation of Sun ONE Application Server 7 set as default web server

3. Find your web server instance under the Installed Servers node, right-click it, and choose Set As Default.

Your server is set as the default server for J2EE and Web Tier applications.

Creating the Tutorial Database Table

Before you can start the CDShopCart tutorial, you must create and install a database table in the PointBase Network Server database. Use the SQL script in Appendix B to create these tables. A script file, *CDCatalog_pb.sql*, is also available within the *cdshop.zip* file for the CDShopCart tutorial, available from:

<http://forte.sun.com/ffj/documentation/tutorialsandexamples.html>

Note – If your Sun ONE Studio 5 IDE and Sun ONE Application Server 7 were not installed together, actions you must take to connect the IDE and the application server to the PointBase database software are described in the *Sun ONE Studio 5, Standard Edition Getting Started Guide*. You must perform these before beginning the following procedure.

To install the tutorial table in a PointBase database:

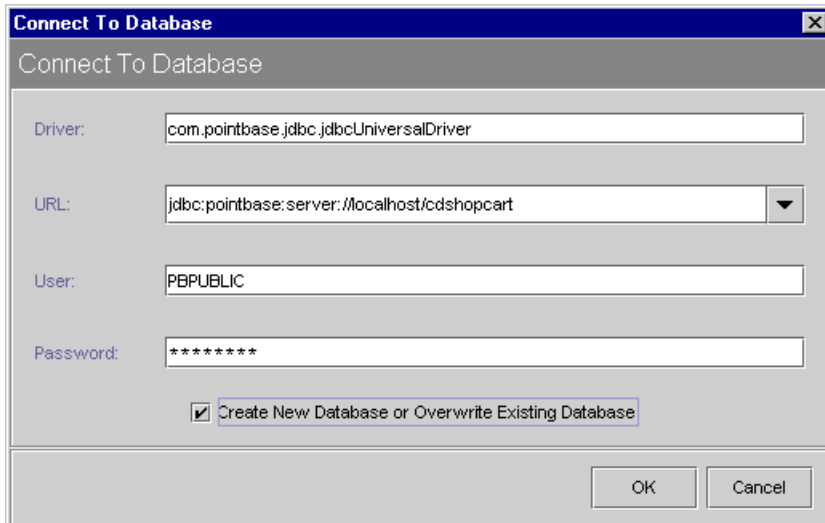
1. **Start the PointBase Server from the IDE by choosing Tools → PointBase Network Server → Start Server.**

The PointBase Network Server window is displayed. Minimize the window.

2. **Start the PointBase Console from the IDE by choosing Tools → PointBase Network Server → Start Console.**

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

3. **Change the word `sample` at the end of the URL field to `cdshopcart`, as shown.**



4. **Set the Create New Database option and click OK.**

The PointBase Console is displayed. Wait until the status message ending in “Ready” is displayed before proceeding.

5. **Copy the PointBase script from “Script for a PointBase Database” on page 100 and paste it into the SQL entry window of the Console.**

Alternatively, if you have the `CDCatalog_pb.sql` file from the tutorial source zip file, do this:

- a. **Choose File → Open to display the file browser dialog box.**
- b. **Use the file browser to locate the `CDCatalog_pb.sql` file and click Open.**

6. Choose SQL → Execute All.

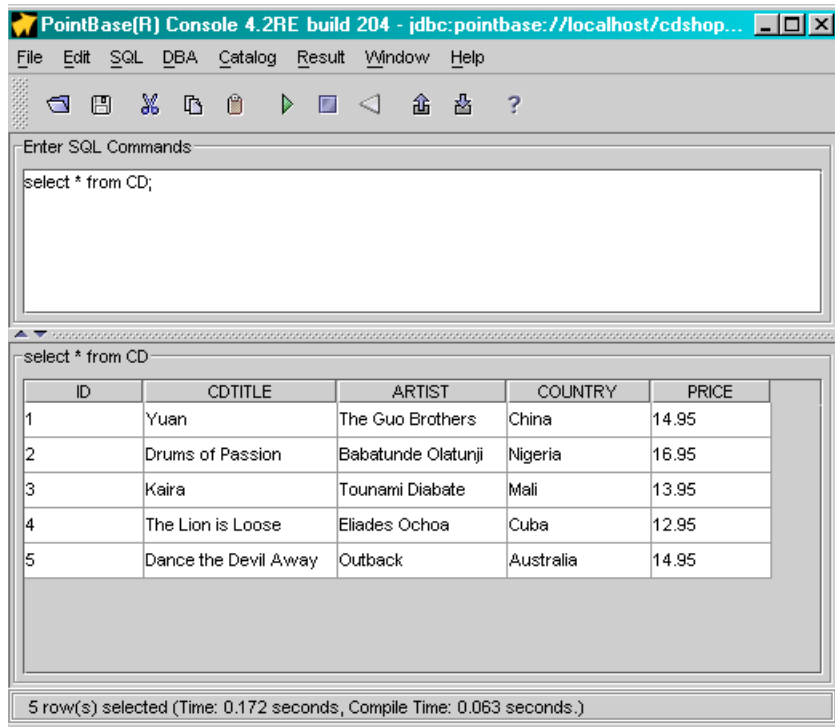
The message window confirms that the script was executed. (Ignore the initial message beginning “Cannot find the table...” This is a harmless error and appears because there is a DROP statement for a table that has not yet been created yet. This DROP statement will be useful in the future if you want to rerun the script to initialize the table.)

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

```
select * from CD;
```

8. Choose SQL → Execute.

Your console should display the CD table.



Note – If your display does not look like this table, choose Window → Windows to change the display type.

9. Close the PointBase Console window.

The CDCatalog script creates the database schema shown in TABLE 1-2.

TABLE 1-2 CDCatalog Database Table

Table Name	Columns	Primary Key	Other
CD	id cdtitle artist country price	yes	

The CD table is populated with the records shown in TABLE 1-3.

TABLE 1-3 CD Table Records

ID	CDtitle	Artist	Country	Price
1	Yuan	The Guo Brothers	China	14.95
2	Drums of Passion	Babatunde Olatunji	Nigeria	16.95
3	Kaira	Tounami Diabate	Mali	13.95
4	The Lion is Loose	Eliades Ochoa	Cuba	12.95
5	Dance the Devil Away	Outback	Australia	14.95

Now you are ready to start the tutorial application. Either continue to Chapter 2 to get an overview of the application you will build, or go directly to Chapter 3 and start building it.

Introduction to CDShopCart

In the process of creating the tutorial example application, you learn how to build components of a web application with Sun ONE Studio 5, Standard Edition features.

This chapter describes the application you will build, first laying out its requirements and then describing an architecture that will fulfill those requirements. The final section describes how you use Sun ONE Studio 5 features—web module constructs and Apache tag libraries—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 33
- “Architecture of the Tutorial Application” on page 37
- “Overview of Tasks for Creating the Tutorial Application” on page 40

Functionality of the Tutorial Application

The tutorial example application, CDShopCart, is a simple online shopping cart application for the purchase of music CDs. The customer uses a web browser to interact with the application’s interface, as follows:

1. The customer selects items from a catalog page to add to a shopping cart.
2. The customer can add more items from the catalog page or remove existing items from the shopping cart.
3. When the customer is ready to make a purchase, the application displays a Thank You message for the order and ends the session.
4. The customer can then exit the application or return to the order page to start a new shopping session.

Application Scenarios

The interaction of the CDShopCart application begins with the customer's visit to the application's catalog page and ends when the customer either completes an order or quits the site. The following scenarios describe a few of the customer's interactions with the CDShopCart application. Walking through these scenarios illustrates the requirements of the application, as well as interactions that happen within the application.

1. The customer starts the application by pointing the browser to the URL of the application's home page.

The home page is the CD Catalog List page, which displays a list of available music CDs and their associated information: title, the CD id number, the name of the performing artist, the artist's country, and the price.

2. The customer selects a CD for purchase by clicking the Add button associated with a CD.

This action causes the application to display a Shopping Cart page showing the selected CD title, its ID number, and price.

3. The customer selects more CDs for purchase.

The customer clicks the Resume Shopping button on the Shopping Cart page, which causes the application to redisplay the CD catalog page for further selections. The customer can repeat this sequence as many times as she likes, even adding the same CD multiple times (which adds more rows of the same CD to the cart—there is no Amount column for each CD).

4. The customer removes an item from her shopping cart by clicking the Delete button associated with the item on the Shopping Cart page.

Clicking this button redisplay the shopping cart minus the item, unless she deleted the last CD in the cart. When the last CD is removed, a page is displayed that reports that the cart is empty.

5. The customer can click the Resume Shopping button on the page to return to the CD Catalog List page, or the Cancel Order button to end the session. (The Cancel Order page is discussed in Scenario 7.)

6. The customer decides to make a purchase and clicks the Place Order button on the Shopping Cart page.

This action displays a Thank You message and ends the session. The customer can click the Resume Shopping link on the message page to start another session, or leave the application by closing the browser or going to a different URL.

7. The customer cancels an order at any time by clicking the Cancel Order button on the Shopping Cart page.

This causes the application to display a Cancel Order message page that ends the session. The Cancel Order page includes a Resume Shopping button, in case the customer wants to start a new session.

Application Functional Specification

Given the kinds of scenarios in which the CDShopCart application would be used, the following items list the main functions for a user interface of an application that supports these shopping interactions.

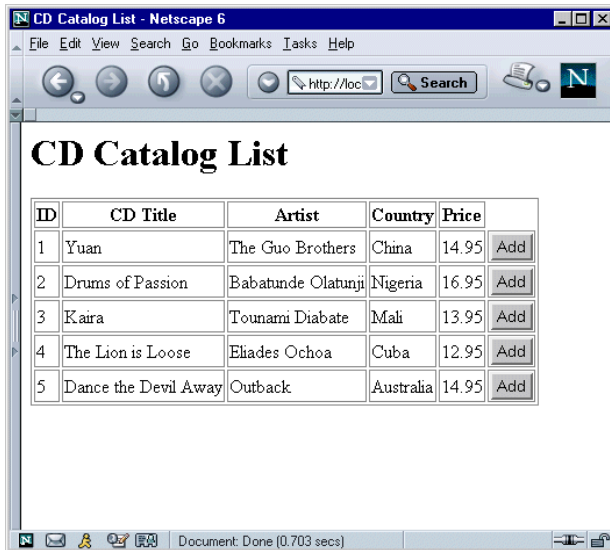
- A set of links to navigate from page to page
- A master view of all the site's offerings through a displayed list
- A view of items selected for purchase
- Buttons on the catalog page for adding each item
- Buttons on the shop cart page for removing items
- A button on the shop cart page to initiate checkout
- A button on the shop cart page to cancel the order
- A button on the checkout page to return to the home page to begin a new order
- A button on the empty cart page to return to the home page
- A button on the empty cart page to cancel the order

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 31 are realized.

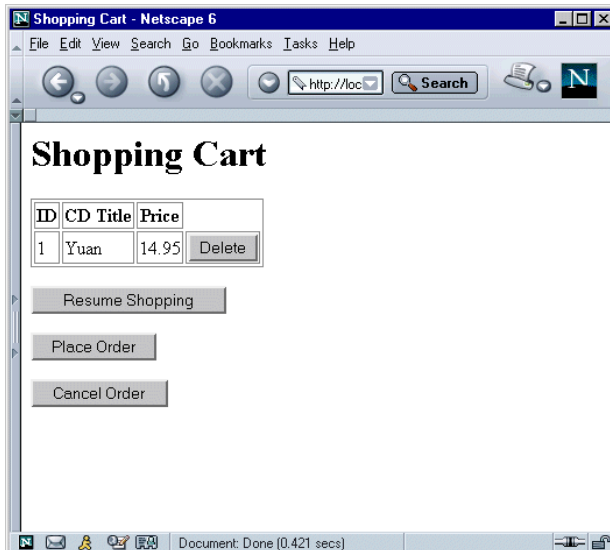
To run the CDShopCart application:

1. **The application starts with a CD Catalog List page that displays a list of CD titles.**
This page is created with the ProductList JSP page.



2. To add a CD to your shopping cart, click the Add button in that CD's row.

This action displays the Shopping Cart page with the selected CD on it. This page is created by the ShopCart JSP page.



3. To add another CD, click Resume Shopping, which takes you back to the CD Catalog List page.

4. Click Add on the same or a different CD.

The Shopping Cart page is redisplayed with an additional selection.

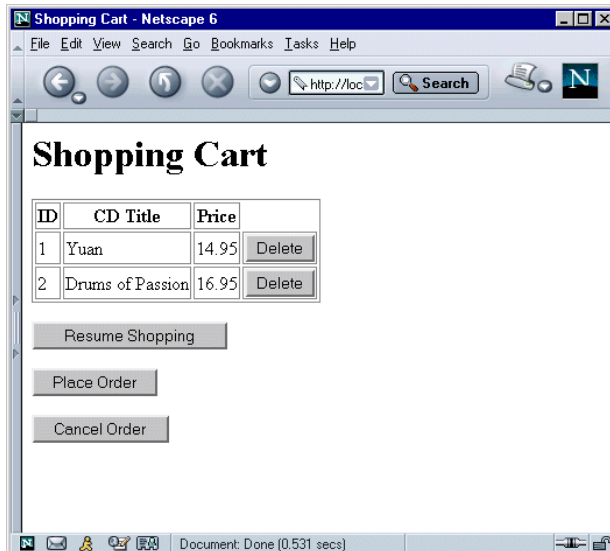
5. Repeat Step 2 and Step 3 until you have all the CDs you want to purchase.

The Shopping Cart displays your items. If you have chosen more than one of the same item, these are displayed as separate rows.



6. To remove an item, click the item's Delete button.

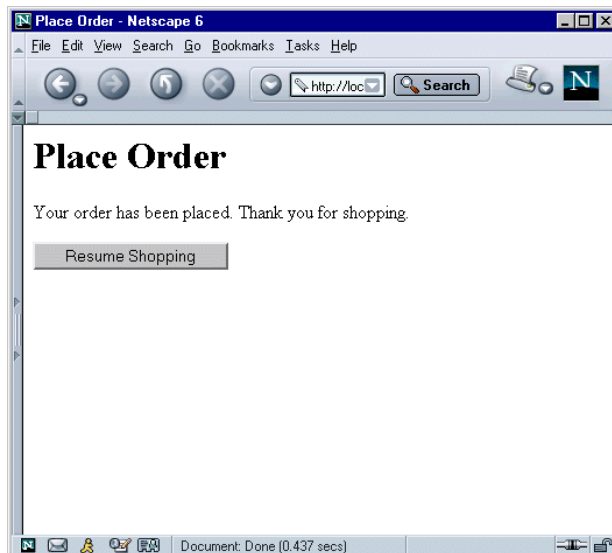
The table is redisplayed minus the removed item.



If you remove the last item in the table, the Empty Cart JSP page is displayed:



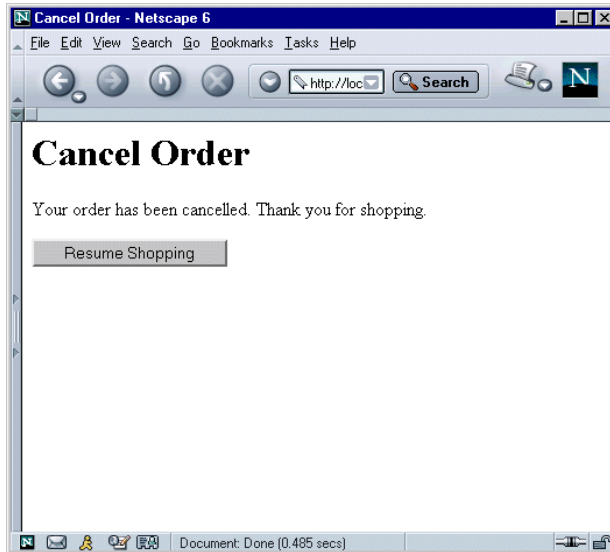
7. Click the **Resume Shopping** button to return to the CD Catalog List page.
8. To place an order, click the **Place Order** button on the Shopping Cart page. The Place Order page is displayed. This page is created by the PlaceOrder JSP page.



You can either exit the application by pointing the browser at a different URL, or start a new session by clicking the Resume Shopping button.

9. Or, to cancel your order, from the Shopping Cart page, click the Cancel Order button.

The Cancel Order page is displayed. This page is created by the CancelOrder JSP page.



You can start a new session by clicking the Resume Shopping button.

Architecture of the Tutorial Application

The CDShopCart application is a web-centric application that uses a web client to send requests to and receive results from a web application. A *web application* is a bundle of web components and their supporting classes, beans, and files. *Web components* are server-side J2EE components, such as Servlets and JSP pages.

The CDShopCart application consists of a single web module. A *web module* is the smallest deployable and usable unit of web resources in a J2EE application. A feature introduced in the *Java Servlet Specification Version 2.3* and implemented by the Sun ONE Studio 5 IDE is the web module construct, which automatically creates the required directory structures, default versions of required data objects, and other special services required by the web module.

For more information on web modules and related concepts, see *Building Web Components*. For information specific to the web module construct, see the Developing Web Modules section under JavaServlet Pages and Servlets in online help.

FIGURE 2-1 shows the CDShopCart application elements and their relation to one another.

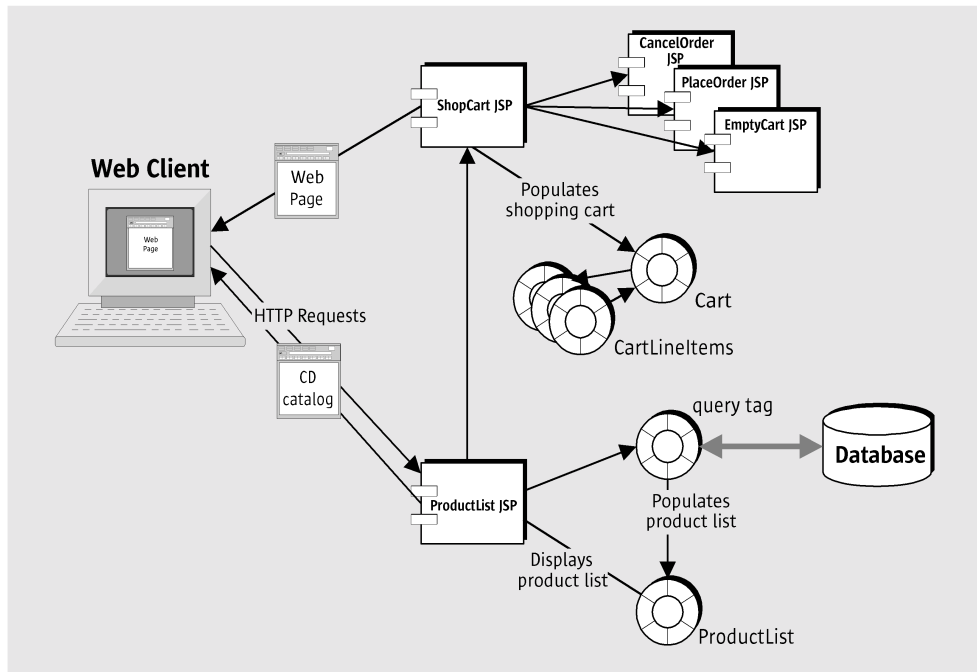


FIGURE 2-1 Architecture of the CDShopCart Application

Application Elements

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

- The *client* component
 - The client component is a web browser that displays the application pages.
- The *service* component (a web module) that includes:
 - A ProductList JSP page that retrieves the product data from a database and displays it in a table on the CD Catalog List web page. ProductList also provides an Add button by which a user can add a CD to the shopping cart.
 - A ShopCart JSP page that displays CDs selected for purchase in a table on the Shopping Cart web page, and provides Delete, Place Order, Cancel Order, and Resume Shopping buttons.
 - An EmptyCart JSP page that displays a message when the customer deletes the last item in the shopping cart. Includes a Resume Shopping button.

- A CancelOrder JSP page that displays a message that the order has been canceled and a Resume Shopping button for returning to the ProductList JSP page.
- A PlaceOrder JSP page that displays a message that the order has been placed and a Resume Shopping button for returning to the ProductList JSP page.
- A Cart bean that represents the items selected for purchase.
- A CartLineItem bean that represents a cart line item.
- A CartLineItemBeanInfo beaninfo component to specify that id and price are properties of CartLineItem, despite having overloaded setter methods

Service Component Details

The service component of the CDShopCart application is a web module that includes four JSP pages that coordinate the application's behavior, given input from the client. Supporting elements to the web module include JavaBeans elements and an HTML page file.

- ProductList JSP page

This page locates the session for the current user or creates one if it doesn't exist. `ProductList` uses SQL tags from the Apache Foundation's JSP Standard Tag Library (JSTL) to access the list of CDs from the database and core tags to display them in a table. `ProductList` also provides an Add button on each CD line item it displays.

- ShopCart JSP page

When the user clicks the Add button on the ProductList page, the data of the line item is passed to this JSP page, which instantiates a Cart object consisting of CartLineItem objects, then uses core tags from the JSTL tag library to display them in a table. ShopCart provides a Delete button on each cart item. When this button is clicked, the page uses a scriptlet to remove the item, update the table data, and redisplay it. If the item removed is the last item in the cart, ShopCart forwards to the EmptyCart page. ShopCart also provides Resume Shopping, Cancel Order, and Place Order buttons. These buttons forward to the ProductList page, the CancelOrder page, and the PlaceOrder page, respectively.

- Cart JavaBeans component

This bean has a `lineItems` attribute and includes the methods for getting and removing the CartLineItem objects. This bean is imported by the ShopCart page.

- CartLineItem JavaBeans component

This bean has CD-related attributes, and includes methods for getting and setting attributes of Cart line items (ID, title, artist, country, and price).

- CancelOrder JSP page

This JSP page is called when the user clicks the Cancel Order button on the ShopCart page. This page invalidates the session, displays a message that the order is cancelled, and provides a Resume Shopping button to return to the ProductList page.

- EmptyCart JSP page

This JSP page is called when the user removes the last item from the ShopCart page. EmptyCart displays a message that the cart is empty and includes a Resume Shopping button.

- PlaceOrder JSP page

This JSP page is called when the user clicks the Place Order button on the ShopCart page when there are items in the cart. PlaceOrder displays a message that the order has been placed, invalidates the session, and includes a Resume Shopping button.

Overview of Tasks for Creating the Tutorial Application

The tutorial consists of one chapter, in which you create the basic application. Before you can create the tutorial application, you must have the Sun ONE Studio 5 software installed and set up to run, and the tutorial database tables installed, as described in Chapter 1.

In Chapter 3 you learn how to use the following Sun ONE Studio 5, Standard Edition features:

- Web modules (creating, developing, and test running)
- Embedded JSTL tags for connecting to and interacting with a database
- Embedded JSTL tags for iterating through and presenting the retrieved data

In addition, you create the supporting elements: several beans and an HTML page.

Creating a Web Module

The Sun ONE Studio 5 IDE provides tools for automatically creating the hierarchical directory structure of a web application. This structure is the web module. You develop the entire CDSShopCart application within a single web module construct.

The tutorial doesn't try to provide complete information about how to develop a web module. The section "Creating a Web Module" on page 43 serves as an introduction to the subject, outlining the basic elements of the structure, and the (very easy) method for creating it. When you want to know more about how to develop web modules, see *Building Web Components* and online help.

Using the JSTL Tag Libraries

Within the CDShopCart web module, you create the ProductList JSP page to fetch the CD catalog data to display on the CD Product List web page. You then create the ShopCart JSP page to display CDs that the user selects for purchase. You use JSTL tags for database access and data presentation functions to accomplish this.

The section "Using JSP Tags to Fetch and Display Database Data" on page 47 describes how to use the SQL tags to make the JDBC connection to the database and fetch the CD data. It then describes how to use core tags to iterate through the resulting data, so that ProductList can display it in an HTML form on the web page.

The ShopCart JSP page displays CD data passed to it by the ProductList JSP page. The section "Adding Code to Add or Remove an Item From the Shopping Cart Table" on page 68 demonstrates how to use core JSTL tags to iterate through the passed values to find the individual field values, so they can be displayed in the correct columns in the cart table.

Creating the Supporting Elements

The supporting elements for the ShopCart JSP page are two beans (Cart and CartLineItem) and three JSP pages (CancelOrder, PlaceOrder, and EmptyCart).

The section "Creating the CartLineItem Bean" on page 59 shows you how to create a bean whose object holds the parameters of a line item passed to ShopCart from ProductList when a user clicks the Add button on the item. Then, in "Creating the Cart Bean" on page 65, you learn how to create a bean whose object holds the accumulated line items that have been selected. The Cart bean has methods for adding and removing line items from the cart.

In "Creating the Empty Cart Page" on page 74, you create a JSP page that displays a message that the cart is empty. This is to avoid displaying an empty form on the Shopping Cart page.

You create two more JSP pages that hold minor logic compared with ProductList and ShopCart. In “Creating the Place Order Page” on page 75, you create a JSP page that thanks the user for placing an order and then invalidates the session. In “Creating the Cancel Order Page” on page 77, you create a similar page that announces that an order is canceled and invalidates the session.

Testing the Application

Throughout this chapter, you test each element just after you create it. The IDE automatically deploys a web module to its internal container when you execute one of the web module’s components.

End Comments

The tutorial application is designed to be brief enough for you to create in a relatively short time (a day or so). This places certain restrictions on its scope. For example:

- There is no error handling.
- There are no debugging procedures.
- There is no description of how to create the WAR file for deployment.

Future releases will have these procedures.

Although the tutorial application 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 CDShopCart application is accessible from the Sun ONE Studio 5 Developer Resources portal at <http://forte.sun.com/ffj/documentation/tutorialsandexamples.html>

Creating the CDShopCart Application

This chapter describes, step by step, how to create the CDShopCart application. Before you can create the tutorial application, you must have the Sun ONE Studio 5 software installed and set up to run, and you must have the tutorial database table installed, as described in Chapter 1.

This chapter is organized under the following topics:

- “Creating a Web Module,” which follows
- “Using JSP Tags to Fetch and Display Database Data” on page 47
- “Creating the Shopping Cart Page and Supporting Elements” on page 58
- “Creating the Three Message Pages” on page 73

You test each component as you create it. By the end of this chapter, you will be able to run the basic application, as described in Chapter 2.

Tip – Complete source code for all JSP pages and JavaBeans components described in this chapter is found in Appendix A.

Creating a Web Module

The CDShopCart application is a web application. Web applications consist of web modules. The CDShopCart application is a very simple application, containing only one web module.

This section shows you how to implement the shopping cart features within a web module by using the Sun ONE Studio 5, Standard Edition IDE.

What Is a Sun ONE Studio 5 Web Module?

According to the *Java Servlet Specification, version 2.3*, “a web application exists as a structured hierarchy of directories.” The root of this hierarchy is the document root, which holds all the files that are part of the web application. The hierarchy also includes a special non public subdirectory, the `WEB-INF` directory, for items that are related to the web application but are not to be served directly to the client. Items in the `WEB-INF` directory include the web deployment descriptor (the `web.xml` file) and servlet and utility classes used by the web application loader for loading classes.

Your application’s files must eventually be part of a web module structure to package them as a WAR file (a Web ARchive format file) for delivery into a web container. The Sun ONE Studio 5 IDE’s web module feature automates much of the process of creating the required directory hierarchy, as well as filling in the hierarchy with default versions of some of the objects.

Note – This tutorial does not try to provide complete information about developing web modules. For that task, see the *Building Web Components* book. Also, consult the Sun ONE Studio 5 online help for more details on web modules.

Creating the CDShopCart Web Module

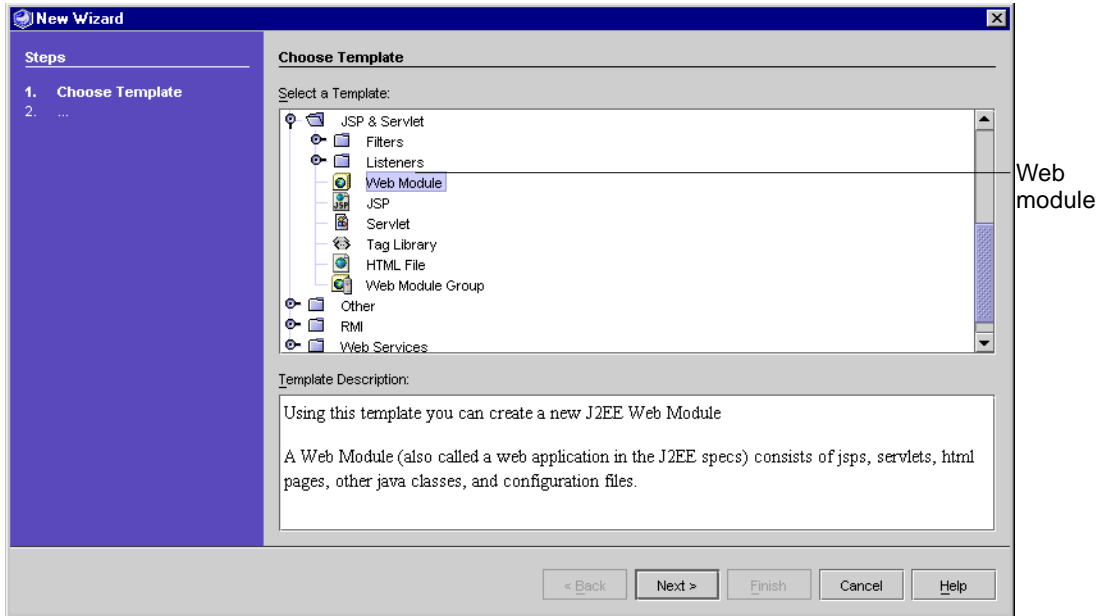
In this section, you create the web module for the CDShopCart application. You’ll use the Sun ONE Studio 5 web module feature to build this web module directory from scratch, although you could also convert an existing directory into a web module.

To create the CDShopCart web module:

- 1. In the Filesystems pane of the Explorer window, select the default (or any) file system and choose File → New to display the New wizard.**

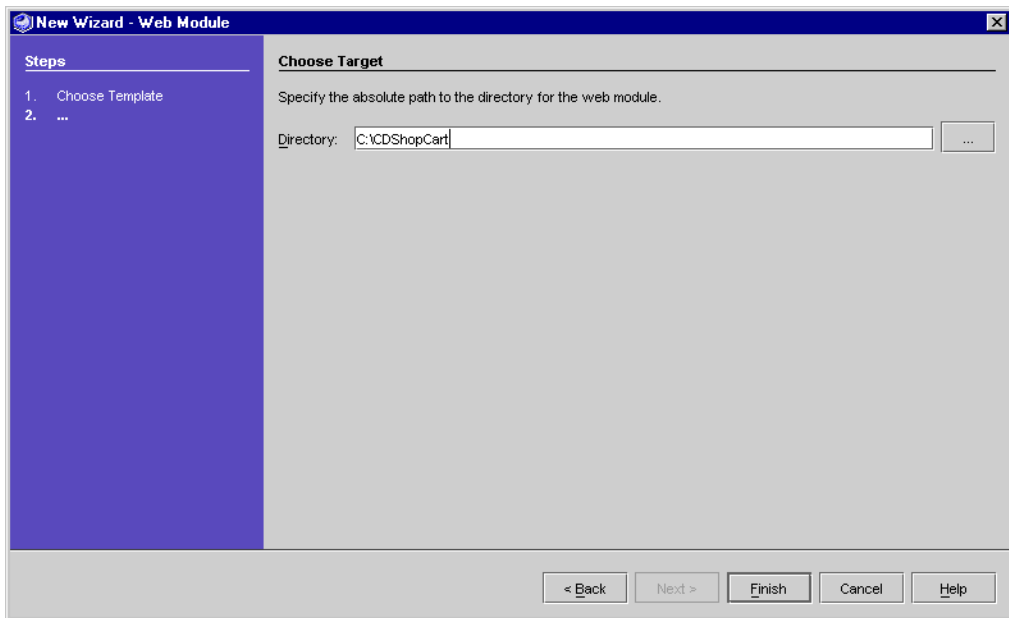
You can use this wizard to create many different types of objects from the provided templates.

- 2. Open the JSPs & Servlets node and select Web Module.**



3. Click Next to display a page for specifying the web module directory name.

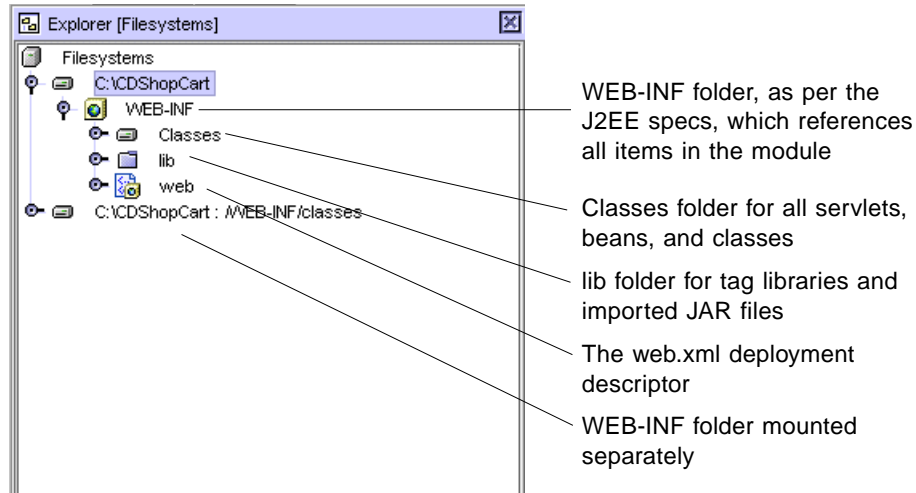
4. In the Directory field, erase the default directory filespec and type *your-directory\CDSShopCart*.



5. Click Finish.

The new CDShopCart web module is created in the Explorer. A dialog box appears, stating that an alternate view of the web module is installed in the Default Project window. Click OK to dismiss this message.

6. Open the nodes in the web module to reveal what has been created automatically.



Now you are ready to create the first component of the application, the ProductList JSP page.

Setting the Web Application's Context Root

In order to deploy the CDShopCart web application, the web server must know its context root. You set this as a property of the WEB-INF folder, as follows:

1. Display the WEB-INF folder's properties.

The properties window is beneath the Explorer. This is a static property window and displays the properties of whatever node is selected. Select the WEB-INF file to display its properties. Alternatively, right-click the WEB-INF folder and choose Properties. A dynamic properties window opens, which only displays the properties of the WEB-INF folder, regardless of what is selected.

2. Click in the value field of the Context Root property.

3. Type /CDShopCart and press Return (or Enter).

Using JSP Tags to Fetch and Display Database Data

In this section, you create a JSP page named `ProductList` that fetches and displays the CD product data. To connect to the database, retrieve the table data, and format the data for display, you'll use JSP Standard Tag Library (JSTL) tags. JSTL is from the Jakarta Project, a project of the Apache Foundation. JSTL is embedded in Sun ONE Studio 5.

For complete information on JSTL, including descriptions, examples, and a tutorial, see the Jakarta Project web site at <http://jakarta.apache.org/taglibs/index.html>.

What Is a JSP Tag?

The body of a JSP file can contain two types of code: fixed template data and elements.

- Fixed template data—code type not known to the JSP container; this data passes through the HTTP response unchanged.

Examples of fixed template data are XML and HTML code. You'll use HTML code in the `CDSShopCart` application to create headings, titles, tables, and buttons.

- Element types—include three different types:
 - Directives—used to declare global information about the JSP page, such as which packages to import and whether the JSP page must join a session.
 - Scripting elements—enable you to embed Java code within a JSP file.
 - JSP tags—XML-style elements that provide a way to work with Java objects without having to write Java code.

Java classes associated with each tag implement the tag's functionality.

Standard Tags and Custom Tags

Standard tags, which are defined in the JSP specification document, are available in any JSP container. Custom tags are tags defined in an XML document called a *tag library*. A custom tag library is made available to the JSP page by means of a declaration in a directive element.

JSTL Tags

The JSP Standard Tag Library is a custom tag library produced by the Jakarta Project. Embedded in the Sun ONE Studio 5 IDE are JAR files that contains the JSTL tags and supporting classes and interfaces. This files are:

- `standard.jar`—contains multiple JSTL tag libraries, including core and SQL tags
- `jstl.jar`—contains supporting interfaces and classes for the JSTL libraries in `standard.jar` used by CDSShopCart

The two tag library descriptor (TLD) files you will use in these JAR files are `sql.tld` (for database actions) and `core.tld` (for everything else).

Using JSTL Tags

To use any JSP tags, you need to declare the tag library, and then follow the prescribed syntax of the particular tag.

For complete information on JSTL syntax conventions, see the JSTL documentation, available from the Jakarta Project web site at <http://jakarta.apache.org/taglibs/index.html>.

Using the `taglib` Directive

To use tags in a JSP page, you must first declare the tag library with a `taglib` directive.

The `taglib` directive must declare that the page uses the tag library of a given URI (Uniform Resource Identifier) and specify the tag prefix that is used in calls to actions in the library. The URI and the prefix for JSTL tags are defined with Apache JSTL conventions.

The generic syntax for the directive is:

```
<%@ taglib prefix="prefix" uri="http://java.sun.com/jstl/taglibname"
```

For example, to declare the `core` `taglib`:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core
```


Using Tag Syntax

JSP tags are based on XML syntax and have one of two forms:

- Start tag (the element name) plus possible attribute/attribute value pairs, an optional body, and a matching end tag
- Empty tag with possible attributes

An example start tag you'll use in CDSShopCart is the `query` tag, which fetches data from the data source. The `query` tag uses the following syntax:

```
<sql:query var="stored_query" dataSource="${dataSource}" >  
    body  
</sql:query>
```

An example empty tag you'll use is the `sql` library `setDataSource` tag, which creates a JDBC connection to a database. The `setDataSource` tag uses the following syntax:

```
<sql:setDataSource var="dataSource"  
    url="driver_url"  
    driver="driver_string"  
    user="user_id" password="pwd" />
```

A JSTL tag convention is to use “var” for any tag attribute that exports information about the tag. Note that “var” was chosen to emphasize the fact that this is a JSP variable, not a scripting variable, which uses the convention “id” for a variable.

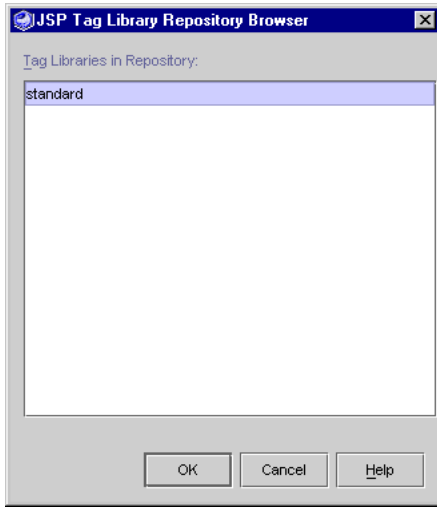
Adding JSTL Tag Libraries to the Web Module

Import the JSTL tag library JAR file, `standard.jar`, and the JAR file of supporting classes and interfaces, `jstl.jar`, to the CDSShopCart web module, because you will use actions that are implemented by these files.

To import JSTL tag libraries into the web module:

1. **In the Explorer, right-click the CDSShopCart web module and choose Add JSP Tag Library → Find in Tag Library Repository.**

The JSP Tag Library Repository Browser appears.

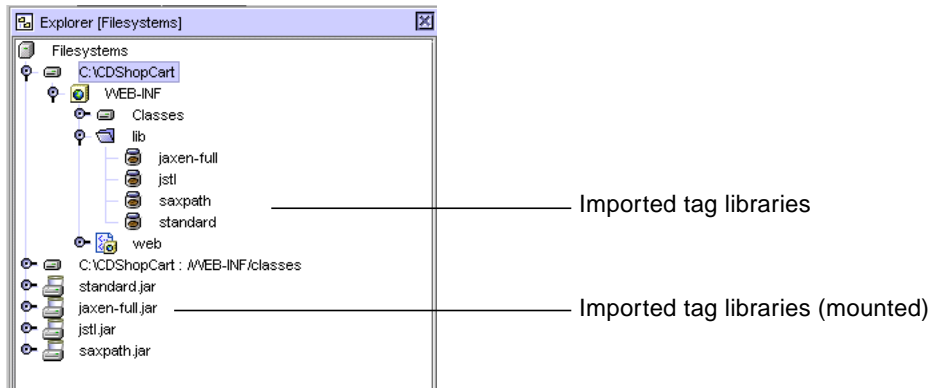


2. Make sure the `standard` file is selected and click OK.

In the Explorer, expand the `lib` node under the `WEB-INF` node.

3. Check the files are under the `lib` node.

The `jstl.jar` and `standard.jar` files, which contain the tags you use in this tutorial, are displayed under the `lib` node. Two additional jar files are also displayed. These contain APIs required for XML tags, which you will not use. All four files are also separately mounted. The Explorer looks like this:



Tip – Right-click the top-level Filesystems node and choose `Customize`. The window that opens displays all the files that are now mounted in your Sun ONE Studio 5 class path. You should be able to see the two JAR files in the list.

Creating the CD Catalog List Page

Create the mechanism for retrieving data from the database you installed in Chapter 1 and displaying it in a table for the user. This mechanism includes:

- A JSP page (ProductList) to hold all the code
- Tag library declarations for the tag libraries referred to in the code
- A setDataSource tag to connect to the database
- A query tag to fetch to CD data
- Iteration tags to display the CD data
- An Add button for each CD row

The page you create looks like FIGURE 3-1.

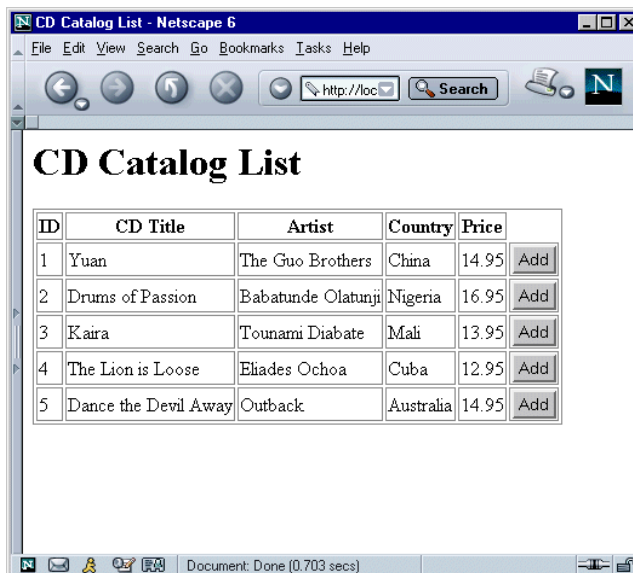


FIGURE 3-1 CD Catalog List Page

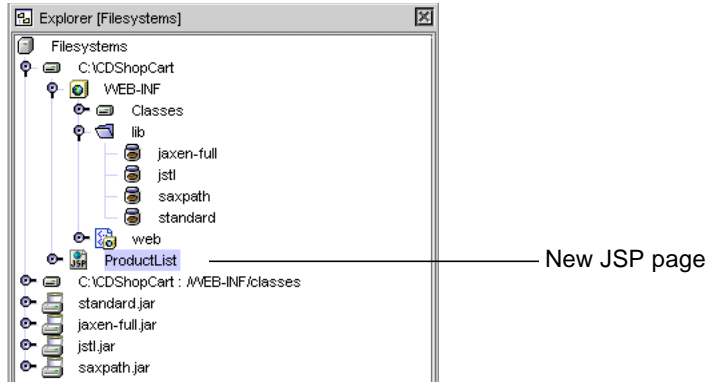
Creating the ProductList JSP Page

Create the page that uses the tags to retrieve the CD data from the database and display the data in a table. The title of this page is CD Catalog List, and the mechanism that produces it is the ProductList JSP page.

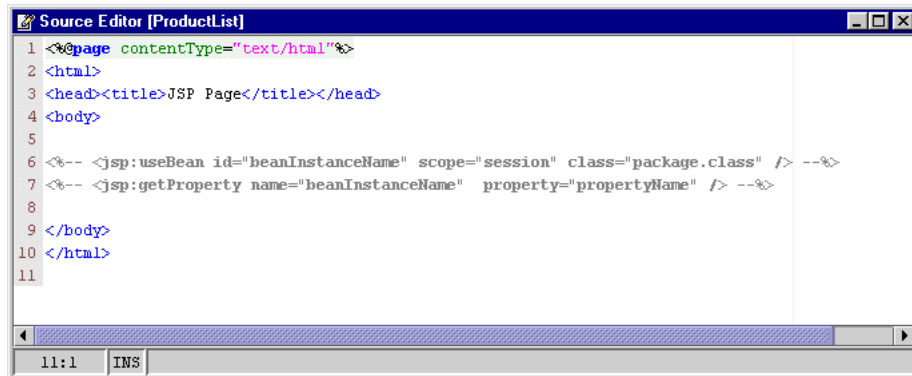
1. **In the Explorer, right-click the CDSShopCart web module and choose New → All Templates from the contextual menu.**

The New wizard appears.

2. Expand the JSPs and Servlets node, select the JSP node under it, and click Next. The New Object Page of the New wizard is displayed.
3. Type **ProductList** in the Name field and click Finish. The ProductList JSP page is displayed in the web module.



A JSP page skeleton is displayed in the Source Editor.



Declaring the Tag Libraries

You must first declare the tag libraries as described in “Using the `taglib` Directive” on page 48. Put the directive above the body of the JSP page, right under the page title. The following procedure shows how to change the title of the page and add the directives for the two tag libraries.

1. In the body of the **ProductList** page, change the document title to **CD Catalog List**:

```
<head><title>CD Catalog List</title></head>
```

2. Underneath this line, add directives to import the core and SQL actions tag libraries, as follows:

```
<%@taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>
```

Using the setDataSource Tag to Connect to the Database

The first tag you'll use is the sql library setDataSource tag, which creates a JDBC connection to a database. First add a page heading (with an <H1> tag), then add the setDataSource tag below the <body> HTML tag to connect to the database.

1. Below the <body> tag in the ProductList page, create a page title:

```
<body>
<h1> CD Catalog List </h1>
```

2. Below the header, create the JDBC connection.

- The following is for a PointBase driver:

```
<sql:setDataSource var="productDS"
url="jdbc:pointbase:server://localhost/cdshopcart"
driver="com.pointbase.jdbc.jdbcUniversalDriver"
user="PBPUBLIC" password="PBPUBLIC" />
```

- If you're using an Oracle database (using the thin driver), use this:

```
<sql:setDataSource var="productDS"
url="jdbc:oracle:thin:@hostname:port#:SID"
driver="oracle.jdbc.driver.OracleDriver"
user="userid" password="password" />
```

The default Oracle port number is 1521.

- If you're using a Microsoft SQLServer database with a Weblogic driver, use this:

```
<sql:setDataSource var="productDS"
url="jdbc:weblogic:mssqlserver4:database@hostname:port#"
driver="weblogic.jdbc.mssqlserver4.Driver"
user="userid" password="password" />
```

The default port number for SQLServer is 1433.

Using the query Tag to Fetch the CD Data

Now use the `sql` tag, query to query the database and get a single result set containing rows of data and store it in the `productQuery` result set. You'll then pass this JSP variable to iterator tags to display the results. The query tag supports the standard SQL statement `SELECT`.

Put the query tag just after the driver tag.

To query the database for all the CD data:

- **In the ProductList page, just after the driver tag, create a query to select all the CD data from the database (use the `productDS` `dataSource` created above):**

```
<sql:query var="productQuery" dataSource="${productDS}" >  
SELECT * FROM CD  
</sql:query>
```

Using the Iteration Tags to Display the Data

You need to create a table and fill the table cells with the data. First, create a table with HTML tags, then use the `forEach` tag to iterate through the data you just fetched. Use the `out` tag to fetch the field data of each row.

The syntax for `forEach` is:

```
<c:forEach var="$Current_collection_item" items="${Collection}" >
```

The `items` variable holds the current collection that is the target of the iteration, and `var` holds the current item of that collection. If the collection is a result set, then the current item is the result set object positioned at the current row. For example, to iterate over a row from a result named `myResultSet`:

```
<c:forEach var="row" items="${myResultSet.rows}" >  
<TR>  
<TD><c:out value="${row.col1}" /></TD>>  
<TD><c:out value="${row.col2}" /></TD>>  
<TD><c:out value="${row.col3}" /></TD>>  
</TR>  
</c:forEach>
```

To create the table and fill the cells with data:

1. Start a table for the CD data:

```
<TABLE border=1>
<TR>
<TH>ID</TH>
<TH>CD Title</TH>
<TH>Artist</TH>
<TH>Country</TH>
<TH>Price</TH>
</TR>
```

2. Next, use the forEach and out tags to populate the table:

```
<c:forEach var="row" items="{productQuery.rows}" >
<TR>
<TD><c:out value="{row.ID}"/></TD>
<TD><c:out value="{row.CDTITLE}"/></TD>
<TD><c:out value="{row.ARTIST}"/></TD>
<TD><c:out value="{row.COUNTRY}"/></TD>
<TD><c:out value="{row.PRICE}"/></TD>
```

You will add more to this code in the next section.

Creating the Add Button for Each CD Row

Each row of the CD table holds the data for one CD. To purchase a CD, the user clicks the Add button on a CD row. Create an HTML form to define the area for user input (clicking the button) and within this area, embed information that is passed to the Shopping Cart JSP page. Add the following code immediately after the previous code:

1. Create a form for the table within a cell:

```
<TD>
<form method=get action="ShopCart.jsp">
```

2. Specify the embedded information for product ID, title, and price:

```
<input type=hidden name=cdId value="<c:out value="\${row.ID}"/>">
<input type=hidden name=cdTitle value="<c:out value=
"\${row.CDTITLE}"/>">
<input type=hidden name=cdPrice value="<c:out value=
"\${row.PRICE}"/>">
```

3. Directly underneath the preceding code, specify the Add button:

```
<input type=submit name=operation value=Add>
```

4. End the form, the cell, and the row:

```
</form>
</TD>
</TR>
```

5. End the iteration and the table:

```
</c:forEach>
</TABLE>
```

6. Choose File → Save to save your work.

Tip – To see the completed source code of this page, see “ProductList.jsp Source” on page 82.

Testing the ProductList JSP Page

Test your work by compiling the ProductList JSP page, deploying the web application, then executing the ProductList page. The IDE automatically launches your default browser and displays the page.

To test the ProductList page:

1. If not already started, start the PointBase Network Server by choosing Tools → PointBase Network Server → Start Server.

A PointBase server window is displayed. Minimize it.

- 2. Make sure your Sun ONE Application Server 7 admin server is running and your application server is the default web server.**

See “Confirming Sun ONE Application Server 7 as the Default Server” on page 27 for information.

- 3. Make sure the Context Root property for the web module is set to /CDShopCart.**


Refer to “Setting the Web Application’s Context Root” on page 46.

- 4. In the Filesystems pane of the Explorer, right-click the CDShopCart web module and choose Build All from the contextual menu.**

Watch the message area in the output window for status messages. If all is well, you see “Finished.” If the output window displays errors, fix any problems and redo this step until you see the “Finished” message. Refer to “ProductList.jsp Source” on page 82 for the source code.

- 5. Right-click the CDShopCart web module and choose Deploy.**

When the application is deployed, the progress meter window closes.

- 6. Execute the ProductList JSP page by selecting it and clicking the Execute button in the toolbar ().**

Alternatively, choose Build → Execute, or right-click ProductList and choose Execute from the contextual menu.

If Sun ONE Application Server 7 is not already running, the IDE starts it (on Microsoft Windows systems, you see a command window with log messages, which you can minimize).

When the Servlet is running, the IDE opens the browser. After a few seconds, the CD Catalog List page is displayed, as in FIGURE 3-1.

Congratulations! You have successfully created a JSP page that uses JSTL tags to open a connection to a database and retrieve and display data from it. Now create the Shopping Cart page.

Creating the Shopping Cart Page and Supporting Elements

Now create the mechanism for displaying items selected for purchase from the CD Catalog List page. This mechanism includes:

- A bean (CartLineItem) to hold the attributes of the selected CD row passed as parameters from the ProductList page
- A beaninfo component (CartLineItemBeanInfo) to specify that id and price are properties of CartLineItem, despite having overloaded setter methods
- Another bean (Cart) to hold the CartLineItem objects
- The ShopCart JSP page to receive the Cart objects and display them as a row in a table
- A Delete button for each item displayed on the Shopping Cart page
- Cancel Order, Resume Shopping, and Place Order buttons with their implementations

When a few items have been selected, the Shopping Cart page you create looks like FIGURE 3-2.

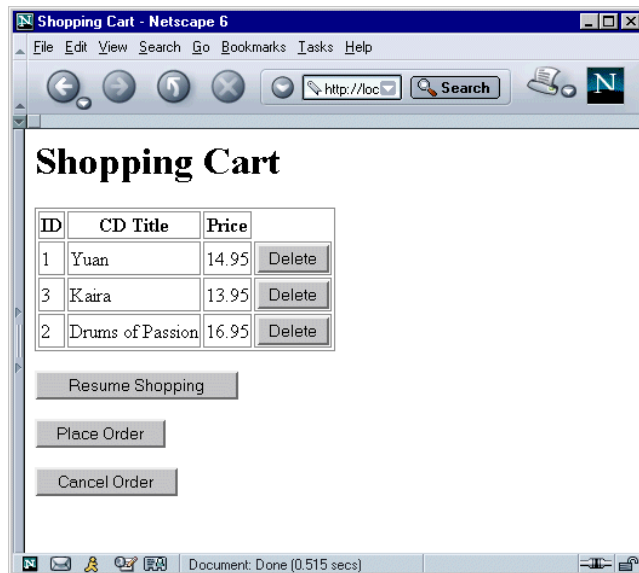


FIGURE 3-2 Shopping Cart Page

Creating the CartLineItem Bean

Create a line item bean whose object can hold the parameters passed to the Shopping Cart page from the CD Catalog List (ProductList) page. To do this, create three properties on the bean with their accessor methods.

Note – J2SE 1.4 requires all classes to be contained within packages, and this tutorial is compliant with J2SE 1.4. The classes subdirectory of the WEB-INF directory is not a package. Therefore, you must create a package under the classes subdirectory to hold your classes (beans). This package is called the ShopCart package in this tutorial.

1. Expand the WEB-INF node of the CDShopCart web module, right-click the Classes node, and choose New → Java Package.

2. Type ShopCart for the package name and click Finish.


The new ShopCart package is displayed in the Explorer.

3. Right-click the ShopCart package and choose New → All Templates.

The New wizard appears.

4. Expand the Java Beans node, select Java Bean and click Next.

5. In the Name field, type CartLineItem and click Finish.

The new CartLineItem bean is displayed in the Explorer, and its code in the Source Editor. The red badge near the bean () is a “need to compile” indicator. Don’t worry about this; you will compile later.

6. Expand the bean and its class to reveal its contents.

7. Right-click the Bean Patterns node and choose Add → Property.

8. In the New Property Pattern dialog box, define the cdtitle property.

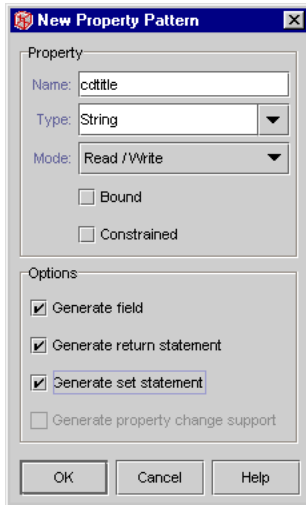
a. Type cdtitle in the Name field.

b. Select String for the Type.

c. Enable the following options:

- Generate field
- Generate return statement
- Generate set statement

The dialog box should look like this:



9. Click OK to accept the information and close the dialog box.

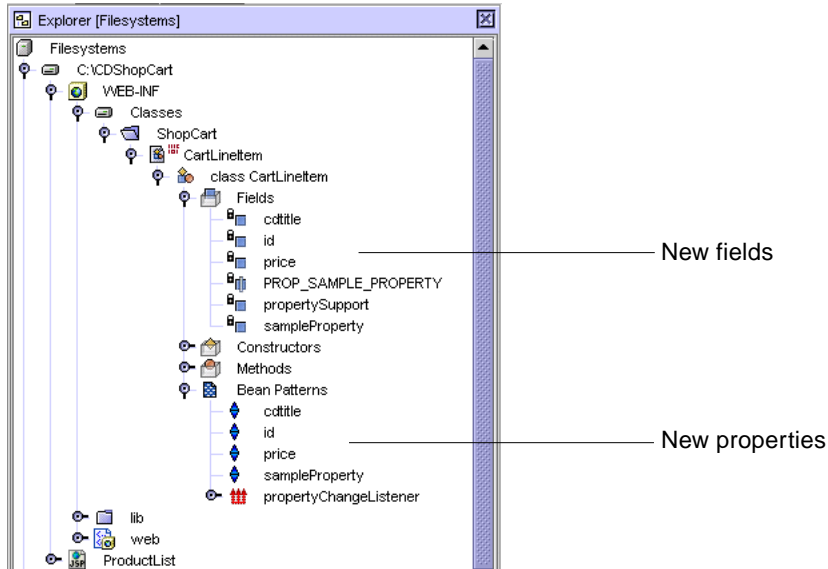
10. Similarly, create the id property as follows:

Property	Value
Name	id
Type	int
Options enabled	Generate field Generate return statement Generate set statement

11. And also create the price as follows:

Property	Value
Name	price
Type	double
Options enabled	Generate field Generate return statement Generate set statement

12. Expand the Fields node (of the CartLineItem bean class) to see the new fields you created.



13. Double-click one of the new fields you just created (for example, cdtitle) to see the code for it in the Source Editor.
14. Expand the Methods node to see the new get and set methods for each field.

Converting the id Property and price Property to Strings

The parameters passed from the ProductList JSP page are all passed as strings. However, because neither the id property nor price property are strings, you must convert them. An efficient way to do this is to overload the properties' setter methods and add the proper code.

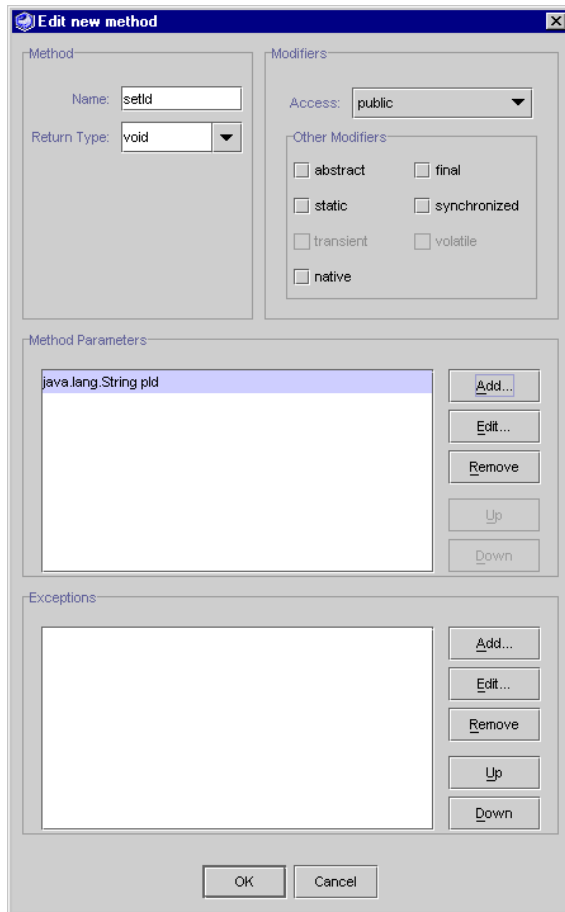
To overload the setId and setPrice methods:

1. Right-click the Methods node (of the CartLineItem bean), and choose Add Method.
2. In the Edit New Method dialog box that appears, define the setId method.
 - a. Type setId in the Name field.
 - b. Select void for Return Type.
3. In the Method Parameters box, click the Add button to display the Enter Method Parameter dialog box.
4. Define the pId parameter.
 - a. Select java.lang.String for the Type.

b. Type pId for the Name.

5. Click OK.

The New Method dialog box looks like this:



6. Click OK to create the method and close the dialog box.

7. Double-click the new method in the Explorer to display its code in the Source editor.

8. Add code to this new method, as follows:

```
public void setId(java.lang.String pId) {  
    int val = Integer.parseInt(pId);  
    this.setId(val);  
}
```

Tip – Whenever you enter code into the Source Editor by copying and pasting, you can automatically reformat it properly by putting the cursor in the Source Editor and typing Ctrl-Shift-F.

Now create the `setPrice` method in a similar manner.

9. Define the `setPrice` method as follows:

Property	Value
Name	<code>setPrice</code>
Return Type	<code>void</code>
Parameter Type	<code>java.lang.String</code>
Parameter Name	<code>pPrice</code>

10. Add the following code to this new method:

```
public void setPrice(java.lang.String pPrice) {  
    double val = Double.parseDouble(pPrice);  
    this.setPrice(val);  
}
```

11. Select the `CartLineItem` bean (🔍), not the class (📁) and choose **Build → **Compile** (or press F9)**

If the bean compiles without errors, the red “need to compile” badge is removed from the bean’s node and you are ready to create the `Cart` bean. If not, check your typing and recompile.

Tip – The completed source code for this bean is found at “`CartLineItem` Bean Source” on page 84.

Creating the `CartLineItemBeanInfo` Component

It is a standard Java convention, built in to the `Introspector` class, that a bean property is a private attribute that is accessible by a pair of public accessor (`get` and `set`) methods. Because you have just overloaded the `set` methods for the `id` and `price` properties, the `Introspector` will no longer recognize these as bean properties.

However, you do want these fields to be recognized as properties, because the JSTL expression language will only make bean properties available if they follow Java conventions. For example, item that comes after the “.” operator in the following code must be a property:

```
<TD><c:out value="\${row.id}"/></TD>
<TD><c:out value="\${row.cdtitle}"/></TD>
```

You can work around this difficulty with a BeanInfo component, with which you can specify the properties of a bean directly. Create a BeanInfo component for the CartLineItem bean, then code it to specify that id and price are properties.

To specify that id and price are properties of the CartLineItem bean:

- 1. Right-click the ShopCart package and choose New → All Templates.**

The New wizard is displayed.

- 2. Expand the Java Beans node, select BeanInfo w/o Icon, and click Next.**

- 3. Name the BeanInfo CartLineItemBeanInfo and click Finish.**

The new CartLineItemBeanInfo node is displayed in the Explorer.

- 4. Expand the BeanInfo node and its Methods node.**

- 5. Double-click the getPdescriptor method.**

The CartLineItemBeanInfo bean code is displayed in the Source Editor at the definition of the getPdescriptor method.

- 6. Add the following code to the body of the getPdescriptor method:**

```
if (properties == null) {
    try {
        PropertyDescriptor[] props = {
            new PropertyDescriptor("cdtitle",
                CartLineItem.class),
            new PropertyDescriptor("id", CartLineItem.class),
            new PropertyDescriptor("price",
                CartLineItem.class)};
        properties = props;
    } catch (IntrospectionException ex) {
        return null;
    }
}
```

Tip – Remember to reformat code you paste or type into the Source Editor by pressing Control-Shift F.

7. Select the `CartLineItemBeanInfo` node and press F9 to compile the bean.

The `BeanInfo` bean should compile without errors.

Creating the Cart Bean

The `ShopCart` JSP page instantiates (or finds, if it already exists) a `Cart` object to hold the CD line item objects that are passed to it by the `ProductList` JSP page whenever a user clicks the Add button. The cart object is based on a `Cart` bean.

To create the `Cart` bean:

1. Right-click the `ShopCart` package and choose `New` → `Java Bean`.

Notice that this item appears in the contextual menu. This shortcut is created for your convenience whenever you select an item from the `New` menu.

2. Name the bean `Cart` and click `Finish`.

3. Right-click the `Cart` bean's `Bean Patterns` node and choose `Add` → `Property`.

4. With the `New Property Pattern` dialog box, create a new `lineItems` bean pattern, as follows:

Property	Value
Name	<code>lineItems</code>
Type	<code>java.util.Vector</code>
Selected Options	Generate field Generate return statement Generate set statement

The `java.util.Vector` type is not in the drop list, so just type it in.

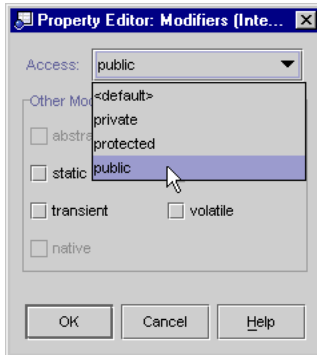
You must now change the access of the `lineItems` field from `private` to `public`.

5. Expand the `Fields` node of the `Cart` class and select the `lineItems` field.

6. Display the `lineItems` properties and click the `Modifiers` value field.

Note – If the Properties window is displayed under the Explorer window, simply selecting an item displays its properties in the window. Alternatively, you can right-click lineItems and choose Properties from the contextual menu.

7. Click the ellipsis (...) button to display the Modifiers dialog box.
8. Choose Public from the Access list.



9. Click OK to accept the change.

Now add code that instantiates a line item object, a method that returns the element number of a selected item, and another method that removes a line item from the cart.

10. Open the Constructors node of the Cart bean and double-click the Cart() constructor.

This action displays the Cart() constructor in the Source Editor.

11. Add the following (bold) code to the Cart bean's constructor to instantiate a new lineItems object, as follows:

```
public Cart() {  
    propertySupport = new PropertyChangeSupport ( this );  
    lineItems = new java.util.Vector();  
}
```

12. Right-click the Cart Methods node, choose Add Method, and define the findLineItem method as follows:

Property	Value
Name	findLineItem
Return Type	int
Parameter Type	int
Parameter Name	pID

13. In the Source Editor, add the following (bold) code to the findLineItem method:



```
public int findLineItem(int pID) {
    System.out.println("Entering Cart.findLineItem()");
    //Return the element number of the item in the cartItems
    //as specified by the passed ID.
    int cartSize = (lineItems == null) ? 0 : lineItems.size();
    int i;
    for (i = 0; i < cartSize; i++ ) {
        if ( pID ==
            ((CartLineItem)lineItems.elementAt(i)).getId() )
            break;
    }
    if (i >= cartSize) {
        System.out.println("Couldn't find line item for ID: " +
            pID);
        return -1;
    }
    else
        return i;
}
```

14. Repeat Step 12, defining the removeLineItem method as follows:

Property	Value
Name	removeLineItem
Return Type	void
Parameter Type	int
Parameter Name	pID

15. Add the following code in the Source Editor to the removeLineItem method:

```
public void removeLineItem(int pID) {
    System.out.println("Entering cart.removeLineItem()");
    int i = findLineItem(pID);
    if (i != -1) lineItems.remove(i);
    System.out.println("Leaving cart.removeLineItem()");
}
```

16. Select the Cart bean (), not the class (), and click F9 to compile the Cart bean.

The bean should compile without errors. Now create the ShopCart JSP page.

Tip – The completed source code of this bean is found at “Cart Bean Source” on page 91.

Creating the Shopping Cart Page

Now create the page that receives the parameters passed from the CD Catalog List page and displays some of them (id, title, and price) as a row in a table. This page also offers mechanisms for deleting an item from the table, returning to the Catalog List page, and placing the order. The title of this page is “Shopping Cart,” and the mechanism that produces it is the ShopCart JSP page.

- 1. Create a JSP page by right-clicking the CDShopCart web module and choosing New → JSP.**
- 2. Name the JSP page ShopCart and click Finish.**

The ShopCart JSP is displayed in the Explorer and in the Source Editor.

Adding Code to Add or Remove an Item From the Shopping Cart Table

Now add code that creates the cart items table. To instantiate a Cart object and a CartLineItem object, use a directive to import the Cart bean, the java.util library (the CartLineItem is a type Vector, from this library), and CartLineItem. Use the same core tags that you used in the ProductList JSP page, so also add a directive to import these tags.

Use scriptlet code to create the cart. Then add code that adds a line item created with the parameters passed from the ProductList JSP page. Add more code that deletes a line item when the user presses the Delete button. Add code to forward the action to a JSP page you will create later (the EmptyCart JSP page) if the table is empty.

As with the ProductList JSP page, use iteration tags to organize the table data. Then use a form to create the table and add a Delete button to each row.

1. Add a Page directive to import the java.util library and the ShopCart package:

```
<%@page contentType="text/html" %>
<%@page import="java.util.*, ShopCart.*" %>
```

2. Change the page title to Shopping Cart and add the directive to import the core.jar library:

```
<head><title>Shopping Cart</title></head>
<%@taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
```

3. Below the <body> tag, create a Shopping Cart heading for the page:

```
<body>
<h1> Shopping Cart </h1>
```

4. Below the heading, use the usebean tag to tell the JSP page to use the Cart bean:

```
<jsp:useBean id="myCart" scope="session" class="ShopCart.Cart" />
```

This code instantiates a Cart object and places it on the session.

Now specify what happens when the current operation for the session is Add. This happens when the user clicks the Add button on the ProductList page. Add code that gets the cdId, cdTitle, and cdPrice objects and adds them to the myCart object.

5. Begin the code by getting the current operation and defining what happens when a user clicks Add:

```
<%
String myOperation = request.getParameter("operation");
session.setAttribute("myLineItems", myCart.getLineItems());

if (myOperation.equals("Add")) {
CartLineItem lineItem = new CartLineItem();
lineItem.setId(request.getParameter("cdId"));
lineItem.setCdtitle(request.getParameter("cdTitle"));
lineItem.setPrice(request.getParameter("cdPrice"));
myCart.lineItems.addElement(lineItem);
}
}
```

Next, specify what happens when the current operation for the session is Delete. This happens when the user clicks the Delete button on the Shopping Cart page.

6. Use the following code to specify what happens when the user clicks Delete:

```
if (myOperation.equals("Delete")) {
String s = request.getParameter("cdId");
System.out.println(s);
int idVal = Integer.parseInt(s);
myCart.removeLineItem(idVal);
}
}
```

Finally, specify what happens when the Delete action deletes the last row of the Cart CD table. Use the JSP forward tag to go to the EmptyCart JSP page, which you will create soon. This last code ends the script you started in Step 5. You have to make a break for the forward tag, then resume before you finally end the scriptlet.

7. Use the following code:

```
//If the last item is removed from the cart...
if (((Vector)session.getAttribute("myLineItems")).size() == 0) {
//End scriptlet temporarily so that you can use the JSP
//"forward" tag to forward to the EmptyCart page.
%>
<jsp:forward page="EmptyCart.jsp" />
//Resume the scriptlet.
<%
}
%>
```

Using Iteration Tags to Populate the Cart Table

Next, use the `forEach` and `out` tags to iterate through the passed data and fetch the field data of each row, much as you did in “Using the Iteration Tags to Display the Data” on page 54.

1. Start a table for the purchase candidate data by creating the table headings:

```
<TABLE border=1>
<TR>
<TH>ID</TH>
<TH>CD Title</TH>
<TH>Price</TH>
</TR>
```

2. Use the `forEach` and `out` tags to populate the table:

```
<c:forEach var="item" items="${myLineItems}">
<TR>
<TD><c:out value="${item.id}"/></TD>
<TD><c:out value="${item.cdtitle}"/></TD>
<TD><c:out value="${item.price}"/></TD>
```

The `out` tags in the preceding code retrieve the value from the named fields in the current row of the results.

3. Create a Delete button for each row, as in “Adding the Buttons to the Page” on page 72:

```
<TD>
<form method=get action="ShopCart.jsp">
<input type=hidden name=cdId value="<c:out value="${item.id}"/>">
<input type=hidden name=cdTitle value="<c:out value=
"${item.cdtitle}"/>">
<input type=hidden name=cdPrice value="<c:out value=
"${item.price}"/>">
<input type=submit name=operation value="Delete">
</form>
</TD>
</TR>
</c:forEach>
</TABLE>
```

Adding the Buttons to the Page

Finally, add the Resume Shopping, Place Order, and Cancel Order buttons to the page bottom.

- **Add the following code to the ShopCart JSP page:**

```
<p>
<!--Create the three buttons.-->
<form method=get action="ProductList.jsp">
<input type=submit value="Resume Shopping">
</form>
<form method=get action="PlaceOrder.jsp">
<input type=submit value="Place Order">
</form>
<form method=get action="CancelOrder.jsp">
<input type=submit value="Cancel Order">
</form>
</p>
<!End the page.>
</body>
</html>
```

Tip – To see the completed source code of this page, see “ShopCart.jsp Source” on page 94.

Testing the Shopping Cart Page

You do not test the ShopCart page directly. You test the ProductList page and navigate (by means of the Add button) to the Shopping Cart page. First, you have to redeploy the CDShopCart web application, because it contains components that were not in the previously deployed version.

Note – Make sure Sun ONE Application Server 7 is the default web server of the IDE. See “Confirming Sun ONE Application Server 7 as the Default Server” on page 27 for information.

1. **If not already started, start the PointBase Network Server by choosing Tools → PointBase Network Server → Start Server.**
2. **Right-click the CDShopCart web module and choose Build All.**

3. Right-click the CDShopCart web module again and choose Deploy.

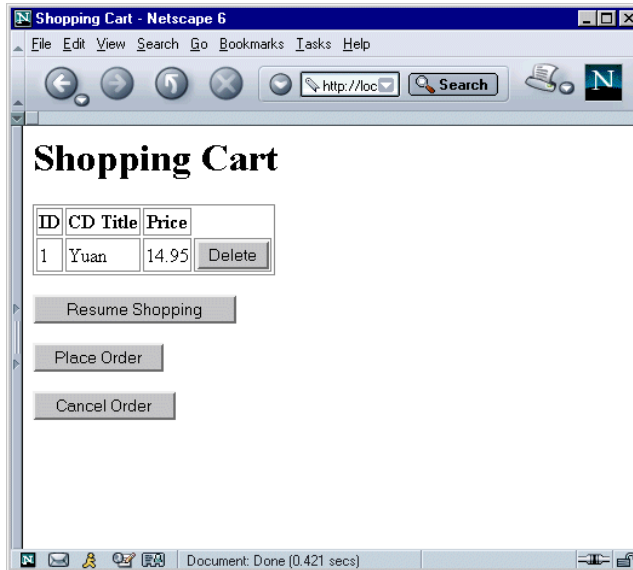
The IDE redeploys the application. A progress window is displayed, showing the progress of the deployment process, then goes away when the process is completed.

4. Right-click the ProductList JSP page and choose Execute.

The IDE launches the default browser and displays the CD Catalog List page.

5. Click one of the Add buttons to navigate to the Shopping Cart page.

The Shopping Cart page looks similar to this example:



6. Use the Resume Shopping button to return to the CD Catalog List page.

You have almost finished the CDShopCart application. You only have to create the EmptyCart and PlaceOrder pages, and you're done.

Creating the Three Message Pages

Now create a JSP page that is displayed when a user empties the cart and two other pages that are displayed when the user clicks the Place Order and Cancel Order buttons.

Creating the Empty Cart Page

When an iterator tag finds an empty vector, it throws an exception rather than creating an empty table. You have dealt with this by testing for this case (Step 7 in “Adding Code to Add or Remove an Item From the Shopping Cart Table” on page 68) and then you handled the exception by displaying the Empty Cart page. This page contains a Resume Shopping button that enables the user to return to the product list page.



FIGURE 3-3 Empty Cart Page

To create the Empty Cart page:

1. **Right-click the CDSShopCart web module and choose New → JSP.**
2. **Type the name `EmptyCart` and click Finish.**

The EmptyCart JSP page appears in the Explorer and its source in the Source Editor.

3. Change the page title to Empty Cart and add code as follows:

```
<%@page contentType="text/html"%>
<html>
<head><title>Empty Cart</title></head>
<body>
<H1> Empty Cart </H1>
<!--Display a message-->
Your shopping cart is empty.
<P>
<!--Add a Resume Shopping button.-->
<form method=get action="ProductList.jsp">
<input type=submit value="Resume Shopping">
</FORM>
</P>
</body>
</html>
```

4. Save the EmptyCart page by choosing File → Save.

Creating the Place Order Page

The Place Order and Cancel Order pages are very simple pages, and represent only one of many ways you can implement these actions. Because programming these pages demonstrates little more of the Sun ONE Studio 5 features than you have already seen, the tutorial uses the simplest possible implementation.

The Place Order page is displayed when the user clicks the Place Order button on the Shopping Cart page. Displaying this page ends the session.



FIGURE 3-4 Place Order Page

To create the Place Order page:

- 1. Right-click the CDShopCart web module and choose New → JSP.**
- 2. Type the name PlaceOrder and click Finish.**

3. Change the page title to Place Order and add code as follows:

```
<%@page contentType="text/html"%>
<html>
<head><title>Place Order</title></head>
<body>
<H1> Place Order </H1>
<!--Invalidate the session-->
<%
session.invalidate();
%>
Your order has been placed. Thank you for shopping.
<P>
<FORM method=get action="ProductList.jsp">
<INPUT type=submit value="Resume Shopping">
</FORM>
</P>
</body>
</html>
```

4. Save the PlaceOrder page by choosing File → Save.

Creating the Cancel Order Page

The Cancel Order page is displayed when the user clicks the Cancel Order button on the Shopping Cart page. Displaying this page ends the session. The page looks like FIGURE 3-5.

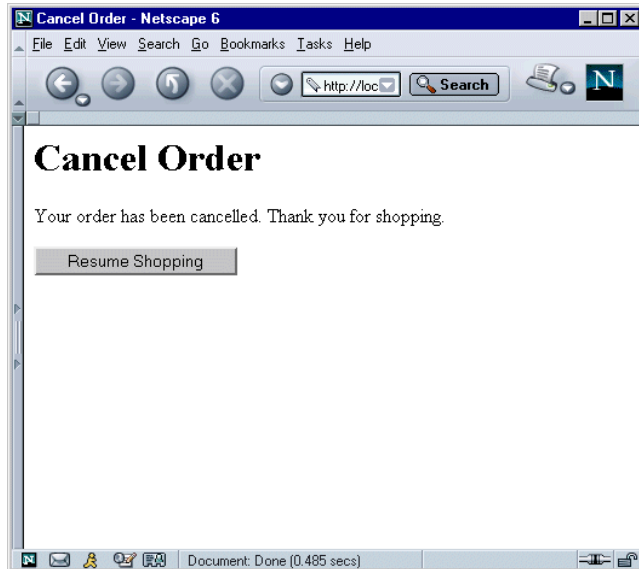


FIGURE 3-5 Cancel Order Page

To create the Cancel Order page:

1. **Right-click the CDShopCart web module and choose New → JSP & Servlet → JSP.**
2. **Type the name CancelOrder and click Finish.**
3. **Change the page title to Cancel Order and add code similar to the PlaceOrder page, as follows:**

```
<%@page contentType="text/html"%>
<html>
<head><title>Cancel Order</title></head>
<body>
<H1> Cancel Order </H1>
<%
session.invalidate();
%>
Your order has been cancelled. Thank you for shopping.
<P>
<FORM method=get action="ProductList.jsp">
<INPUT type=submit value="Resume Shopping">
</FORM>
</P>
</body>
</html>
```

4. Save the CancelOrder page by choosing File → Save.

Testing the Three Message Pages

As with the Shopping Cart page, test the message pages by running the ProductList JSP page. Then add CD items to the Shopping Cart and perform an appropriate action to display each message page. First, you have to redeploy the CDShopCart web application, because it contains components that were not in the previously deployed version.

Note – Make sure Sun ONE Application Server 7 is the default web server of the IDE. See “Confirming Sun ONE Application Server 7 as the Default Server” on page 27 for information.

1. If not already started, start the PointBase Network Server by choosing Tools → PointBase Network Server → Start Server.
2. Right-click the CDShopCart web module and choose Build All.
3. Right-click the CDShopCart web module again and choose Deploy.
The IDE redeploys the application. A progress window is displayed, showing the progress of the deployment process, then goes away when the process is completed.
4. Right-click the ProductList JSP page and choose Execute.
The IDE launches the default browser and displays the CD Catalog List page.
5. Click one of the Add buttons to navigate to the Shopping Cart page.
6. To test the Empty Cart page, click the Delete button on the item you just put into the cart.
The Empty Cart page appears.
7. Click the Resume Shopping button to return to the CD Catalog List page.
8. Add one or more CDs to the cart.
9. Test the Cancel Order page by clicking the Cancel Order button.
10. When the page appears, click the Resume Shopping button to return to the CD Catalog List page.
11. Add another CD to the cart.

- 12. When the Shopping Cart page appears, make sure that the CD you added is the only one in the cart.**

There should be only one CD in the cart because the Cancel Order action (Step 9) ended the previous session.

- 13. Add more CDs to the cart, and then test the Place Order button.**

- 14. When the Place Order page appears, click the Resume Shopping button to return to the Catalog page.**

- 15. Add another CD to the cart.**

The Place Order page ended the session. Therefore only one CD should be in the cart.

- 16. To stop the application, direct the browser to a different URL.**

You have completed the CDShopCart application.

CDShopCart Source Files

This appendix displays the code for the following CDShopCart components:

- “ProductList.jsp Source” on page 82
- “CartItem Bean Source” on page 84
- “CartItemBeanInfo Source” on page 87
- “Cart Bean Source” on page 91
- “ShopCart.jsp Source” on page 94
- “EmptyCart.jsp Source” on page 96
- “PlaceOrder.jsp Source” on page 96
- “CancelOrder.jsp Source” on page 97

This code is also available as source files within the CDShopCart application zip file, which you can download from the Sun ONE Studio 5 Developer Resources portal at:

<http://forte.sun.com/ffj/documentation/tutorialsandexamples.html>

Tip – If you use these files to cut and paste code into the Sun ONE Studio 5 Source Editor, all formatting is lost. To automatically reformat the code, put the cursor in the Source Editor and press Ctrl-Shift-F.

Solaris and Linux users are advised not to copy these files, because the Source Editor does not read the carriage returns at the ends of lines. To view source files, unzip the CDShopCart source zip file and then mount the unzipped directory in the IDE.

ProductList.jsp Source

```
<%@page contentType="text/html"%>
<html>

<head><title>CD Catalog List</title></head>
<%@taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>
<body>
<h1> CD Catalog List </h1>

<sql:setDataSource var="productDS"
url="jdbc:pointbase:server://localhost/cdshopcart"
driver="com.pointbase.jdbc.jdbcUniversalDriver"
user="PBPUBLIC" password="PBPUBLIC" />

<%--<sql:setDataSource var="productDS"
url="jdbc:oracle:thin:@hostname:port#:SID"
driver="oracle.jdbc.driver.OracleDriver"
user="userid" password="password" /> --%>

<%--<sql:setDataSource var="productDS"
url="jdbc:weblogic:mssqlserver4:database@hostname:port#"
driver="weblogic.jdbc.mssqlserver4.Driver"
user="userid" password="password" /> --%>

<sql:query var="productQuery" dataSource="{productDS}" >
SELECT * FROM CD
</sql:query>

<TABLE border=1>
<TR>
<TH>ID</TH>
<TH>CD Title</TH>
<TH>Artist</TH>
<TH>Country</TH>
<TH>Price</TH>
</TR>
<c:forEach var="row" items="{productQuery.rows}" >
```

```
<TR>
<TD><c:out value="\${row.ID}"/></TD>
<TD><c:out value="\${row.CDTITLE}"/></TD>
<TD><c:out value="\${row.ARTIST}"/></TD>
<TD><c:out value="\${row.COUNTRY}"/></TD>
<TD><c:out value="\${row.PRICE}"/></TD>
<TD>
<form method=get action="ShopCart.jsp">
<input type=hidden name=cdId value="\<c:out value="\${row.ID}"/>">
<input type=hidden name=cdTitle value="\<c:out value="\${row.CDTITLE}"/>">
<input type=hidden name=cdPrice value="\<c:out value="\${row.PRICE}"/>">
<input type=submit name=operation value=Add>
</form>
</TD>
</TR>
</c:forEach>
</TABLE>

</body>
</html>
```

CartLineItem Bean Source

```
/*
 * CartLineItem.java
 *
 * Created on April 11, 2003, 2:24 PM
 */

package ShopCart;

import java.beans.*;

public class CartLineItem extends Object implements java.io.Serializable {

    private static final String PROP_SAMPLE_PROPERTY = "SampleProperty";

    private String sampleProperty;

    private PropertyChangeSupport propertySupport;

    /** Holds value of property cdtitle. */
    private String cdtitle;

    /** Holds value of property id. */
    private int id;

    /** Holds value of property price. */
    private double price;

    /** Creates new CartLineItem */
    public CartLineItem() {
        propertySupport = new PropertyChangeSupport( this );
    }

    public String getSampleProperty() {
        return sampleProperty;
    }

    public void setSampleProperty(String value) {
```

```

        String oldValue = sampleProperty;
        sampleProperty = value;
        propertySupport.firePropertyChange(PROP_SAMPLE_PROPERTY, oldValue,
sampleProperty);
    }

    public void addPropertyChangeListener(PropertyChangeListener listener) {
        propertySupport.addPropertyChangeListener(listener);
    }

    public void removePropertyChangeListener(PropertyChangeListener listener) {
        propertySupport.removePropertyChangeListener(listener);
    }

    /** Getter for property cdtitle.
     * @return Value of property cdtitle.
     *
     */
    public String getCdtitle() {
        return this.cdtitle;
    }

    /** Setter for property cdtitle.
     * @param cdtitle New value of property cdtitle.
     *
     */
    public void setCdtitle(String cdtitle) {
        this.cdtitle = cdtitle;
    }

    /** Getter for property id.
     * @return Value of property id.
     *
     */
    public int getId() {
        return this.id;
    }

    /** Setter for property id.
     * @param id New value of property id.

```

```

*
*/
public void setId(int id) {
    this.id = id;
}

/** Getter for property price.
 * @return Value of property price.
 *
 */
public double getPrice() {
    return this.price;
}

/** Setter for property price.
 * @param price New value of property price.
 *
 */
public void setPrice(double price) {
    this.price = price;
}

public void setId(java.lang.String pId) {
    int val = Integer.parseInt(pId);
    this.setId(val);
}

public void setPrice(java.lang.String pPrice) {
    double val = Double.parseDouble(pPrice);
    this.setPrice(val);
}
}

```

CartLineItemBeanInfo Source

```
package ShopCart;

import java.beans.*;

public class CartLineItemBeanInfo extends SimpleBeanInfo {

    // Bean descriptor information will be obtained from
    introspection.//GEN-FIRST:BeanDescriptor
    private static BeanDescriptor beanDescriptor = null;
    private static BeanDescriptor getBdescriptor(){
        //GEN-HEADEREND:BeanDescriptor

        // Here you can add code for customizing the BeanDescriptor.

        return beanDescriptor;    } //GEN-LAST:BeanDescriptor

    // Properties information will be obtained from
    introspection.//GEN-FIRST:Properties
    private static PropertyDescriptor[] properties = null;
    private static PropertyDescriptor[]
    getPdescriptor()//GEN-HEADEREND:Properties

    if (properties == null) {
        try {
            PropertyDescriptor[] props = {
                new PropertyDescriptor("cdtitle",
                    CartLineItem.class),
                new PropertyDescriptor("id", CartLineItem.class),
                new PropertyDescriptor("price",
                    CartLineItem.class)};
            properties = props;
        } catch (IntrospectionException ex) {
            return null;
        }
    }
}
```

```

        return properties;    } //GEN-LAST:Properties

    // Event set information will be obtained from
introspection.//GEN-FIRST:Events
    private static EventSetDescriptor[] eventSets = null;
    private static EventSetDescriptor[] getEdescriptor(){//GEN-HEADEREND:Events

        // Here you can add code for customizing the event sets array.

        return eventSets;    } //GEN-LAST:Events

    // Method information will be obtained from
introspection.//GEN-FIRST:Methods
    private static MethodDescriptor[] methods = null;
    private static MethodDescriptor[] getMdescriptor(){//GEN-HEADEREND:Methods

        // Here you can add code for customizing the methods array.

        return methods;    } //GEN-LAST:Methods

private static int defaultPropertyIndex = -1; //GEN-BEGIN:Idx
private static int defaultEventIndex = -1; //GEN-END:Idx

//GEN-FIRST:Superclass

// Here you can add code for customizing the Superclass BeanInfo.

//GEN-LAST:Superclass

/**
 * Gets the bean's <code>BeanDescriptor</code>s.
 *
 * @return BeanDescriptor describing the editable
 * properties of this bean. May return null if the
 * information should be obtained by automatic analysis.
 */
public BeanDescriptor getBeanDescriptor() {
    return getBdescriptor();
}

```



```

/**
 * Gets the bean's <code>PropertyDescriptor</code>s.
 *
 * @return An array of PropertyDescriptors describing the editable
 * properties supported by this bean. May return null if the
 * information should be obtained by automatic analysis.
 * <p>
 * If a property is indexed, then its entry in the result array will
 * belong to the IndexedPropertyDescriptor subclass of PropertyDescriptor.
 * A client of getPropertyDescriptors can use "instanceof" to check
 * if a given PropertyDescriptor is an IndexedPropertyDescriptor.
 */
public PropertyDescriptor[] getPropertyDescriptors() {
    return getPdescriptor();
}

/**
 * Gets the bean's <code>EventSetDescriptor</code>s.
 *
 * @return An array of EventSetDescriptors describing the kinds of
 * events fired by this bean. May return null if the information
 * should be obtained by automatic analysis.
 */
public EventSetDescriptor[] getEventSetDescriptors() {
    return getEdescriptor();
}

/**
 * Gets the bean's <code>MethodDescriptor</code>s.
 *
 * @return An array of MethodDescriptors describing the methods
 * implemented by this bean. May return null if the information
 * should be obtained by automatic analysis.
 */
public MethodDescriptor[] getMethodDescriptors() {
    return getMdescriptor();
}

/**
 * A bean may have a "default" property that is the property that will

```

```
* mostly commonly be initially chosen for update by human's who are
* customizing the bean.
* @return Index of default property in the PropertyDescriptor array
*     returned by getPropertyDescriptors.
* <P>Returns -1 if there is no default property.
*/
public int getDefaultPropertyIndex() {
    return defaultPropertyIndex;
}

/**
 * A bean may have a "default" event that is the event that will
 * mostly commonly be used by human's when using the bean.
 * @return Index of default event in the EventSetDescriptor array
 *     returned by getEventSetDescriptors.
 * <P>Returns -1 if there is no default event.
 */
public int getDefaultEventIndex() {
    return defaultEventIndex;
}
}
```

Cart Bean Source

```
/*
 * Cart.java
 *
 * Created on April 11, 2003, 2:36 PM
 */

package ShopCart;

import java.beans.*;

public class Cart extends Object implements java.io.Serializable {

    private static final String PROP_SAMPLE_PROPERTY = "SampleProperty";

    private String sampleProperty;

    private PropertyChangeSupport propertySupport;

    /** Holds value of property lineItems. */
    public java.util.Vector lineItems;

    /** Creates new Cart */
    public Cart() {
        propertySupport = new PropertyChangeSupport( this );
        lineItems = new java.util.Vector();
    }

    public String getSampleProperty() {
        return sampleProperty;
    }

    public void setSampleProperty(String value) {
        String oldValue = sampleProperty;
        sampleProperty = value;
        propertySupport.firePropertyChange(PROP_SAMPLE_PROPERTY, oldValue,
sampleProperty);
    }
}
```

```

public void addPropertyChangeListener(PropertyChangeListener listener) {
    propertySupport.addPropertyChangeListener(listener);
}

public void removePropertyChangeListener(PropertyChangeListener listener) {
    propertySupport.removePropertyChangeListener(listener);
}

/** Getter for property lineItems.
 * @return Value of property lineItems.
 *
 */
public java.util.Vector getLineItems() {
    return this.lineItems;
}

/** Setter for property lineItems.
 * @param lineItems New value of property lineItems.
 *
 */
public void setLineItems(java.util.Vector lineItems) {
    this.lineItems = lineItems;
}

public int findLineItem(int pID) {
    System.out.println("Entering Cart.findLineItem()");
    //Return the element number of the item in the cartItems
    //as specified by the passed ID.
    int cartSize = (lineItems == null) ? 0 : lineItems.size();
    int i;
    for (i = 0; i < cartSize; i++ ) {
        if ( pID == ((CartLineItem)lineItems.elementAt(i)).getId() )
            break;
    }
    if (i >= cartSize) {
        System.out.println("Couldn't find line item for ID: " + pID);
        return -1;
    }
    else

```

```
        return i;
    }

    public void removeLineItem(int pID) {
        System.out.println("Entering cart.removeLineItem()");
        int i = findLineItem(pID);
        if (i != -1) lineItems.remove(i);
        System.out.println("Leaving cart.removeLineItem()");
    }
}
```

ShopCart.jsp Source

```
<%@page contentType="text/html"%>
<%@page import="java.util.*, ShopCart.*" %>
<html>
<head><title>Shopping Cart</title></head>
<%@taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<body>
<h1> Shopping Cart </h1>
<jsp:useBean id="myCart" scope="session" class="ShopCart.Cart" />

<%
String myOperation = request.getParameter("operation");
session.setAttribute("myLineItems", myCart.getLineItems());

if (myOperation.equals("Add")) {
CartLineItem lineItem = new CartLineItem();
lineItem.setId(request.getParameter("cdId"));
lineItem.setCdtitle(request.getParameter("cdTitle"));
lineItem.setPrice(request.getParameter("cdPrice"));
myCart.lineItems.addElement(lineItem);
}
if (myOperation.equals("Delete")) {
String s = request.getParameter("cdId");
System.out.println(s);
int idVal = Integer.parseInt(s);
myCart.removeLineItem(idVal);
}
//If the last item is removed from the cart...
if (((Vector)session.getAttribute("myLineItems")).size() == 0) {
//End scriptlet temporarily so that you can use the JSP
//?forward? tag to forward to the EmptyCart page.
%>
<jsp:forward page="EmptyCart.jsp" />
//Resume the scriptlet.
<%
}
%>
```

```

<TABLE border=1>
<TR>
<TH>ID</TH>
<TH>CD Title</TH>
<TH>Price</TH>
</TR>
<c:forEach var="item" items="\${myLineItems}">
<TR>
<TD><c:out value="\${item.id}"/></TD>
<TD><c:out value="\${item.cdtitle}"/></TD>
<TD><c:out value="\${item.price}"/></TD>
<TD>
<form method=get action="ShopCart.jsp">
<input type=hidden name=cdId value="<c:out value="\${item.id}"/>">
<input type=hidden name=cdTitle value="<c:out value="\${item.cdtitle}"/>">
<input type=hidden name=cdPrice value="<c:out value="\${item.price}"/>">
<input type=submit name=operation value="Delete">
</form>
</TD>
</TR>
</c:forEach>
</TABLE>

<p>
<!--Create the three buttons.-->
<form method=get action="ProductList.jsp">
<input type=submit value="Resume Shopping">
</form>
<form method=get action="PlaceOrder.jsp">
<input type=submit value="Place Order">
</form>
<form method=get action="CancelOrder.jsp">
<input type=submit value="Cancel Order">
</form>
</p>
<!End the page.>
</body>
</html>

```

EmptyCart.jsp Source

```
<%@page contentType="text/html"%>
<html>
<head><title>Empty Cart</title></head>
<body>
<H1> Empty Cart </H1>
<!--Display a message-->
Your shopping cart is empty.
<P>
<!--Add a Resume Shopping button.-->
<form method=get action="ProductList.jsp">
<input type=submit value="Resume Shopping">
</FORM>
</P>
</body>
</html>
```

PlaceOrder.jsp Source

```
<%@page contentType="text/html"%>
<html>
<head><title>Place Order</title></head>
<body>
<H1> Place Order </H1>
<!--Invalidate the session-->
<%
session.invalidate();
%>
Your order has been placed. Thank you for shopping.
<P>
<FORM method=get action="ProductList.jsp">
<INPUT type=submit value="Resume Shopping">
</FORM>
```



```
</P>  
</body>  
</html>
```

CancelOrder.jsp Source

```
<%@page contentType="text/html"%>  
<html>  
<head><title>Cancel Order</title></head>  
<body>  
<H1> Cancel Order </H1>  
<%  
session.invalidate();  
%>  
Your order has been cancelled. Thank you for shopping.  
<P>  
<FORM method=get action="ProductList.jsp">  
<INPUT type=submit value="Resume Shopping">  
</FORM>  
</P>  
</body>  
</html>
```


CDSShopCart Database Scripts

This appendix displays the following database scripts for the CDSShopCart tutorial:

- “Script for a PointBase Database” on page 100
- “Script for an Oracle Database” on page 101
- “Script for a Microsoft SQLServer Database” on page 102
- “Script for an IBM DB2 Database” on page 103

These scripts are also available as files within the CDSShopCart application zip file, which you can download from the Sun ONE Studio 5 Developer Resources portal at:

<http://forte.sun.com/ffj/documentation/tutorialsandexamples.html>

Script for a PointBase Database

Use the following SQL script with a PointBase database.

```
drop table CD;

create table CD(
    id      int,
    cdtitle char(20),
    artist  char(20),
    country char(20),
    price  number(8,2),
    primary key(id));

insert into CD (id, cdtitle, artist, country, price)
    values (1, 'Yuan','The Guo Brothers', 'China',14.95);
insert into CD (id, cdtitle, artist, country, price)
    values (2, 'Drums of Passion','Babatunde Olatunji', 'Nigeria',16.95);
insert into CD (id, cdtitle, artist, country, price)
    values (3, 'Kaira','Tounami Diabate', 'Mali',13.95);
insert into CD (id, cdtitle, artist, country, price)
    values (4, 'The Lion is Loose','Eliades Ochoa', 'Cuba',12.95);
insert into CD (id, cdtitle, artist, country, price)
    values (5, 'Dance the Devil Away','Outback', 'Australia',14.95);

commit;
```

Script for an Oracle Database

Use the following SQL script with an Oracle database.

```
/* Run with: sqlplus tutorial/tutorial@dbname @scriptname */
drop table CD;

create table CD(
    id      int,
    cdtitle char(20),
    artist  char(20),
    country char(20),
    price   number(8,2),
    primary key(id));
grant all on CD to public;

insert into CD (id, cdtitle, artist, country, price)
values (1, 'Yuan','The Guo Brothers', 'China',14.95);
insert into CD (id, cdtitle, artist, country, price)
values (2, 'Drums of Passion','Babatunde Olatunji', 'Nigeria',16.95);
insert into CD (id, cdtitle, artist, country, price)
values (3, 'Kaira','Tounami Diabate', 'Mali',13.95);
insert into CD (id, cdtitle, artist, country, price)
values (4, 'The Lion is Loose','Eliades Ochoa', 'Cuba',12.95);
insert into CD (id, cdtitle, artist, country, price)
values (5, 'Dance the Devil Away','Outback', 'Australia',14.95);
commit;
```

Script for a Microsoft SQLServer Database

Use the following SQL script with a Microsoft SQLServer database.

```
use cdcat
go
drop table CD
go

create table CD(
    id      int,
    cdtitle varchar(20) null,
    artist  varchar(20) null,
    country varchar(20) null,
    price   money null,
PRIMARY KEY(id))
go
grant all on CD to public
go

insert into CD (id, cdtitle, artist, country, price)
    values (1, 'Yuan','The Guo Brothers', 'China',14.95)
insert into CD (id, cdtitle, artist, country, price)
    values (2, 'Drums of Passion','Babatunde Olatunji', 'Nigeria',16.95)
insert into CD (id, cdtitle, artist, country, price)
    values (3, 'Kaira','Tounami Diabate', 'Mali',13.95)
insert into CD (id, cdtitle, artist, country, price)
    values (4, 'The Lion is Loose','Eliades Ochoa', 'Cuba',12.95)
insert into CD (id, cdtitle, artist, country, price)
    values (5, 'Dance the Devil Away','Outback', 'Australia',14.95)
go
```

Script for an IBM DB2 Database

Use the following SQL script with an IBM DB2 database.

```
drop table CD

create table CD(
    id      int not null primary key ,
    cdtitle char(20),
    artist  char(20),
    country char(20),
    price   num(8,2))

insert into CD (id, cdtitle, artist, country, price)
    values (1, 'Yuan','The Guo Brothers', 'China',14.95)
insert into CD (id, cdtitle, artist, country, price)
    values (2, 'Drums of Passion','Babatunde Olatunji', 'Nigeria',16.95)
insert into CD (id, cdtitle, artist, country, price)
    values (3, 'Kaira','Tounami Diabate', 'Mali',13.95)
insert into CD (id, cdtitle, artist, country, price)
    values (4, 'The Lion is Loose','Eliades Ochoa', 'Cuba',12.95)
insert into CD (id, cdtitle, artist, country, price)
    values (5, 'Dance the Devil Away','Outback', 'Australia',14.95)
```


Creating the Tutorial with an Oracle Database

This appendix describes the steps you must perform to create and run the CDShopCart tutorial with an Oracle database. The topics covered are:

- “Connecting the IDE to an Oracle Database” on page 106
- “Creating the Database Tables” on page 108
- “Creating the CDShopCart Application with an Oracle Database” on page 110

Note – There are several references in this book to the *CDShopCart application files*. These files include a completed version of the tutorial application, a readme file describing how to run the completed application, and SQL script files for creating the required database tables. These files are compressed into a zip file you can download from the Sun ONE Studio 5 Developer Resources portal at <http://forte.sun.com/ffj/documentation/tutorialsandexamples.html>

Connecting the IDE to an Oracle Database

Configuring the Sun ONE Studio 5 software to connect to the Oracle database requires the following actions:

- Enabling the Oracle JDBC driver
- Connecting the IDE to the Oracle Server

Enabling the Oracle Type 4 JDBC Driver

Enabling a JDBC driver means putting the driver library in the Sun ONE Studio 5 and Sun ONE Application Server 7 class paths. To do this, you need the Oracle Type 4 JDBC driver library (the `classes12.zip` file). You can download this driver from the Oracle portal.

To enable the Oracle Type 4 JDBC driver:

1. **Copy the Oracle Type 4 driver library to the `s1studio-install-directory/lib/ext` directory.**

For example, copy the `classes12.zip` file to `c:\Sun\studio5_se\lib\ext`. Solaris and Linux users will have a different target directory.

Note – You must have root or administrator privileges to write to the Sun ONE Studio 5 home directories.

2. **Restart the IDE.**
3. **In the Runtime pane of the Explorer, select your application server instance.**

It is labeled *app-server-name* (*app-server-host:app-server-port*). For example, the default server is `server1` (`localhost:4848`), or a standard user's server could be `MyServer` (`localhost:4855`).
4. **Display the properties of the application server instance.**

The property window is usually below the Explorer window. Selecting the node displays the properties in the window. Alternatively, right-click the server instance node and choose Properties.
5. **Open the property editor for the Classpath Suffix property.**

Click on the value field of this property, then on the ellipsis button that appears. The Classpath Suffix editor window is displayed.

6. Click the Add JAR/ZIP button.

Use the Add JAR File file finder to locate your `classes12.zip` file.

7. Select the `classes12.zip` file and click OK.

8. Click OK to close the property editor window. J

Connecting the IDE to the Oracle Server

To perform various actions in the tutorial, you must connect the IDE to the Oracle database. You can either connect before creating the components or during the creation process. Here is how you connect to the database beforehand:

1. Make sure the Oracle server is running.

2. In the Runtime pane of the Explorer, expand the Databases node and its Drivers subnode.

A node labeled Oracle thin is displayed.

If this node has a red strike across it, you have not enabled the Oracle JDBC driver properly. Follow the procedures in “Enabling the Oracle Type 4 JDBC Driver” on page 106.

3. Right-click this node and choose Connect Using.

The New Database Connection dialog box is displayed.

4. Make sure Oracle thin is selected in the name field.

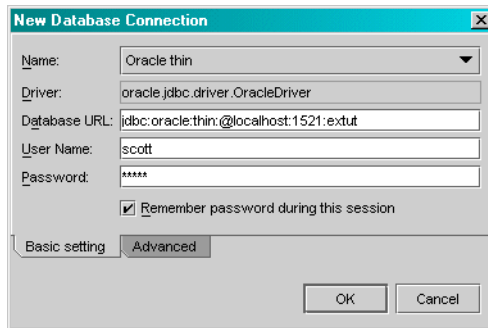
5. Fill in the property values for URL, User, and Password.

For example, the following values are correct for a locally installed Oracle database with a SID of “extut,” and the default Oracle login of “scott” for User and “tiger” for Password: (1521 is the standard Oracle port number.)

Name	Value
URL	<code>jdbc:oracle:thin:@localhost:1521:extut</code>
User	scott
Password	tiger

6. Enable the Remember password during this session option.

The New Database Connection dialog box looks like this:



7. Click OK.

8. Close the Drivers node.

The new Oracle thin driver node is displayed, labeled `jdbc:oracle:thin:@hostname:1521:sid [Username on Password]`.

Creating the Database Tables

The CDShopCart tutorial uses one database table, which you must create in an Oracle Server database. The instructions that follow describe how to use the provided SQL script to create the table. Microsoft Windows users can copy and paste the SQL script provided in Appendix B to create this table. Solaris and Linux users can use a script file, `CDCatalog_ora.sql`, which is available within CDShopCart application files.

To install the tutorial table in an Oracle database on Microsoft Windows systems:

1. Open the Oracle Console by choosing **Start** → **Programs** → **Oracle (your version)** → **Application Development** → **SQL Plus**.
2. Log in to SQL Plus using your user name and password.
3. When the SQL prompt appears, copy the script from Appendix B and paste it next to the prompt.

Tip – The initial DROP statement, which refers to a table that has not yet been created, will create a harmless error. This DROP statement will be useful in future, however, if you want to rerun the script to initialize the table.

To install the tutorial table on Solaris or Linux environments:

1. **Unzip the `CDSshopCart.zip` file from the Developer Resources portal.**

For example, unzip it to the `/MyZipFiles` directory.

2. **At a command prompt, type:**

```
$ cd your-unzip-dir/CDSshopCart/db
$ sqlplus db-userid/db-password@db-servicename @CDCatalog_ora.sql
```

For example,

```
$ cd /MyZipFiles/CDSshopCart/db
$ sqlplus scott/tiger@MyDB @CDCatalog_ora.sql
```

The DROP statement will generate errors, but they are harmless.

Viewing the Database Tables in the IDE

Before performing this procedure, connect the IDE to the database, as in “Connecting the IDE to the Oracle Server” on page 107.

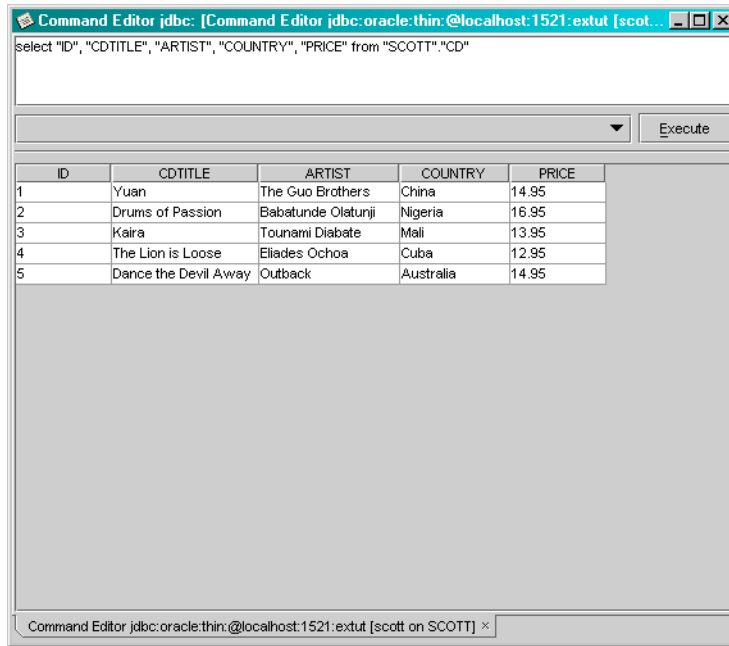
To test that you have created the CD table in the database that you have connected to the IDE:

1. **In the Runtime tab of the Explorer, expand the Databases node, the Oracle connection node, and its Tables node.**

If you don't see the CD table, right-click the Tables node and choose Refresh from the contextual menu.

2. **Right-click the CD table and choose View Data from the contextual menu.**

The Command Editor is displayed, showing the table's data.



Creating the CDShopCart Application with an Oracle Database

There is only one change you have to make to Chapter 3 to create the application using an Oracle database:

In the section, "Using the `setDataSource` Tag to Connect to the Database" on page 53, in Step 2, use the `setDataSource` statement for Oracle instead of PointBase. Fill in the variables with the correct values for your connection. For example, for an Oracle database on a local system, with a database named "MyOracleDB," a userid of "scott." and password of "tiger":

```
<sql:setDataSource var="productDS"
url="jdbc:oracle:thin:@localhost:1521:MyOracleDB"
driver="oracle.jdbc.driver.OracleDriver"
user="scott" password="tiger" />
```

Otherwise, proceed with all other instructions as described.

Index

A

- Add Admin Server menu item, 23
- Add JSP TagLibrary menu item, 49
- Add Method menu item, 61, 67
- Add Property menu item, 59

B

- bean properties, adding, 59
- BeanInfo w/o Icon menu item, 64

C

- CancelOrder JSP page
 - creating, 77 to 79
 - description, 39
 - displaying, 78
 - source code, 97
- Cart bean
 - adding the `findLineItem` method, 67
 - adding the `lineItems` property, 65
 - adding the `removeLineItem` method, 67
 - coding the constructor, 66
 - creating, 65 to 68
 - description, 39
 - source code, 91
- CartLineItem bean
 - creating, 59 to 63
 - description, 39
 - `setId` and `setPrice` overloading, 61
 - source code, 84

- CartLineItemBeanInfo component
 - creating, 63 to 65
 - description, 39
 - source code, 87

- CD table, 30

- CDSShopCart application
 - application scenarios, 32
 - architecture, 38
 - functional description, 31
 - functional specs, 33
 - zipped source files, 19, 105

- CDSShopCart application pages
 - Cancel Order, 78
 - CD Catalog List, 51
 - Empty Cart, 74
 - Place Order, 76
 - Shopping Cart, 58

- CDSShopCart web module
 - creating, 44 to 46
 - setting the context root, 46

- class path, relation to mounted Filesystems, 50

- Compile menu item, 63
- context root property, 46

- core JSP tags
 - `forEach`, 54, 71
 - `out`, 54, 71
 - `using`, 54

- `core.tld` file
 - description, 48
 - importing, 53

- Create a Server Instance menu item, 24

- D**
- database JSP tags
 - query, 49, 54
 - setDataSource, 53, 110
 - databases
 - creating the tutorial tables (Oracle), 108 to 109
 - creating the tutorial tables (PointBase), 28 to 29
 - installing the Oracle JDBC driver, 106
 - setting up connectivity (Oracle), 106 to 108
 - supported versions, 20
 - viewing data in the IDE, 109
- E**
- EmptyCart JSP page
 - creating, 74
 - description, 38
 - source code, 96
 - example applications, where to download, 16
- F**
- findLineItem method, creating, 67
 - forEach JSP tag, 54, 71
- H**
- HTML pages
 - creating, 74
 - viewing source, 74
- I**
- IBM DB2 software, tutorial database script source, 103
 - installing the database table, 27 to 29
- J**
- JavaBeans components
 - adding a method, 61, 67
 - adding a property, 65
 - Javadoc technology, using in the IDE, 16
- JDBC drivers, enabling (Oracle), 106 to 107
 - JSP code, types, 47
 - JSP pages
 - creating, 68
 - testing, 56 to 57
 - JSP tag libraries
 - description, 47
 - JSTL, *See* JSTL tag library
 - JSTL tag library
 - See also* core JSP tags, database JSP tags and properties, 64
 - definition of JAR files, 48
 - examples, 47
 - importing in a JSP page, 48
 - importing into a web module, 49
 - jstl.jar file, 48
 - online information, 47
 - standard.jar file, 48
 - using var, 49
 - jstl.jar file, 48
- L**
- lineItems property, creating, 65
- M**
- methods, creating, 61, 67
 - Microsoft SQLServer software
 - jdbc connect string, 53
 - tutorial database script source, 102
- N**
- Netscape browser, supported version, 20
 - New Java Bean menu item, 59
 - New Java Package menu item, 59
 - New JSP menu item, 68
 - New wizard, opening, 44

O

Oracle database

See also databases

- connecting to the IDE, 107 to 108
- installing a type 4 JDBC driver, 106
- JDBC connect string, 53, 110
- tutorial database script source, 101
- viewing data in the IDE, 109

out JSP tag, 54, 71

P

PlaceOrder JSP page

- creating, 75
- description, 39
- display before TP, 76
- source code, 96

PointBase database

- installing a database table, 27 to 29
- JDBC connect string, 53
- supported version, 20
- tutorial database script source, 100

ProductList JSP page

- connecting to Oracle, 110
- creating, 51 to 56
- description, 38
- displaying in a browser, 51
- source code, 82
- testing, 56 to 57
- user view, 33

Q

query database JSP tag, 54

R

reformatting code, 63

removeLineItem method, creating, 67

runide.sh script, 21

S

setDataSource database JSP tag

- Oracle setting, 110
- PointBase setting, 53
- syntax, 49

setId method, overloading, 61

setPrice method, overloading, 63

ShopCart JSP page

- coding the body, 68
- creating, 68 to 72
- creating the buttons, 72
- description, 38
- displaying in a browser, 58
- source code, 94
- test executing, 72

ShopCart package, creating, 59

sql.tld file

- description, 48
- importing, 53

standard.jar file, 48

Sun ONE Application Server 7

- confirming as the default server, 27
- starting the admin server (standard user), 23 to 25
- starting the admin server (superuser), 22
- starting the application server, 26

Sun ONE Studio 5 IDE

- class path, 50
- connecting to the Oracle server, 107 to 108
- setting up database connectivity (Oracle), 106 to 108
- starting the IDE, 21

Sun ONE Studio 5, Standard Edition, where to obtain, 20

T

taglib directive, using, 48, 53

V

var, in JSTL tag syntax, 49

W

WAR files, definition, 44

web applications, 37

web applications, *See* web modules

web browsers, supported versions, 20

web components, 37

web modules

- creating, 43

- deploying, 57

- description, 40

- directory hierarchy, 44

- executing, 57

- performing a Build All, 57

- screenshot of parts, 46

- where to find more information, 37

web servers

- confirming the default server, 27

- supported version, 20

web.xml file, description, 44

WEB-INF directory, 44