

IBM WebSphere Commerce



NewFashion Sample Store: Online Help Files

Version 5A

IBM WebSphere Commerce



NewFashion Sample Store: Online Help Files

Version 5A

Note!

Before using this information and the product it supports, be sure to read the general information in the Notices section.

Contents

Chapter 1. Creating a store using a sample	1
Creating a store using a sample	1
Creating a sample store archive	1
Changing store database assets	1
Changing Web assets	10
Publishing a sample store archive from Store Services	11
Publishing a store archive using the command line (Windows)	12
Setting up Payment Manager for your store	14
Creating scheduled jobs for a sample store	15
Configuring e-mail notification for a sample store..	17
Chapter 2. Sample store database assets	19
Store database assets	19
NewFashion Catalog database assets	20
NewFashion contract and business policy database assets	21
NewFashion Shipping database assets	22
Sample store access control database assets	24
Chapter 3. NewFashion sample store	27
NewFashion sample store	27
Chapter 4. NewFashion store pages	29
NewFashion Sample store search pages	29
Sample store address book pages	30
NewFashion Sample store address book pages	32
WebFashion and NewFashion Sample store bundle display pages	34
Sample Store catalog group pages	35
Sample store pages: common implementation techniques	36
Sample store contact us page	39
Sample store footer	39
Sample store header	40
Sample store help page	41
Sample store home page	42
Sample store login pages	42
Sample store privacy page	45
Sample store product pages	45
Sample store registration page	47
Sample store new arrivals page	49
NewFashion Sample store order confirmation page	50
WebFashion and NewFashion E-mail notification page	50
Sample store order summary page	51
NewFashion Sample store check product availability page	52
WebFashion and NewFashion Sample store package display pages	53
Sample store left navigation frame	54
NewFashion Sample store left navigation frame	55
Sample store select billing address page	57
NewFashion Sample store select billing address page	58
Sample store shopping cart	59
NewFashion Sample store shopping cart	61
Sample store shipping method page	62
NewFashion Sample store select shipping method page	63
Sample store select shipping address page	64
NewFashion Sample store select shipping address page	65

Sample store quick checkout summary page	66
WebFashion and NewFashion sample store view orders pages	67
WebFashion and NewFashion Sample store wish list pages	69
Sample store use cases	71
Chapter 5. Sample store use cases	73
Add new address use case (Business Edition)	73
WebFashion and NewFashion add item to the wish list use case	74
View product category use case	74
Checkout shopping cart use case	75
WebFashion and NewFashion display bundle page use case	78
WebFashion and NewFashion Display package page use case	78
Edit an address use case (Business Edition)	79
Home page use case	80
Logon use case	81
Manage Personal Account use case	82
WebFashion and NewFashion View orders use case	83
Display product page use case	83
Change personal information use case	84
Registration use case	85
Display shopping cart use case	86
WebFashion and NewFashion View wish list use case	87
Chapter 6. NewFashion shopping flows	89
NewFashion Sample store shopping cart flow	89
NewFashion Sample store shopping flow	89
Notices	91

Chapter 1. Creating a store using a sample

Creating a store using a sample

To create a store using one of the sample stores, do the following:

1. (Optional)

Business

Create an organization to act as the seller.

To determine if you want to create a new organization to act as the seller, see [Shared data assets](#).

2. Create users for the following roles:

- Site Administrator (if you are not using the default Site Administrator)
- Seller Administrator
- Store Administrator
- Store Developer

Important: In order to publish a store archive you must have Site Administrator, Store Administrator, or Store Developer of all stores access.

If you are creating a user with Store Administrator authority, ensure that the access group applies to all stores.

3. Create a store archive using Store Services
4. Change store database assets.
5. Change Web assets.
6. Publish a store archive.
7. (Optional) Configuring stores.
8. Set up Payment Manager for your store.

Important:

1. In some instances, the sample stores require some set up before all features will function properly. If you are basing your store on a sample store, you may need to complete the set up. For more information, see [Setting up a sample store](#).
2. Certain store data assets, like catalogs and fulfillment centers, are shared between stores. As a result, if you publish multiple stores based on the same sample store, the catalog and fulfillment center will be the same for each store. If you have made changes in one catalog, publishing another store on the same sample will overwrite these changes. For more information and how to avoid overwriting any changes, see [Shared data assets](#).

Creating a sample store archive

To create a store archive that can be used as a sample with Store Services, see the *IBM WebSphere Store Developer's Guide*.

Changing store database assets

If you create your store archive using the tools in Store Services, your new store archive will initially contain the same store database assets as the sample store archive on which you based it, for example, `infashion.sar`. In a store archive, the store database assets take the form of XML files.

In most cases, to change the store database assets, you must edit the XML files directly. In some cases, you can use the tools in Store Services to edit the database assets.

You also have the option of editing database assets directly, that is, once you have published your store archive to the commerce server, you can edit the database using the WebSphere Commerce Accelerator, the Loader package, or directly through SQL inserts. If you choose to edit the database rather than the assets in the store archive, you must either update your store archive to match your changes in the database, or discontinue using the store archive.

Your options are listed in the following table.

Important:

1. The tools in Store Services search for the asset names listed in the table below. As a result, if you want to edit your store archive using the tools in Store Services, you must use the same assets names in your store archive.
2. When a store archive is published to the WebSphere Commerce Server, the database information is loaded in the order specified in the Assets column below. As a result, The order of your assets, as specified in the sarinfo.xml file, should match the order of the assets specified below.
3. The order of the database information in each XML file does not necessarily have to match the order specified in the Database Tables column below. However information for a parent table must precede information for a child table.
4. The information marked optional is not required to create a functioning store.
5. For the Database Edit Options column, note that unless specified otherwise all database assets can be edited using SQL inserts or the Loader package. As a result, the column reflects which assets can be edited by the WebSphere Commerce Accelerator.

Store Database Assets	Database Tables	Store Archive Edit Options	Database Edit Options
fulfillment	FulfillmentCENTER (0...1)	<ul style="list-style-type: none"> • Edit store archive XML files. 	<ul style="list-style-type: none"> • New Fulfillment Center and Changing Fulfillment Center pages in WebSphere Commerce Accelerator. • See Creating a fulfillment center and Changing a fulfillment center

Store	Database Tables Database Assets	Store Archive Edit Options	Database Edit Options
store	STOREENT	<ul style="list-style-type: none"> • Edit store archive XML files. • Changing store information using the Store Profile notebook. Store Profile notebook edits the following database columns: <ul style="list-style-type: none"> – IDENTIFIER – DIRECTORY – SETCURR 	
	STADDRESS	<ul style="list-style-type: none"> • Changing store information using the Store Profile notebook. 	<ul style="list-style-type: none"> • New Fulfillment Center and Changing Fulfillment Center pages in WebSphere Commerce Accelerator. • See Creating a fulfillment center and Changing a fulfillment center
	STOREENTDS	<ul style="list-style-type: none"> • Changing store information using the Store Profile notebook. Store Profile notebook edits the following database columns: <ul style="list-style-type: none"> – DESCRIPTION – DISPLAYNAME 	
	STORELANG	<ul style="list-style-type: none"> • Changing store information using the Store Profile notebook. Store Profile notebook edits the following database columns: <ul style="list-style-type: none"> – LANGUAGE_ID 	
	STORELANGDS	<ul style="list-style-type: none"> • Edit store archive XML files. 	
	STORE	<ul style="list-style-type: none"> • Changing store information using the Store Profile notebook. Store Profile notebook edits the following database columns: <ul style="list-style-type: none"> – PHONE1 – CITY – STORE_ID – COUNTRY – STATE – EMAIL1 – ADDRESS1 – ADDRESS2 – FAX – ZIPCODE 	
	DISPENTREL (default CATENTRY template, CATENTRY_ID = -1)	<ul style="list-style-type: none"> • Changing store information using the Store Profile notebook. Store Profile notebook edits the PAGENAME database column. 	
	DISPCGPREL (default CATEGORY template, CATGROUP_ID = -1)	<ul style="list-style-type: none"> • Changing store information using the Store Profile notebook. Store Profile notebook edits the PAGENAME database column. 	
	VENDOR	<ul style="list-style-type: none"> • Edit store archive XML files. 	

Store Database Assets	Database Tables	Store Archive Edit Options	Database Edit Options
catalog	CATGROUP	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	CATGRPATTR	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	
	CATGRPDESC	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	CATALOG	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	
	CATALOGDSC	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	
	CATTOGRP	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	
	CATGRPREL	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	CATENTRY	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	CATENTDESC	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	ATTRIBUTE	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	ATTRVALUE	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	CATGPENREL	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	CATENTREL	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	
	BASEITEM	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	ITEMSPC	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	VERSIONSPC	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	
	DISTARRANG	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	
4	RECEIPT New Fashion Sample Store: On-line Catalog	<ul style="list-style-type: none"> Changing catalog information Edit store archive XML files. 	
	RCPTAVAIL	<ul style="list-style-type: none"> Changing catalog information 	

	Store Database Tables Database Assets	Store Archive Edit Options	Database Edit Options
tax	JURSTGROUP (common between tax and shipping)	<ul style="list-style-type: none"> • Edit store archive XML files. • Changing tax settings using the Tax notebook. Tax notebook edits the following database columns: <ul style="list-style-type: none"> – JURSTGROUP_ID – DESCRIPTION – SUBCLASS – STOREENT_ID – CODE 	
	JURST (common between tax and shipping)	<ul style="list-style-type: none"> • Changing tax settings using the Tax notebook. Tax notebook edits the following database columns: <ul style="list-style-type: none"> – JURST_ID – COUNTRY – STOREENT_ID – CODE – SUBCLASS – STATE 	
	JURSTGPREL (common between tax and shipping)	<ul style="list-style-type: none"> • Changing tax settings using the Tax notebook. Tax notebook edits the following database columns: <ul style="list-style-type: none"> – JURST_ID – JURSTGROUP_ID – SUBCLASS 	
	CALMETHOD (common between tax and shipping)	<ul style="list-style-type: none"> • Edit store archive XML files. 	
	TXCDCLASS (optional, only for categorizing CALCODE, used by the WebSphere Commerce Accelerator)	<ul style="list-style-type: none"> • Edit store archive XML files. 	
	TAXCGRY	<ul style="list-style-type: none"> • Changing tax settings using the Tax notebook. Tax notebook edits the following database columns: <ul style="list-style-type: none"> – TAXCGRY_ID – STOREENT_ID – NAME 	
	CALCODE (common between tax and shipping)	<ul style="list-style-type: none"> • Changing tax settings using the Tax notebook. Tax notebook edits the following database columns: <ul style="list-style-type: none"> – CALCODE_ID – CODE – CALUSAGE_ID – STOREENT_ID – GROUPBY – CALMETHOD_id – CALMETHOD_id_app – CALMETHOD_id_qfy 	
	CALCODEDSC (common between tax and shipping, optional)	<ul style="list-style-type: none"> • Edit store archive XML files. 	

Store	Database Tables Database Assets	Store Archive Edit Options	Database Edit Options
taxfulfillment	TAXRULE	<ul style="list-style-type: none"> • Edit store archive XML files. • Changing tax settings using the Tax notebook. Tax notebook edits the following database columns: <ul style="list-style-type: none"> – CALRULE_ID – FFMCENTER_ID – JURSTGROUP_ID 	
storecatalog	CATALCD (optional)	<ul style="list-style-type: none"> • Edit store archive XML files. 	<ul style="list-style-type: none"> • Product Management tools in WebSphere Commerce Accelerator
	CATGPCALCD (optional)	<ul style="list-style-type: none"> • Edit store archive XML files. 	

Store Database Assets	Database Tables	Store Archive Edit Options	Database Edit Options
shipping	JURSTGROUP (common between tax and shipping)	<ul style="list-style-type: none"> Edit store archive XML files. Changing shipping settings using the Shipping notebook. Shipping notebook edits the following database columns: <ul style="list-style-type: none"> JURSTGROUP_ID DESCRIPTION SUBCLASS STOREENT_ID CODE 	
	JURST (common between tax and shipping)	<ul style="list-style-type: none"> Changing shipping settings using the Shipping notebook. Shipping notebook edits the following database columns: <ul style="list-style-type: none"> JURST_ID COUNTRY STOREENT_ID CODE SUBCLASS STATE 	
	JURSTGPREL (common between tax and shipping)	<ul style="list-style-type: none"> Changing shipping settings using the Shipping notebook. Shipping notebook edits the following database columns: <ul style="list-style-type: none"> JURST_ID JURSTGROUP_ID SUBCLASS 	
	SHIPMODE	<ul style="list-style-type: none"> Changing shipping settings using the Shipping notebook. Shipping notebook edits the following database columns: <ul style="list-style-type: none"> CODE CARRIER SHIPMODE_ID 	
	SHPMODEDSC	<ul style="list-style-type: none"> Changing shipping settings using the Shipping notebook. Shipping notebook edits the following database columns: <ul style="list-style-type: none"> SHIPMODE_ID LANGUAGE_ID 	
	CALMETHOD (common between tax and shipping)	<ul style="list-style-type: none"> Edit store archive XML files. 	
	CALCODE (common between tax and shipping)	<ul style="list-style-type: none"> Changing shipping settings using the Shipping notebook. Shipping notebook edits the following database columns: <ul style="list-style-type: none"> CALCODE_ID CODE CALUSAGE_ID STOREENT_ID GROUPBY CALMETHOD_ID 	
	CALCODEDSC (optional)	<ul style="list-style-type: none"> Changing shipping settings using the Shipping notebook. 	

Store	Database Tables Database Assets	Store Archive Edit Options	Database Edit Options
ship fulfillment	SHPJCRULE (at least one default rule for store)	<ul style="list-style-type: none"> Changing shipping settings using the Shipping notebook. Shipping notebook edits the following database columns: <ul style="list-style-type: none"> – CALRULE_ID – SHPARRANGE_ID – JURSTGROUP_ID 	
	SHPARRANGE	<ul style="list-style-type: none"> Changing shipping settings using the Shipping notebook. Shipping notebook edits the following database columns: <ul style="list-style-type: none"> – SHARRAND_ID – STORE_ID – FFMCENTER_ID – SHIPMODE_ID 	
store catalog	STORECAT	<ul style="list-style-type: none"> Edit store archive XML files. 	
	STORECENT	<ul style="list-style-type: none"> Edit store archive XML files. 	
	STORECGRP	<ul style="list-style-type: none"> Edit store archive XML files. 	
	DISPENTREL	<ul style="list-style-type: none"> Edit store archive XML files. 	
	DISPCGPREL	<ul style="list-style-type: none"> Edit store archive XML files. 	
store fulfillment	INVENTORY	<ul style="list-style-type: none"> Edit store archive XML files. 	
offerings	TRADEPOSCN (1)	<ul style="list-style-type: none"> Edit store archive XML files. 	
	MGTRDPSCN (optional, for customer group)	<ul style="list-style-type: none"> Edit store archive XML files. 	
	OFFER	<ul style="list-style-type: none"> Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	OFFERDESC	<ul style="list-style-type: none"> Edit store archive XML files. 	
	OFFERPRICE	<ul style="list-style-type: none"> Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	LISTPRICE	<ul style="list-style-type: none"> Edit store archive XML files. 	
command	URLREG	<ul style="list-style-type: none"> Edit store archive XML files. 	
	CMDREG	<ul style="list-style-type: none"> Edit store archive XML files. 	
	VIEWREG	<ul style="list-style-type: none"> Edit store archive XML files. 	

Store Database Assets	Database Tables	Store Archive Edit Options	Database Edit Options
currency	CURCONVERT	<ul style="list-style-type: none"> Edit store archive XML files. 	
	CURFORMAT	<ul style="list-style-type: none"> Edit store archive XML files. 	
	CURFMTDESC	<ul style="list-style-type: none"> Edit store archive XML files. 	
	CURCVLIST	<ul style="list-style-type: none"> Edit store archive XML files. 	
	CURLIST	<ul style="list-style-type: none"> Changing store information using the Store Profile notebook. Store Profile notebook edits CURRSTR. 	
campaign	EMSPOT	<ul style="list-style-type: none"> Edit store archive XML files. 	<ul style="list-style-type: none"> e-Marketing Spot Management tools in WebSphere Commerce Accelerator
	CAMPAIGN	<ul style="list-style-type: none"> Edit store archive XML files. 	<ul style="list-style-type: none"> Campaign Management tools in WebSphere Commerce Accelerator
store catalog	CATENCLAPPID	<ul style="list-style-type: none"> Edit store archive XML files. 	<ul style="list-style-type: none"> Product Management tools in WebSphere Commerce Accelerator
	CATGPCALCD	<ul style="list-style-type: none"> Edit store archive XML files. 	
store	STOREDEF	<ul style="list-style-type: none"> Edit store archive XML files. 	
consistency check		<ul style="list-style-type: none"> Edit store archive XML files. 	
payment	CMDREG	<ul style="list-style-type: none"> Changing payment settings. Edit store archive XML files. 	
	VIEWREG	<ul style="list-style-type: none"> Changing payment settings. Edit store archive XML files. 	
business policy	POLICY	<ul style="list-style-type: none"> Edit store archive XML files. 	
	POLICYCMD	<ul style="list-style-type: none"> Edit store archive XML files. 	
organization	ORGENTITY	<ul style="list-style-type: none"> Edit store archive XML files. 	
	MPRREL	<ul style="list-style-type: none"> Edit store archive XML files. 	
	ADDRBOOK	<ul style="list-style-type: none"> Edit store archive XML files. 	
	ADDRESS	<ul style="list-style-type: none"> Edit store archive XML files. 	

Store Database Assets	Database Tables	Store Archive Edit Options	Database Edit Options
business account	TERMCOND	The account assets cannot be updated and republished through the store archive. If you need to change the account assets you must edit the assets in the database using the Business Relationship Management tools in the WebSphere Commerce Accelerator.	Business Relationship Management tools in WebSphere Commerce Accelerator Important: Accounts cannot be loaded using the Loader Package.
	ACCOUNT		
	TRADING		
	TCDESC		
	PATTRVALUE		
	CREDITLINE		
	TRDDESC		
	POLICYTC		
	BUYERPO		
	PARTICIPNT		
	ATTACHMENT		
	TRDATTACH		
contract	CONTRACT	<ul style="list-style-type: none"> Editing contract XML files in the store archive 	Business Relationship Management tools in WebSphere Commerce Accelerator Important: Contracts cannot be loaded using the Loader Package.
	TERMCOND		
	PRODUCTSET		
	TRADING		
	TCDESC		
	PATTRVALUE		
	TRDDESC		
	POLICYTC		
	PARTICIPNT		
	TRADEPOSCN		
	ATTACHMENT		
	OFFER		
	TRDATTACH		
	OFFERPRICE		
	STORECNTR		
	PURCHASELT		
	STOREDEF (used by Store Services)		
APRVSTATUS			
FLINSTANCE			

Changing Web assets

To change the Web assets in your store archive, you have the following options:

- Import the store archive into WebSphere Studio and make the necessary changes to the Web assets using Page Designer or a tool of your choice. If necessary, create new store pages. Then export the Web assets back to the store archive or the running store.

- Download the Web assets from the store archive to a location of your choice using the Web Asset dialog in Store Services, then change them using a tool of your choice. Or replace the Web assets in the store archive with your existing Web assets using the Web Asset dialog in Store Services. If necessary, create new store pages.
- Manually open the Web assets compressed archived file in the store archive and make changes to existing files or add new files.

Publishing a sample store archive from Store Services

Publishing a sample store archive to a WebSphere Commerce Server allows you to create a running store. To publish a store archive, complete the following steps:

1. Ensure that you have Site Administrator or Store Administrator access. If you have Store Administrator access, ensure that the access is for all stores.
2. Ensure that you have completed the tasks in Ensure appropriate services and servers are running.
3. Open Store Services.
4. From the **Store Archive** list, select the store archive you wish to publish.
5. Click **Publish**.
The Publish Store Archive page displays.
6. Select your desired publishing options. For more information on publishing options, click **Help**.
Tip: To create a fully functional store, select all publishing options, including the product data option, the first time you publish a store archive.
7. Select **OK**.
While the store publishes you are returned to the Store Archive list page. The publishing state is reflected in the Publish status column. Click **Refresh** to update the status.
8. Select the store archive from the list and click **Publish Summary** to see the results of the publish.
9. When publishing is complete, click **Launch Store** to view and test your store. When you have finished, bookmark the site, and close the browser.

Important:

1. If you change the Web application Web path or the Web application document root, you must ensure that they match the paths defined in the WebSphere Commerce Server.
2. Only one store archive at a time can be published. Concurrent publishing is not supported and causes the publish of both stores to fail.
3. During publish, the consistency checker confirms that the files referenced by the store archive exist. If the consistency check finds an error, the error will be written to the log. Publishing continues as normal.
4. Before republishing a store, delete the files from following directory:

▶ NT

drive:\WebSphere\CommerceServer\instances\instancename\cache

▶ 2000

drive:\Program Files\WebSphere\CommerceServer\instancename\cache

▶ AIX

/usr/WebSphere/CommerceServer/instances/instancename/cache

▶ Solaris

/opt/WebSphere/CommerceServer/instances/instancename/cache

▶ 400

/QIBM/UserData/WebCommerce/instances/*instancename*/cache

While in the store development phase, you may want to disable caching. For more information, see [Configure caching](#).

5. When you launch the store from Store Services, you are logged into the store with the same user name and password that you used to log into Store Services. If you change your password in the store, you are also changing it for Store Services. Instead, to test the features in the store, including changing your password, save the site address, close all browser windows, then log on to the store again.
6. If you are logged in as the default administrator, you cannot browse a store based on the business to business sample store. Instead create a new user that belongs to the default organization, then browse the store.
7. Certain store data assets, like catalogs and fulfillment centers, are shared between stores. As a result, if you publish multiple stores based on the same sample store, the catalog and fulfillment center will be the same for each store. If you have made changes in one catalog, publishing another store on the same sample will overwrite these changes. For more information and how to avoid overwriting any changes, see [Shared data assets](#).

Publishing a store archive using the command line (Windows)

Although the primary method of publishing a store archive is through Store Archive Services, you can also publish a store archive using the command line. To publish using the command line, do the following:

1. Ensure that you have Site Administrator or Store Administrator access. If you have Store Administrator access, ensure that the access is for all stores.
2. Type the following command, using valid parameters for your store archive:

```
publishstore sarName hostname logonId logonPwd {insert|update}
  destination1=webapp.zip,destination2=properties.zip
```

where:

- *sarName* is the store archive name. *sarName* is case sensitive. Ensure you use the correct case.
- *hostname* where *hostname* is the fully qualified TCP/IP name of your WebSphere Commerce Server the tools port number for your instance. You can find the tools port number in the Configuration Manager, under **Instance properties > WebSphere**. By default this is `hostname:8000`.
- *logonId* is the WebSphere Commerce user ID.
- *logonPwd* is the user logon password for WebSphere Commerce
- *insert|update* determines whether the store is being created (insert) or updated (update).
- *{ALL|NOCATLG}* determines which XML files in the SAR should be published. To publish all, use ALL. To publish everything except for catalog, use NOCATLG.
- *destination1=webapp.zip,destination2=properties.zip*, is the list of the file asset files in the SAR, for example `webapp.zip`, and the paths to which they will be published, for example, *destination1* is

▶ NT

```
drive:\Websphere\AppServer\installedApps\
WC_Enterprise_App_instancename.ear\wcstores.war
```

▶ 2000

```
drive:\Program Files\Websphere\AppServer\installedApps\
WC_Enterprise_App_instancename.ear\wcstores.war
```

destination2 is

▶ NT

```
drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_demo.ear\
wcstores.war\WEB-INF\classes
```

▶ 2000

```
drive:\ProgramFiles\WebSphere\AppServer\installedApps\
WC_Enterprise_App_demo.ear\wcstores.war\WEB-INF\classes
```

The following is an example of the command:

```
publishstore mysar.sar myhost wcsadmin wcsadmin insert ALL
"d:\websphere\AppServer\installedApps\WC_Enterprise_App_demo.ear\
wcstores.war=webapp.zip,d:\websphere\AppServer\installedApps\
WC_Enterprise_App_demo.ear\wcstores.war\WEB-INF\classes=properties.zip"
```

3. Open Internet Explorer. Go to the following Web address:
http://hostname/webapp/wcs/stores/store_directory/index.jsp, where the *store directory* is the directory of the store you just published.
Your store displays.

Note: If you are publishing a store archive created with WebSphere Commerce Suite, version 5.1, you will need to complete the following steps before launching the store through the URL:

1. If you are using DB2, do the following. If you are using Oracle, go to step 2.
 - a. When the command has finished running, from the **Start** menu, select **Programs, DB2 for Windows NT**, then **Command Window**.
 - b. In the **DB2 CLP** window, type `db2 connect to dbname`, where `dbname` is the database to which you are publishing your store. Press **Enter**.
 - c. In the command line, type `db2 select * from store`. Press **Enter**. A list of stores displays. Take note of the number of the store you created.
 - d. In the command line, type `db2 select * from catalog`. Press **Enter**. A list of catalogs displays. Take note of the number of the sample store catalog.
 - e. Go to step 3.
2. If you are using Oracle, do the following:
 - a. When the command has finished running, from the **Start** menu, select **Programs, Oracle - HomeOra81, Application Development**, then **SQL Plus**.
 - b. In the window, type your user name and password and host string.
 - c. In the SQL Plus window, type `select * from store;`. Press **Enter**. A list of stores displays. Take note of the number of the store you created.
 - d. In the SQL Plus window, type `select * from catalog;`. Press **Enter**. A list of catalogs displays. Take note of the number of the sample store catalog.
 - e. Go to step 3.
3. Open Internet Explorer. Go to the following URL:
<http://hostname/webapp/wcs/stores/servlet/StoreCatalogDisplay?storeId=storeId from step1c or 2c&langId=-1&catalogId=catalogId from step1d or 2d>
Your store displays.
If you have problems displaying your store, see Troubleshooting publishing.

Important:

1. During publish, the consistency checker confirms that the files referenced by the store archive exist. If the consistency check finds an error, the error will be written to the log. Publishing continues as normal.
2. Before republishing a store, delete the files from following directory:

▶ NT

drive:\WebSphere\CommerceServer\instances*instancename*\cache

▶ 2000

drive:\Program Files\WebSphere\CommerceServer\instances*instance name*\cache

3. While in the store development phase, disable caching triggers and the cache. Leaving the cache on may result in the following:
 - Changes made to JSP files may not display in the browser.
 - Caching triggers will be invoked during publish, when the database is updated. Caching triggers may generate unnecessary database activity that could result in a database transaction log overflow. For more information, see *Configure caching*.
4. If you are logged in as the default administrator, you cannot browse a store based on the business to business sample store. Instead create a new user that belongs to the default organization, then browse the store.

Setting up Payment Manager for your store

You can complete the Payment Manager setup for your store using the Administration Console or the Payment Manager user interface. If you use the Administration Console, menu items appear on the **Payment Manager** menu. If you use the Payment Manager user interface, menu items appear under **Administration** in the navigation frame.

If you create your store using the sample store archive (which is recommended), Payment Manager will be partially configured.

To complete the setup of Payment Manager for your store, do the following:

1. Open the Administration Console or Payment Manager user interface.
2. Assign Payment Manager user roles to WebSphere Commerce users as required. To assign Payment Manager user roles, select **Users**.

The default WebSphere Commerce Site Administrator, `wcsadmin`, is assigned the Payment Manager administrator role by default. You may wish to assign other WebSphere Commerce users various Payment Manager roles.

3. Authorize cassettes for your store by doing the following:
 - a. Select **Merchant Settings**.
 - b. Click your store name in the **Merchant name** column.
 - c. Select the cassettes you wish to authorize for your store.
 - d. Click **Update**.

Important: If you created your store manually, you must add a new merchant (your store) in order to authorize cassettes for your store. When creating a new merchant, the merchant number specified must match the WebSphere Commerce Store ID. You can create a new merchant by selecting **Merchant Settings** then clicking **Add a Merchant**.

4. Configure cassettes for your store by doing the following:
 - a. Select **Merchant Settings**.
 - b. Select a cassette to configure by clicking the icon appearing in the row for your store and the column for the cassette you want to configure.
 - c. Click **Accounts** on the cassette's page for your store and do one of the following:
 - To change existing accounts, click the account name.
 - To create a new account, click **Add an Account**.

For more information on configuring the Cassette for BankServACH, refer to the *IBM WebSphere Payment Manager for Multiplatforms, Cassette for BankServACH Supplement, Version 3.1*.

For information on configuring the OfflineCard and CustomOffline cassettes, refer to the *IBM WebSphere Payment Manager for Multiplatforms, Administrator's Guide, Version 3.1*.

For information on configuring the Cassette for SET, refer to the *IBM WebSphere Payment Manager for Multiplatforms, Cassette for SET Supplement, Version 3.1*.

For information on configuring the Cassette for CyberCash, refer to the *IBM WebSphere Payment Manager for Multiplatforms, Cassette for CyberCash Supplement, Version 3.1*.

For information on configuring the Cassette for VisaNet, refer to the *IBM WebSphere Payment Manager for Multiplatforms, Cassette for VisaNet Supplement, Version 3.1*.

For help when using Payment Manager from the Administration Console or the Payment Manager user interface, click



in the upper right corner of the Payment Manager page you are working on.

For more information on the above or any other Payment Manager administration tasks, refer to the *IBM WebSphere Payment Manager for Multiplatforms, Administrator's Guide, Version 3.1*.


For details on installing WebSphere Payment Manager, refer to the following:

- *IBM WebSphere Commerce Business Edition, Installation Guide, Version 5.4*
- *IBM WebSphere Payment Manager for Multiplatforms, Install Guide, Version 3.1*.

Creating scheduled jobs for a sample store

After publishing a sample store, scheduled jobs needs to be created for the store. The following table shows the scheduled jobs that must be created for each sample store:

Sample store	Scheduled jobs required
InFashion	<ul style="list-style-type: none">• BalancePayment• PayCleanup• ReturnCreditAndCloseScan
NewFashion	<ul style="list-style-type: none">• BalancePayment• PayCleanup• ProcessBackorders• RAreallocate• ReleaseExpiredAllocations• ReleaseToFulfillment• ReturnCreditAndCloseScan
WebFashion	<ul style="list-style-type: none">• BalancePayment• PayCleanup• ReturnCreditAndCloseScan

Sample store	Scheduled jobs required
 ToolTech	<ul style="list-style-type: none"> • BalancePayment • PayCleanup • ProcessBackorders • RAReallocate • ReleaseExpiredAllocations • ReleaseToFulfillment • ReturnCreditAndCloseScan
WebAuction	<ul style="list-style-type: none"> • BalancePayment • PayCleanup • ReturnCreditAndCloseScan

Here is a brief descriptions of the jobs:

BalancePayment

This job calls the DoDepositCmd task command to capture payment once the order has been shipped. This command implements the automatic payment capture function for WebSphere Commerce.

PayCleanup

This job cancels WebSphere Commerce orders with payment authorization requests rejected by the respective financial institutions for longer than a store configured period of time.

ProcessBackorders

This job allocates inventory to backorders which were created when inventory was not available.

RAReallocate

(Redistribute allocations against expected inventory) This job redistributes open Expected Inventory Records (EIR) against existing backorders. This is required to more accurately predict when backordered order items will be available as EIR information is added or modified, and as previously backordered items are deleted or allocated.

ReleaseExpiredAllocations

This job returns allocated inventory back into the receipt table from a previously allocated order line item which has exceeded its expiration time limit.

ReleaseToFulfillment

This job releases allocated items on an order to fulfillment.

ReturnCreditAndCloseScan

This job scans for return merchandise authorizations that are eligible to be credited and marked closed.

To create a scheduled job, follow the instructions found in [Scheduling a store level job](#) for each job. The following table lists the recommended parameters for each job:

Scheduled job name	Recommended start time	Recommended interval (seconds)	Recommended priority
BalancePayment	00:00	86400	1
PayCleanup	00:00	86400	1
ProcessBackorders	00:00	43200	8
RAReallocate	00:00	86400	1
ReleaseExpiredAllocations	00:00	3600	8
ReleaseToFulfillment	00:00	600	10

Scheduled job name	Recommended start time	Recommended interval (seconds)	Recommended priority
ReturnCreditAndCloseScan	00:00	86400	1

Setting the start time to 00:00 starts the scheduled job immediately.

Note: The **Job Parameters** field in the Schedule Job window does not need to be filled in for these jobs.

Configuring e-mail notification for a sample store.

The following procedure enables customer e-mail notification when payment is authorized, when an order is authorized, and when an order is cancelled. Note that you must have a mail server set up in order to e-mail customers.

Note: If you do not have a mail server set up, you will not be able to send e-mail notifications from your store, but the rest of the features of the sample store will still work.

The different sample stores support different e-mail notifications. The following table shows the e-mail notifications supported for each store:

Sample Store	E-mail notifications supported	Message type
InFashion	Password reset	Notification message for password reset
NewFashion	Authorized order	Notification message for a authorized order
	Password reset	Notification message for password reset
	Submission order	Notification message for a received order
	Canceled order	Notification message for a canceled order
	Shipping notification	Message for notifying the customer of an order release manifestation
WebFashion	Authorized order	Notification message for a authorized order
	Password reset	Notification message for password reset

Note: The WebAuction sample store is based on WebFashion. To set up e-mail notification you need to do all of the steps for WebFashion plus additional auction-related steps. For more information on the e-mail notification steps for WebAuction, see the Related Tasks below.

To enable e-mail notifications, do the following:

1. Ensure that the IBM WebSphere Application Server administration server is started.
2. Open the Administration Console using a Site Administrator ID.
3. On the Administration Console Site/Store Selection page select **Store**. The Select Store and Language section displays.
4. From the **Name** drop-down, select the store.
5. From the **Language** drop-down list, select the language. Click **OK**. The Store Administration Console home page displays.

6. From the **Configuration** menu, click **Transports**. The Transport Configuration page displays.
 - a. Verify that the **E-mail** transport has a status of Active.
If e-mail is inactive, select it, then click **Change Status**.
 - b. Select **E-mail** then click **Configure**. The Transport Configuration Parameters page displays.
 - c. In the **Host** field, type your fully qualified mail server name, for example, `myserver.ibm.com`.
 - d. In the **Protocol** field, type `smtp` in lowercase letters, or the protocol of your choice. Click **OK**.
7. From the **Configuration** menu, click **Message Types**. The Message Type Configuration page displays.
8. Create the notification to be sent when payment is authorized as follows:
 - a. Click **New**. The Message Transport Assignment page displays.
 - b. Select the message type from the **Message Type** drop-down list. See the table above for the message types to use in your store.
 - c. In the **Message Severity** field, type 0 to 0
 - d. From the **Transport** drop-down list, select **E-mail**.
 - e. From the **Device Format** drop-down list, select **Standard Device Format**.
 - f. Click **Next**. The Message Transport Assignment Parameters page displays.
 - g. Complete the fields as follows:

Host	The fully qualified name of your mail server, for example, <code>example.ibm.com</code>
Protocol	Type <code>smtp</code> (you must use lowercase letters), or the protocol you are using.
Recipient	Enter a valid e-mail address. This address will be replaced by the customer's e-mail address at run time.
Sender	Enter an email address that you want to use as the sender of the message, for example, <code>orders@example.ibm.com</code> . The address must be an email address for a valid user on the mail server.
Subject	Enter the text that you want to display as the subject line of the message, for example, Your order has been accepted.

- h. Click **Finish**. The Message Type Configuration page displays.
9. For each message type in your sample store, repeat step 8.

Notes:

- It will often take a long time before canceled order notification e-mail is sent. The time can be shortened by:
 1. Setting the value of the `REJECTEDORDEREXPIRY` column in the `STORE` database table to a smaller value.
 2. Change the `PayCleanup` scheduled job to have a smaller schedule interval.
- For more information about shipping notifications, see `ReleaseShipNotify` message

Chapter 2. Sample store database assets

Store database assets

Store data is the information loaded into the WebSphere Commerce Server database, which allows your store to function. In order to operate properly, a store must have the data in place to support all customer activities. For example, in order for a customer to make a purchase, your store must contain a catalog of goods for sale (catalog data), the data associated with processing orders (tax and shipping data), and the inventory to fulfill the request (inventory and fulfillment data).

Data may be exclusive to a store, or shared between stores. For more information, see *Shared data assets*.

The store database assets in the sample store archives provided with WebSphere Commerce are well-formed, XML files valid for the Loader package, with the following exceptions: the store archive XML files are intended to be portable and should not contain generated primary keys that are specific to a particular instance of the database. Instead they use internal-aliases (outlined in the Store archive Loader conventions) which are resolved by the IDResolver at the time of publish. The store archives also use a set of DTD macros (known in XML as entities). The macros act as place holders for values you select in Store Services during store creation. The use of these two conventions allows the sample store archives to be copied and published multiple times.

The sample store archive includes all the database assets necessary to create a functional store. You can modify these files for use in your own store archive, or use them as a guide for creating your own XML files. WebSphere Commerce requires that certain data be loaded into the WebSphere Commerce database to create a functional store, and that this data must be loaded in the order determined by the schema. For example, the FFMCENTER table must be populated before the STOREENT table. Since the sample stores include all the mandatory data in the order and structure that WebSphere Commerce requires, using the database assets as a base, or guide, for your own store saves you a substantial amount of time during the initial creation period.

For a list of the database asset files used in the sample store archive, see *Sample store archive database assets*. For more detailed information on store data, see the *IBM WebSphere Commerce Store Developer's Guide*.

Note: The DTDs for the sample store database asset XML files are not in the store archive files. They are located in the following directory:

▶ NT

`drive:\WebSphere\CommerceServer\xml\sar`

▶ 2000

`drive:\Program Files\WebSphere\CommerceServer\xml\sar`

▶ AIX

`/usr/WebSphere/CommerceServer/xml/sar`

▶ Solaris

`/opt/WebSphere/CommerceServer/xml/sar`

▶ 400

`/QIBM/ProdData/WebCommerce/xml/sar`

NewFashion Catalog database assets

The catalog.xml file stores the catalog information for the sample stores in WebSphere Commerce. Each sample store has its own catalog.xml file. For more information see the catalog.xml file.

NewFashion catalog database assets are divided into the following sections:

- Catalog groups
- Catalog entities
- ATP feature

Catalog groups

Catalog groups are groups of categories and products. A category is, a catalog group in itself. For example, Men's Fashions is a grouping of the categories that make up Men's Fashions such as Pants and Shirts, while the category Pants, is a grouping of products.

In the NewFashion sample store there are several catalog groups:

- Men's Fashions
- Women's Fashions
- New Arrivals
- Homepromo

Catalog groups are created in the CATGROUP table and are associated with a specific catalog in the CATTOGRP table. Catalog groups can have top categories and sub-categories. For example, a top category could be Men's Fashions, and a sub-category Pants. If you have a sub-category, you must put it under the top category in the CATGRPREL table.

Note: Products in the HomePromo category or New Arrival's category appear in the special categories only. After the promotions are over, these products must be moved back to their regular category.

Catalog entities

The NewFashion sample store catalog is made up of catalog entities. These entities are made up of the following:

- products
- items
- packages
- bundles

Information about catalog entities is stored in the CATENTRY table. The CATENTREL table stores relationships between catalog entities such as product-item, bundle, and package relationships. For more information on entities see Packages and bundles.

Products

For each product in the NewFashion store there is an entry in the BASEITEM table. A base item is a product. Each product can only have one version which is determined by an expiration date. Therefore, you can have only one women's red shirt size 12, that expires on January 1, 2010. Version information per product, is stored in the ITEMVERSN table.

In the NewFashion store, inventory is not shared with other merchants. As a result, every product in the catalog has only one entry in the DISTARRANG table. Each row of the DISTARRANG table represents a distribution arrangement, enabling a store to sell its own inventory.

Items

An item is a specific instance of a product, defined by attributes. The information held about a specific item is stored in the ITEMSPC table. Each item's inventory information is held in the RECEIPT table. The date that a backordered item is expected to be received is found in the RADETAIL table.

Note: In the NewFashion store, the catalog uses pricing for items in the OFFERPRICE table. For more information see the offering.xml file.

Packages

Packages are products that are sold together such as a golf package. A package appears as one line in a shopping cart. Products in a package cannot be purchased separately. Products are set up in the product.xml file, and are grouped into packages in the CATENTRY table. Each product under the package has a separate entry.

Bundles

Bundles are products that are grouped together but can be sold separately. Every product in a bundle appears as one line in the shopping cart. Bundles are stored in the CATENTRY table, and products associated with a bundle are stored in the CATGPENREL table. The product-bundle relationship is defined in CATENTREL table.

The attributes for catalog entities, packages, and bundles are set up in the PKGATTR and PKGATTRVAL tables. The PKGATTR table refers back to a existing attribute table belonging to the product in the package. The PKGATTRVAL table refers back to the product's attribute value in the package.

ATP Feature

The NewFashion store has some special items that showcase the ATP feature, such as the following:

- Mens,ActiveWear,TeamShirt has zero on-hand inventory.
- Womens,Accessories,Evening Bag is forced backorder.
- Mens, Active Wear, TeamShirt has RADetails.

If an item is zero on hand or forced backorder, no matter how many of these items the customers order, they will automatically go on backorder. No RADetail means that the expectation date for the backorder will be current date plus the store's default backorder offset, (DEFAULTBOFFSET column in store's table), which is set to 30 days.

NewFashion contract and business policy database assets

In WebSphere Commerce, all stores must have a default contract. The default contract is made up of terms and conditions which are stored in the contract.xml file. If the terms and conditions refer to a business policy, the business policy information is stored in the businesspolicy.xml file.

NewFashion contract and business policy database asset information is divided into the following sections:

- Contract
- Business policy

Contract

Terms and conditions

All contracts in the sample stores must have terms and conditions. Terms and conditions are the set of rules agreed upon in a contract, and control the purchasing process between a buyer and a seller. For every contract you must have one set of terms and conditions for the price.

You can also have terms and conditions for other types of charges such as shipping charges. Each contract must have only one shipping charge terms and conditions.

The buyer and seller must be referred to in the `contract.xml`.

Business policy

Terms and conditions

A policy outlines the rules followed by a business for a particular business process. If the terms and conditions of your contract refer to a business policy, you must populate `businesspolicy.xml` before importing the `contract.xml` file.

Note: Some policies referenced in `contract.xml` are not contained in `businesspolicy.xml`. These policies are part of the bootstrap data. For more information on the bootstrap data see [Bootstrap Files](#).

NewFashion Shipping database assets

The NewFashion shipping database assets are stored in the following XML files:

- `shipping.xml`
- `store-catalog-shipping.xml`
- `store-defaults.xml`
- `shipfulfill.xml`

NewFashion shipping database assets are divided into the following sections:

- Jurisdictions
- Shipping modes
- Calculation codes
- Calculation rules
- Calculation scale
- Calculation range
- Calculation lookup
- Calculation combinations
- Shipping fulfillment

Jurisdictions

The `shipping.xml` file identifies jurisdictions for shipping. Jurisdictions are defined in the `JURST` table, `JURSTGROUP` assigns the jurisdiction to a group and sub-class, and `JURSTPREL` assigns the jurisdiction and jurisdiction group to the same subclass.

Shipping modes

A shipping mode is a combination of a shipping carrier and its shipping service. For example, XYZ Carrier, Overnight is a shipping mode. Information on shipping modes is stored in the `SHIPMODE` table.

Calculation codes

Calculation codes are used to calculate discounts, shipping charges, sales tax, and shipping tax. The `shipping.xml` file contains all the calculation codes shipping. The `CALCODE` table sets up the calculation codes for shipping. The `displaylevel` field shows a number displaying what amount was calculated.

- 0 = Order Item
- 1 = Order
- 2 = Product
- 3 = Item
- 4 = Contract

Calculation rules

Each calculation code has a set of calculation rules which define how the calculation will be done. For example, if you are shipping goods to one region you may apply certain rules to the calculation, and if you are shipping goods to another region, you may apply different rules to the calculation. The `CALRULE` table stores the calculation rules for shipping. The `flag` field specifies whether the `CalculationCodeQualifyMethod` of the specific `CalculationCode` should be invoked.

- 0 = The method will not be invoked.
- 1 = The method will be invoked.

Calculation scale

A calculation scale is the set of ranges that applies to the calculation. For example, for shipping costs you may have a set of weight ranges that each correspond to a particular cost. That is, a product that weighs between 0 to 5 kg might cost \$10.00 to ship and a product weighing 5 to 10 kg might cost \$15.00 to ship. The `CALSCALE` table stores the scale code for shipping, one for per order and one for per item.

Note: The `CALSCALE` table stores the scale code for currencies if they apply to the corresponding range.

Calculation range

The range for the scale codes is stored in the `CALRANGE` table. `calmethod_id_10` is used for per order shipping and `calmethod_id_11` for per item shipping.

Calculation lookup

Calculation lookup values are the values associated with the calculation scale. The calculation lookup values for a product that weighs between 0 to 5 kg might and costs \$10.00 to ship and a product weighing 5 to 10 kg that costs \$15.00 to ship, would be \$10.00 and \$15.00 respectively. There is one lookup value per currency for a given `CALRANGE` ID. The `CARLOOKUP` table defines the lookup ID and value.

Calculation combinations

Calculation rules and the scale ranges are combined in the `CRULESCALE` table as seen in the code below: Calculation methods and rules are combined in the `STENCALUSG` table. The store default for calculations are also stored in this table. The `usageflag` field controls how the `OrderPrepare` command uses the calculation.

- 1 = use - use this `CalculationUsage`.
- 2 = check - throw an `ECApplicationException` if this calculation does not produce a value for an order item.

Shipping fulfillment

Shipping fulfillment assets associate a shipping jurisdiction group to the calculation rules, and fulfillment center to the shipmode for the store. The shipping fulfillment information is stored in the SHPJCRULE and SHPARRANGE tables and can be seen below as displayed in shipfulfill.xml.

Sample store access control database assets

In WebSphere Commerce all stores have access control policies. An access control policy authorizes users or a group of users to perform particular actions. There are two access control policy files for each store:

- *samplestorenameAccessPolicies.xml*
- *samplestorenameAccessPolicies_locale.xml*

Both the *AccessPolicies.xml* and *AccessPolicies_locale.xml*, are high level native access control files. *AccessPolicies.xml* is national language independent, whereas *AccessPolicies_locale.xml* is national language dependant. Each file includes the possible actions, action groups, resources, and policy definitions used in the sample stores. These files are transformed into the *AccessPoliciesOut.xml*, and *AccessPoliciesOut_locale.xml* files respectively. Each transformed file populates the database. For more information on transforming the access policy files, refer to the related links below.

Note: Only the transformed files can be mass loaded or used directly in the SAR file, the pre-transformed files cannot.

The database asset information for the sample stores can be divided into the following sections:

- Actions
- Resource Categories
- Resource Groups
- Action Groups
- Policy Definitions

Actions

Actions that can be performed under the access control policy are defined in each sample store's *AccessPolicies.xml* file.

Resource Categories

Resource categories define protectable resources.

Resource Groups

Resource groups contain the resources controlled by the access control policy. A resource group can include business objects such as "contract" or "exchange position," or a set of related commands. Each sample store's *AccessPolicies.xml* file defines the resource groups in the policy.

Action Groups

Action groups define the actions that can be performed on the resource groups in the access control policy. These groups are defined in the *AccessPolicies.xml* file for each store.

Policy Definitions

Each sample store's policy is defined in the individual store's `AccessPolicies.xml` file. The ToolTech sample store has two policies.

Chapter 3. NewFashion sample store

NewFashion sample store

NewFashion is one of the business-to-consumer online fashion stores provided with WebSphere Commerce. It implements many of the most commonly used features in today's top retail sites. Some of the features included in the NewFashion sample store are:

- Multicultural support
- Availability dates for order items
- Backordering of items not currently in stock
- Splitting of orders based on customer preference
- Tracking order status
- E-mail notification of order status
- Search capability
- Collaboration through real-time customer support

The NewFashion store provides all the pages and features necessary for a functioning online store. NewFashion is packaged with WebSphere Commerce as a store archive, and as a result, no further installation is necessary. All that is required to view the sample store is to create a new store archive based on NewFashion using the Store Services tools, then publish it to the WebSphere Commerce Server. For more information, see [Creating a store archive using Store Services](#).

Since store creation in WebSphere Commerce is based upon selecting a sample store archive and modifying it, NewFashion is designed to act as the basis for your store. It is based on a simple yet proven shopping flow, and all the sample store pages can be easily customized.

For more information on NewFashion's shopping flow, see the sample store shopping flow diagram, and the use cases. The use cases detail the flow of each user interaction in the store, for example, registration or displaying a product. For more technical detail on how each page works, see the corresponding reference information for each page.

Chapter 4. NewFashion store pages

NewFashion Sample store search pages

The search results page, `resultlist.jsp` is displayed after a customer has entered a search request from the home page, `sidebar.jsp`, or the advanced search page, `advancedsearch.jsp`.

Note:The page `subcategory.jsp` does not include `sidebar.jsp` as the sidebar on the page. As a result, search functionality has been built directly into this page and the search features are the same as those for `sidebar.jsp`.

Beans

`resultlist.jsp` uses the following beans:

- `CatalogDataBean`
- `CategoryDataBean`
- `CatEntrySearchListDataBean`
- `CatalogEntryDataBean`

`advancedsearch.jsp` uses the following beans:

- `CatalogDataBean`
- `CategoryDataBean`

Commands

`resultlist.jsp` uses the following commands:

- `ProductDisplay`
- `AdvancedSearchView`
- `CatalogSearchResultView`

`advancedsearch.jsp` uses the following commands:

- `CatalogSearchResultView`

`sidebar.jsp` uses the following commands:

- `CatalogSearchResultView`
- `AdvancedSearchView`
- `LogonForm (Search only)`
- `HelpView (Search only)`
- `StoreCatalogDisplay (Search only)`
- `SetCurrencyPreference (Search only)`

`subcategory.jsp` uses the following commands:

- `CatalogSearchResultView`
- `AdvancedSearchView`

Note:The page `subcategory.jsp` does not include `sidebar.jsp` as the sidebar on the page. As a result, search functionality has been built directly into this page and the search features are the same as those for `sidebar.jsp`.

Implementation

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see *Sample store pages: common implementation techniques*.

When the customer clicks on **GO** from the left navigation screen, `sidebar.jsp`, or `subcategory.jsp`, or clicks on **Submit** from the advanced search page `advancedsearch.jsp`, the `resultlist.jsp` displays.

When the `CatEntrySearchListDataBean` is activated, it goes to the database and retrieves all of the results for the search as seen in the code below.

```
com.ibm.commerce.beans.DataBeanManager.activate(catEntSearchListBean, request);
```

The request parameter is the URL string that is passed to the `resultlist.jsp` page. In the `CatEntrySearchListDataBean`, the method `getResultList()`, returns an array of `CatalogEntryDataBean`'s. Each `CatalogEntryDataBean` is one result that is listed on the search result page. The `getResultList()` method returns a specific number of results per page. This number is specified in the URL request address with the `pageSize` variable when the `CatEntrySearchListBean` is activated. When `result.jsp` is displayed and customers click on the **Previous** or **Next** buttons, `CatalogSearchResultView` lists the results and the `beginindex` URL parameter is passed to `CatalogSearchResultView`. `CatalogSearchResultView` tells `CatEntrySearchListDataBean` at what number result in the search result list the display should begin.

In the *NewFashion* sample store only products are displayed in search results. When the `CatalogSearchResultView` command is called the `resulttype` variable is passed to the command as an URL argument. `resulttype` tells the `CatEntrySearchListBean` to list products, items or both. The `resulttype` values are:

- 1 = List items only.
- 2 = List products only.
- 3 = List both items and products.

The following code lists products only.

```
<input type="hidden" name="resultType" value="2">
```

Limitations

The store catalog follows a hierarchical structure with top categories at the top, such as Men's, Women's, and New Arrivals, and sub-categories beneath them, such as Shirts, and Pants.

When customers select **Advanced Search**, they can only search under the specific top or sub-category chosen. If customers search under the Men's category, only products directly under that category will be displayed. Products under the Men's Shirts or Pants category will not be displayed. To search the Pants or Shirts sub-category, customers must select that specific category, in order for search results to be displayed.

Sample store address book pages

The sample store address book pages allow registered customers to add addresses, including shipping and billing addresses, to an address book.

A registered customer logs in, then clicks **Edit my address book** from the My Account (`account.jsp`) page. The Address Book (`addressbookform.jsp`) page displays, from which a customer can add a new address, or edit an existing one. For more information, see the Add new address use case and the Edit an address use case.

The sample store address book procedures uses the following JSP files:

- `account.jsp` (My Account page)
- `addressbookform.jsp` (Address Book page)
- `addressform.jsp` (includes the parameters for the `AddressForm` command. Does not display to the customer)
- `address.jsp`(Add Address page and Update Address page)
Note: `address.jsp` is used for both the Add Address page and the Update Address page. If the `addressId` is provided, the `address.jsp` loads as an Update Address page. Otherwise, it loads as an Add Address page. If the `addressId` is provided as a parameter for the `AddressAdd` command, the command updates the address of the specified `addressId`. Otherwise, a new address is created.

Commands

`account.jsp` uses the following commands:

- `UserRegistrationForm`
- `AddressBookForm`

`addressbook.jsp` uses the following commands:

- `AddressForm`
- `AddressDelete`

`address.jsp` uses the following commands:

- `AddressAdd`
- `PrivacyView`

Beans

`addressbook.jsp` uses the following beans:

- `AddressAccessBean`

`address.jsp` uses the following beans:

- `ErrorDataBean`
- `AddressDataBean`

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see `InFashion` pages: common implementation techniques.

When a customer clicks **Edit my address book** from the My Account page, the `AddressBookForm` command is called. `AddressBookForm` then loads the Address Book page (`addressbook.jsp`). If a customer clicks **Add a new address** the `AddressForm` command is called. `AddressForm` is registered with `AddressForm.jsp` in the database, and checks the page parameter. If page is set to `newshipaddress` the Checkout 1: Add Billing Address page (`billingaddress.jsp`) loads, otherwise the Add Address page (`address.jsp`) loads.

Note: `AddressForm` will load the `billingaddress.jsp` page if it is set to `newshipaddress` because of the error handling for add billing address. If page equals `newshipaddress`, it means that customers were creating a new address from the billing address page, and attempting to go to the ship address page, but there was an error. As a result, customers are sent back to the Billing Address page.

After the customer completes the fields in the Add Address page (`address.jsp`), it checks to see if the `addressId` exists. If the `addressId` exists, the address book is updated, if not a new address is created.

If the addressId is provided, the address.jsp loads as an Update Address page. Otherwise, it loads as an Add Address page. In the Update Address page, the entry fields are pre-filled with the previous entered values as seen below:

Before a new address is created, a nickname (a unique identifier for the address, including time and date) is created for the address, using the following JavaScript:

When a customer completes the address, and clicks **Submit** in both the Add Address page (address.jsp) and the Update address page (address.jsp), the AddressAdd command is called. The address book page (addressbook.jsp) displays existing addresses.

Error handling

If the customer does not complete required fields in either Add Address (address.jsp) or Update Address page (address.jsp), the system asks them to re-enter the fields.

NewFashion Sample store address book pages

The sample store address book pages allow registered customers to add shipping and billing addresses to an address book.

A registered customer logs in and then clicks **Edit my address book** from the My Account (myaccount.jsp) page. The Address Book (addressbookform.jsp) page displays. On it a customer can add a new address or edit an existing one. For more information, see the Add new address use case and the Edit an address use case.

The sample store address book uses the following JSP files:

- account.jsp (My Account page)
- addressbookform.jsp (Address Book page)
- addressform.jsp (includes the parameters for the AddressForm command. Does not display to the customer)
- address.jsp (Add Address page and Update Address page)
Note: address.jsp is used for both the Add Address page and the Update Address page. If the addressId is provided, the address.jsp loads as an Update Address page. Otherwise, it loads as an Add Address page. If the addressId is provided as a parameter for the AddressAdd command, the command updates the address of the specified addressId. Otherwise, a new address is created.

Commands

account.jsp uses the following commands:

- AddressBookForm
- InterestItemDisplay
- TrackOrderStatus

addressbookform.jsp uses the following commands:

- AddressForm
- AddressDelete
- OrderItemDisplay

address.jsp uses the following commands:

- AddressAdd
- PrivacyView

Beans

addressbook.jsp uses the following beans:

- AddressAccessBean
- AddressBookDataBean

address.jsp uses the following beans:

- ErrorDataBean
- AddressDataBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

When a customer clicks **Edit my address book** on the My Account page, the AddressBookForm command is called. AddressBookForm then loads the Address Book page (addressbookform.jsp). If a customer clicks **Add a new address**, the AddressForm command is called. AddressForm is registered with AddressForm.jsp in the database, and checks the page parameter. If page is set to newshipaddress the Checkout 1: Add Billing Address page (billingaddress.jsp) loads; otherwise the Add Address page (address.jsp) loads.

Note: AddressForm will load the billingaddress.jsp page if it is set to newshipaddress because of the error handling for add billing address. If page equals newshipaddress, it means that customers were creating a new address from the billing address page, and attempting to go to the ship address page, but there was an error. As a result, customers are sent back to the Billing Address page.

After the customer completes the fields in the Add Address page (address.jsp), the system checks to see if the addressId exists. If the addressId exists, the address book is updated. If not, a new address is created. If the addressId is provided, the address.jsp loads as an Update Address page. Otherwise, it loads as an Add Address page. In the Update Address page, the entry fields are pre-filled with the previous entered values as seen below:

```
if (addressId != null)
bUpdateAddress = true;
else
bUpdateAddress = false;
```

When a customer completes the address and clicks **Submit** in both the Add Address page (address.jsp) and the Update Address page (address.jsp), the AddressAdd command is called. The address book page (addressbookform.jsp) displays existing addresses.

Customers can click **Edit Address Book** from the Select shipping address page to go to the Address book. If customers go to the Address book page from the Select shipping address page, the Address book page displays a **Return to Checkout** link. These customers are not allowed to remove an address from the Address Book page. As a result, new code is added to check if customers are coming from the Select shipping address page as seen below:

```
String mode = jhelper.getParameter("mode"); if (mode.equals("AddressBookReturnToCheckout"))
```

Error handling

If the customer does not complete required fields in either Add Address (address.jsp) or Update Address page (address.jsp), the system prompts the customer to re-enter the fields. The following code handles the error:

```

TypedProperty hshErrorProperties = bnError.getExceptionData();

if (hshErrorProperties != null)
{
//There is an error in the submitted address.
strErrorCode = hshErrorProperties.getString(ECConstants.EC_ERROR_CODE, "");
if (strErrorCode.equals(ECUserConstants.EC_ADDR_ERR_BAD_NICKNAME))
strErrorMessage = infashiontext.getString("ERROR_MESSAGE1");

...
}

```

WebFashion and NewFashion Sample store bundle display pages

A bundle display page features promotional groups of products in an online store. It typically includes a description, a list of components that make up the bundle, a price for each component, an image, and a list of attributes (size and color) if the products that compose the bundle have variations, and a list of values for each attribute (red and blue for color and Large and X-Large for size).

Customers can add all products in the bundle to the shopping cart or wish list with a single click. Each product in the bundle displays as a separate line item in the shopping cart, unlike the package, which displays a single line item for the entire package.

For more information on the sample store bundle display pages and how they work, see the Display bundle page use case.

Commands

bundledisplay.jsp uses the following commands:

- OrderItemAdd
- InterestItemAdd

Beans

bundledisplay.jsp uses the following beans:

- BundleDataBean
- CompositeProductDataBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see [Sample store pages: common implementation techniques](#).

Bundledisplay.jsp is registered in the database (in DISPENTREL table) to display all products in the store. Bundledisplay.jsp displays the following:

- Description (including a list of components that make up the bundle) and image of the bundle
- An image, price, and a list of attributes (size and color) for each product in the bundle. Also, a list of values for each attribute (red and blue for color and Large and X-Large for size).
- Add to shopping cart link and Add to wish list link
- Quantity field (NewFashion only)

Description (including a list of components that make up the bundle) and image of the bundle

The BundleDataBean retrieves the description and image of the bundle.

Product image, price, attributes and attribute values

The BundleDataBean calls the ProductDataBean to retrieve the information about each product that composes the bundle. For more information on how the ProductDataBean works, see Sample store product pages.

Add to shopping cart and Add to wish list

For more information, see Sample store product pages.

Quantity field (NewFashion only)

There is a quantity field for each product in the bundle which allows the customer to specify the number of each product to add to the shopping cart.

Sample Store catalog group pages

Catalog group pages help customers navigate through the various departments or groups of products or services available in a store. The first catalog group pages lead customers to the areas in which they want to shop and typically display the name of each catalog group, a brief description, and an image. Subsequent catalog group pages further narrow the selected product type that the customer wishes to browse. The last catalog group page in a shopping path contains links to product pages.

Typically, there are three types of catalog group pages:

- pages that link to sub catalog group pages (catalog group pages)
- pages that link to product pages (product list pages)
- pages that link to both sub catalog group and product pages

For more information on the sample store catalog group pages and how they work, see the View product category use case.

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

Catalog group page display and product list page display

The sample stores feature catalog group pages, which display the subcategories in each of the top-level categories (Men's and Women's) as well as a featured special product, and product list pages, which display a list of the products in each of the subcategories (Pants and Shirts).

All hyperlinks to the top level categories contain an extra parameter called `top` that is set to `Y`. The JavaServer page file `categorydisplay.jsp` is registered in the database (in `DISPCGREL` table) as the page that displays all categories in the store. In this page we check for the existence of the `top` parameter. If it exists, the page includes `topcategory.jsp`. Otherwise `subcategory.jsp` is included. `topcategory.jsp` is the catalog group page and `subcategory.jsp` is the product list page.

Catalog group page

`topcategory.jsp` retrieves a list of subcategories using the `getSubCategories()` method in the `CategoryDataBean`. The featured specials contained in `InFashion` were created in the catalog rather than as part of a campaign with `WebSphere Commerce Accelerator`. Featured special products were added to the catalog group and then retrieved by `topcategory.jsp` using the `getProducts()` method in the `CategoryDataBean`.

In `WebFashion` and `NewFashion`, the men's `topcategory.jsp` displays a bundle, and the women's `topcategory.jsp` displays a package. The bundle is retrieved using the `getBundles()` method in the `CategoryDataBean`, and the package is retrieved using the `getPackages()` method in the `CategoryDataBean`.

Product list page

`subcategory.jsp` displays a list of all products in the catalog group and a list of all sibling

categories (all other categories under the same top-level category) in the left navigation bar. For each product, `subcategory.jsp` displays the product's short description, full image, and price using the following methods: `getDescription().getShortDescription()`, `getCalculatedContractPrice()`, and `getDescription().getFullImage()`. The parent category ID is needed to display the list of sibling categories. To retrieve the parent category ID, the parameter `parent_category_rn` is supplied in the hyperlink, which then constructs a `CategoryDataBean` for the parent category. By default, `CategoryDataBean` retrieves the category ID from the `CategoryId` parameter. However, in this case, ID is stored in `parent_category_rn`. As a result, the category ID must be set explicitly as follows:

```
String parentCategoryId = request.getParameter("parent_category_rn");
parentCategory = new CategoryDataBean ();
parentCategory.setCategoryId(parentCategoryId);
com.ibm.commerce.beans.DataBeanManager.activate(parentCategory, request);
```

A list of products that belong to this catalog group is then retrieved using the `getProducts()` method in the `CategoryDataBean`.

Note:In the `NewFashion` store, the `subcategory.jsp` page also contains a link to the collaboration customer care feature, if enabled in `Store Services`. The `subcategory.jsp` page has the link added because it has its own sidebar and does not include `sidebar.jsp` like the other pages. ` `

If customers click **Live Chat with Customer Care**, a pop-up window is displayed on the screen, and customers can chat online in real-time with a customer service representative. This link will only be displayed if this feature is enabled in `Store Services`. The **Live Chat with Customer Assistance** link is contained within the body of a pair of custom tags (the `<flow:ifEnabled feature="customerCare">` tag and `</flow:ifEnabled>` tag), and can be enabled or disabled based on what option is selected in `Store Services`. As long as the custom tag remains in place, `Store Services` can be used to automatically switch between a site that has collaboration support and one that does not, without having to modify the `JavaServer` page. To permanently enable or disable collaboration support in the page, the custom tags and the **Live Chat with Customer Care** link can be removed from the `JavaServer` page by clicking **Apply Permanently**, in the `Store Services` GUI. It is not recommended to manually remove or alter the custom tags or the code it surrounds. Instead, use the **Apply Permanently** button in `Store Services`.

Note:The custom tag that surrounds the **Live Chat with Customer Care** link cannot be copied into the `JavaServer` pages of other stores. These tags are only intended to work in the store that originally contained them. For more information on collaboration, see the related links below.

Sample store pages: common implementation techniques

Most sample store pages use the following implementation techniques. This page uses the `InFashion` store as an example but can be applied to all sample stores. For more information on techniques specific to an individual page, see the reference file for that page.

Multicultural content

The sample store pages are also used to display multicultural content. That is, the same set of pages can be used for several different locales. Most of the code to enable multicultural display is in `getResource.jsp`. `getResource.jsp` does the following things:

- Retrieves the current locale from the command context and stores it as the `locale` variable.
- Retrieves the store directory and stores it as the `storeDir` variable.
- Retrieves the store's name and stores it as the `storeName` variable.
- Loads the language-specific properties file using the `ResourceBundle` API and stores it as the `infashiontext` variable.

In order for a JavaServer Page file to access the above variables, `getResource.jsp` must be included in the JavaServer Page file, using the compile-time include action:

```
<%@ include file="include/getResource.jsp"%>
```

Since `getResource.jsp` is included in almost all of the samples store JavaServer Page files, it may be executed several times in a single request. To avoid duplication of effort, much of the information retrieved in this page is stored in the request context. For example:

```
String storeDir = (String) request.getAttribute("storeDir"); String includeDir = (String)
request.getAttribute("includeDir"); String fileDir = (String) request.getAttribute("fileDir");
String bundleDir = (String) request.getAttribute("bundleDir");

String storeName = "";

if (storeDir == null) {

storeDir = sdb.getJspPath(); fileDir = sdb.getFilePath(); includeDir = storeDir + "include" +
"/"; bundleDir = sdb.getDirectory(); storeName =
sdb.getDescription(cmdcontext.getLanguageId()).getDisplayName();
request.setAttribute("storeName", storeName); request.setAttribute("storeDir", storeDir);
request.setAttribute("fileDir", fileDir); request.setAttribute("includeDir", includeDir);
request.setAttribute("bundleDir", bundleDir); }
```

Language-specific messages

Language-specific messages like “Thank you for ordering!” are stored in resource bundle properties files. These files are located in the following directory:

▶ NT

```
drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_instance_name.ear\wcstores.war\WEB-
INF\classes\storedir
```

▶ 2000

```
drive:\Program
Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_instance_name.ear\wcstores.war\WEB-
INF\classes\storedir
```

▶ AIX

```
/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_instance_name.ear/wcstores.war/WEB-
INF/classes/storedir
```

▶ Solaris

```
/opt/WebSphere/Appserver/installedApps/WC_Enterprise_App_instance_name.ear/wcstores.war/WEB-
INF/classes/storedir
/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_instance_name.ear/wcstores.war/WEB-
INF/classes/storedir
```

▶ 400

```
/QIBM/UserData/WebASAdv4/WAS_instance_name/installedApps/WC_Enterprise_App_instance_name.ear/wcstores.war
```

For example, if your store directory is “storedir”, then the English property file will be:

▶ NT

```
drive:\drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_instance_name.ear\wcstores.war\WEB-INF\classes\infashiontext_en_US.properties.
```

▶ 2000

```
drive:\Program
Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_instance_name.ear\wcstores.war\WEB-
INF\classes\storedir
```

▶ AIX

```
/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_instance_name.ear/wcstores.war/WEB-
INF/classes/storedir/infashiontext_en_US.properties.
```

▶ Solaris

```
/QIBM/UserData/WebASAdv4/WAS_instance_name/installedApps/WC_Enterprise_App_instance_name.ear/wcstores.war/W
```

Contents of this file are loaded using the Java `java.util.ResourceBundle` API from the JSP file `getResource.jsp`. The bundle is stored as the variable `infashiontext`. A language-specific message is displayed as follows:

```
<title><%=infashiontext.getString("REGISTER_TITLE")%></title>
```

Setting content encoding

Most modern browsers understand HTML data encoded in UTF-8 encoding. However, some older browsers may only understand native encoding. For example, an older Japanese browser may only understand HTML data encoded in “Shift_JIS” encoding. To solve this problem, the sample store specifies language-specific encoding in the resource bundle properties files as the property name `ENCODESTATEMENT`. For example, the English property file `infashiontext_en_US.properties` contains the following entry:

```
ENCODESTATEMENT = text/html; charset=ISO_8859-1
```

For individual JSP files, the encoding is set using the JSP request object as follows:

```
<% response.setContentType(infashiontext.getString("ENCODESTATEMENT")); %>
```

Setting of encoding type needs to be done as early in the JSP file as possible, as the HTTP header is sent out before any HTML content. As a result, if you set the content type (which is sent as HTTP header) after any HTML content, it may not have the desired effect. The browser may not be able to display the data properly.

Including the header, footer, and left navigation frame

Almost all sample store pages display the header (`header.jsp`), footer (`footer.jsp`), and left navigation frame (`sidebar.jsp`) pages. They are included in the other JSP files using the following execution-time include directive:

```
<% String incfile;
```

```
incfile = includeDir + "header.jsp"; %> <jsp:include page="<%=incfile%>" flush="true"/>
```

If you know the exact location of the JSP files, you can simplify the include process by using the following:

```
<jsp:include page="/storedir/include/header.jsp"/>
```

where `header.jsp` is located in the `storedir` directory under the Web application document root.

Sample store contact us page

When a customer clicks the **Contact Us** link, the Contact Us page (`contact.jsp`) displays. It shows the various types of contact information for the store, including phone numbers and locations for “brick and mortar” stores.

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see *Sample store pages: common implementation techniques*.

A new view called `ContactView` is created in the `VIEWREG` table and associated with `contact.jsp`. The URL of the privacy page is :

`http://machine_name/webapp/wcs/stores/servlet/ContactView?parameter_list`

Sample store footer

The sample stores have a footer at the bottom of each page (`footer.jsp`) that includes links to the following:

- Home
- Shopping cart
- My account
- Contact us
- Privacy policy
- Help

Commands

`footer.jsp` uses the following commands:

- `StoreCatalogDisplay`
- `OrderItemDisplay`
- `LogonForm`
- `ContactView`
- `PrivacyView`
- `HelpView`

Beans

`footer.jsp` uses the following beans:

- `UserRegistrationDataBean`
- `ErrorDataBean`

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see *Sample store pages: common implementation techniques*.

Shopping cart

The **SHOPPING CART** link on the footer links to the `OrderItemDisplay` controller command, which returns the `OrderItemDisplayViewShiptoAssoc` view command. `OrderItemDisplayViewShiptoAssoc`

view command is registered with OrderItemDisplay.jsp in the database. OrderItemDisplay.jsp loads shoppingcart.jsp to display the Shopping cart page.

For more information on commands, see Commands and the "WebSphere Commerce Programmer's Guide."

My account

If you are a registered customer, clicking **My account** takes you to the My Account page (account.jsp). If you are not a registered customer, clicking **My account** takes you to the Register or Logon page (myaccount.jsp). This is accomplished using the following code:

```
if (userType.equalsIgnoreCase("G")){ %>
<font class="buttonson"><a
href="LogonForm?langId=<%=languageId%>&storeId=<%=storeId%>&catalogId=
<%=catalogId%>" style="color:
#CCCC99"><%=infashiontext.getString("MY_ACCOUNT")%></a></font></td>
<%} else {%>
<font class="buttonson"><a
href="LogonForm?langId=<%=languageId%>&storeId=<%=storeId%>&catalogId=
<%=catalogId%>&page=account" style="color:
#CCCC99"><%=infashiontext.getString("MY_ACCOUNT")%></a></font></td>
```

Contact us

Clicking Contact us calls the ContactView command, which loads the Contact Us page (contact.jsp).

Help Clicking Help calls the HelpView command, which loads the Help page (help.jsp).

Privacy policy

Clicking Privacy policy calls the PrivacyView command, which loads the Privacy Policy page (privacy.jsp).

Sample store header

The sample stores have a header at the top of each page (header.jsp) that includes links to the following:

- Shopping cart
- My account
- Contact us
- Help
- Men's
- Women's
- New arrivals

Commands

header.jsp uses the following commands:

- OrderItemDisplay
- LogonForm
- ContactView
- HelpView
- StoreCatalogDisplay
- CategoryDisplay

Beans

header.jsp uses the following beans:

- UserRegistrationDataBean
- CatalogDataBean
- CategoryDataBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

SHOPPING CART

The **SHOPPING CART** link on the header links to the OrderItemDisplay command, which returns the OrderItemDisplayViewShiptoAssoc view command. OrderItemDisplayViewShiptoAssoc view command is registered with OrderItemDisplay.jsp in the database. OrderItemDisplay.jsp loads shoppingcart.jsp to display the Shopping Cart page.

MY ACCOUNT

If you are a registered customer, clicking **MY ACCOUNT** takes you to the My Account page (account.jsp). If you are not a registered customer, clicking **MY ACCOUNT** takes you to the Register or Logon page (myaccount.jsp). This is accomplished using the following code:

```
if (userType.equalsIgnoreCase("G")){ %>
<font class="buttonson"><a
href="LogonForm?langId=<%=languageId%>&storeId=<%=storeId%>&catalogId=
<%=catalogId%>" style="color:
#CCCC99"><%=infashiontext.getString("MY_ACCOUNT")%></a></font></td>
<%> } else {%>
<font class="buttonson"><a
href="LogonForm?langId=<%=languageId%>&storeId=<%=storeId%>&catalogId=
<%=catalogId%>&page=account" style="color:
#CCCC99"><%=infashiontext.getString("MY_ACCOUNT")%></a></font></td>
```

CONTACT US

Clicking **CONTACT US** calls the ContactView command, which loads the Contact Us page (contact.jsp).

HELP Clicking **HELP** calls the HelpView command, which loads the Help page (help.jsp).

Top-level categories (Men's, Women's, New arrivals)

When a customer clicks one of the top-level categories in the header, the CategoryDisplay command is called. CategoryDisplay is registered with CategoryDisplay.jsp in the database. When the top parameter is set to Y, as in the following example, topcategory.jsp loads, displaying the appropriate category page, as follows:

```
<a href="CategoryDisplay?catalogId=<%=catalogId%>&storeId=<%=storeId%>&categoryId=<%=
=category.getCategoryId()%>&langId=<%=languageId%>&top=Y">
```

Sample store help page

The Help page (help.jsp) displays when a customer clicks **Help**.

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

A new view called HelpView is created in the VIEWREG table and associated with help.jsp. The URL of the privacy page is: http://machine_name/webapp/wcs/v5/stores/HelpView?parameter_list.

Sample store home page

The home page (`storecatalogdisplay.jsp`) acts as the storefront that brings customers into your store. The sample store home page displays all top-level categories in the store such as Men's and Women's, and promotes a few featured special items. For more information on the sample store home page, see the Home page use case.

Commands

`storecatalogdisplay.jsp` uses the following commands:

- `CategoryDisplay`
- `ProductDisplay`

Beans

`storecatalogdisplay.jsp` uses the following beans:

- `CatalogDataBean`
- `CategoryDataBean`
- `ProductDataBean`
- `EMarketingSpotBean` (WebFashion only)

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see *Sample store pages: common implementation techniques*.

The `storecatalogdisplay.jsp` is launched by the `index.jsp` page which provides an URL for invoking the homepage of the sample store. `index.jsp` calls the `parameters.jsp` file which contains the parameters needed to launch the store. `storecatalogdisplay.jsp` displays the following:

Top categories

Top categories are registered in the `CATTOGRP` table. `storecatalogdisplay.jsp` retrieves the top categories with the `getTopCategories()` method in the `CatalogDataBean`.

Featured specials

The featured specials contained in `InFashion` and `NewFashion` were created in the catalog rather than with WebSphere Commerce Accelerator. To create the promotions, a special top-level category was added to the catalog, with the identifier `HOMEPAGE_PROMO`. Products belonging to this category are then displayed as featured specials.

In `WebFashion` the targeted products on the home page are part of a campaign. The campaign is created using e-Marketing Spots and the WebSphere Commerce Accelerator. The e-Marketing Spot on the home page is called `StoreHomePage` and is used to create gender-based promotions. In order to activate the e-Marketing Spot, you must create a campaign using the WebSphere Commerce Accelerator.

Guest customers and registered customers who did not supply gender information will see the default products (`HOMEPAGE_PROMO`).

Note: Products in the `HomePromo` category or `New Arrival's` category appear in the special categories only. After the promotions are over, these products must be moved back to their regular category.

Sample store login pages

The sample store login pages allow registered customers to log in.

When registered customers click the link **Register now and receive advance notice of promotions!** on the side bar, the Register or Login page (`account.jsp`) displays. Customers then provide their e-mail addresses and passwords and click **Login**, and the My Account page (`myaccount.jsp`) displays. For more information, see the Logon use case.

The sample store login procedure uses the following JSP files:

- `account.jsp` (Register or Login page)
- `myaccount.jsp` (My Account page)
- `LoginForm.jsp` (Includes the parameters for the Logon command. Does not display to the customer.)
- `Logoff.jsp` (Includes the parameters for the Logoff command. Does not display to the customer.)
- `forgetpassword.jsp` (Forgot Your Password page)
- `ResetPasswordForm.jsp` (Includes the parameters for the reset password command. Does not display to the customer.)
- `password.jsp` (displays a message to customers that their password has been set to them)
- `ChangePasswordForm.jsp` (Change Password page)
- `ResetPasswordError.jsp` (Called if there is a problem resetting the password. Does not display to the customer.)

Commands

`account.jsp` uses the following commands

- Logon
- Logoff

`myaccount.jsp` uses the following commands:

- `UserRegistrationForm` (InFashion, WebFashion, NewFashion, and WebAuction)
- `AddressBookForm` (InFashion, WebFashion, NewFashion, and WebAuction)
- `InterestItemDisplay` (WebFashion, NewFashion, and WebAuction)
- `ProfileFormView` (WebFashion and WebAuction)
- `TrackOrderStatus` (WebFashion, NewFashion, and WebAuction)

`forgetpassword.jsp` uses the following commands:

- Logoff
- ResetPassword

`ChangePasswordForm.jsp` uses the following commands:

- ResetPassword

`forgetpassword_err.jsp` uses the following commands:

- ResetPassword
- Logoff

`password.jsp` uses the following commands:

- LogonForm

Beans

`forgetpassword.jsp` uses the following beans:

- `ErrorDataBean`

forgetpassword_err.jsp uses the following beans:

- ErrorDataBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

After customers have registered on the Register page, or typed their e-mail addresses and passwords on the Login page (account.jsp), the values are converted to lowercase using the following code:

```
function prepareSubmit(form)
{
form.<%=ECUserConstants.EC_UREG_LOGONID%>.value =
form.<%= ECUserConstants.EC_UREG_LOGONID%>.value.toLowerCase()
form.submit()
}
```

account.jsp also sets fields that the Logon command expects, for example:

```
<INPUT TYPE="hidden" NAME="URL" VALUE="LogonForm?page=account">
```

When customers click **Login** from the Register or Login page, the Logon command is called. Logon is registered with LoginForm.jsp in the database. LoginForm.jsp uses the page parameter to determine which page (myaccount.jsp or account.jsp) to load.

```
String state = request.getParameter("page");
.
.
.
if (state == null)
{
incfile = "/" + storeDir + "/myaccount.jsp";
}
else if (state.equals("account"))
{
incfile = "/" + storeDir + "/account.jsp";
}
}
```

If a correct e-mail address and password combination was entered, LoginForm.jsp loads the My Account page (myaccount.jsp). If an incorrect e-mail address and password combination was entered, LoginForm.jsp reloads the Register or Login page (account.jsp).

If customers forget their passwords, and click **Forget your password?** the Logoff command is called, with the state=forgetpassword parameter. The Logoff command is registered with Logoff.jsp in the database. Logoff.jsp checks the states of the parameter, as described below:

```
if (state == null)
{
String [] arrstate = (String []) request.getAttribute("state");
if (arrstate != null)
state = arrstate[0];
}
}
```

```

if (state == null || state.length() == 0)
{
incfile = "/" + storeDir + "/UserRegistrationForm.jsp";
}
else if (state.equals("forgetpassword"))
{
incfile = "/" + storeDir + "/forgetpassword.jsp";
}

```

The Forgot Your Password page (`forgetpassword.jsp`) is loaded if the state is equal to `forgetpassword`. When customers complete the fields on the page and click **Send me my password**, the `ResetPassword` command is called. The old password is set to expired in the database and the new password is e-mailed to customers. When customers login using the new password, they are forced to change their password and are brought to the Change Password page (`ChangePasswordForm.jsp`).

Note: If customers passwords are set to expire, they will automatically be brought to the Change Password page the next time they login.

Error handling

If customers provide an incorrect e-mail address or password, or do not complete the fields, an error message displays and the Logon command reloads the Register or Login page without setting the page parameter. If an incorrect password was entered, customers have to wait for a few seconds before logging in or the following error will be displayed:

Please wait a few seconds before attempting to login again.

Sample store privacy page

The Privacy page (`privacy.jsp`) displays when a customer clicks **Privacy Policy**.

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see *Sample store pages: common implementation techniques*.

A new view called `PrivacyView` is created in the `VIEWREG` table associated with `privacy.jsp`. The URL to view the privacy page is `http://machine_name/webapp/wcs/v5/stores/PrivacyView?parameter_list`.

Note: You must create your own privacy policy and include it in your online store. For more information refer to the IBM privacy site `http://www.ibm.com/privacy/`.

Sample store product pages

A product page features one particular product in an online store. It typically includes a description, a price, and an image, and if the product has attributes (for example, different sizes and colors) it allows customers to choose an attribute.

For more information on the product pages and how they work, see the *Display product page use case*.

Commands

`productdisplay.jsp` uses the following commands:

- `OrderItemAdd`

- InterestItemAdd (WebFashion and NewFashion only)

Beans

productdisplay.jsp uses the following beans:

- CategoryDataBean
- ProductDataBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

ProductDisplay.jsp is registered in the database (in DISPENTREL table) to display all products in the store. ProductDisplay.jsp displays the following:

- Description, image, attributes, and attribute values of the product
- Short description of the parent category
- Add to shop cart link
- Quantity text box (NewFashion only)
- Add to wish list link (WebFashion and NewFashion only)

Description, image, attributes, and attribute values of the product

The product description and image are displayed using ProductDataBean properties.

Product attributes are retrieved using the getAttributes() method for ProductDataBean. The values for each attribute are retrieved using the getDistinctAttributeValues() method in the AttributeAccessBean. If the ProductDataBean finds information in this column, the Product Display page displays the Hotmedia image instead of the full image.

Short description of the parent category

The ID of the parent category is supplied to product pages through the parent_category_rn parameter. A short description of the parent category is retrieved by CategoryDataBean. By default, CategoryDataBean gets the category ID from the categoryId parameter. In the following example, the parameter name is parent_category_rn, and the category ID is set explicitly:

```
String parentCategoryId = request.getParameter("parent_category_rn");
parentCategory = new CategoryDataBean ();
parentCategory.setCategoryId(parentCategoryId);
com.ibm.commerce.beans.DataBeanManager.activate(parentCategory, request);
```

Add to shopping cart

The **Add to shopping cart** link is implemented by creating a form that calls the OrderItemAdd command. In InFashion and WebFashion the quantity of products ordered is set to 1 by default, using a hidden field as follows:

```
<input type="hidden" name="quantity" value="1">
```

You can replace the hidden field with a text box so that customers can enter a different quantity.

In the NewFashion store, the quantity of products ordered is set to 1 by default using a text field as follows:

```
<input type="text" name="quantity" value="1",size="2">
```

The text field allows customers to enter a different quantity.

Add to shopping cart and Add to wish list

When a customer selects **Add to Shopping Cart** or **Add to wish list**, the following javascript is called.

```
<SCRIPT language="javascript">
    function Add2ShopCart(form){
        form.action='OrderItemAdd'
        form.URL.value='OrderItemDisplay'
        form.submit()
    }

    function Add2WishList(form){
        form.action='InterestItemAdd'
        form.URL.value='InterestItemDisplay'
        form.submit()
    }
</SCRIPT>
```

When the customer adds to the shopping cart, the OrderItemAdd Command is called. When the customer adds to the wish list, the InterestItemAdd command is called.

Note: You can use the WebSphere Commerce Accelerator to create products. When you create products, you must create a price for the product in order to view it in the sample store product pages. If you don't create a price, you must take away the getCalculatedContractPrice method for the ProductDataBean.

Sample store registration page

The sample store registration page allows customers to register at the sample store. When customers register, they must provide their first and last names and an e-mail address, and then create a password.

When customers click **Register now and buy!** the Register or Logon page displays. The customers then click **Register**, and the Registration page (register.jsp) displays.

When customers wish to update their registration information, they click **Change personal information** on the My Account page. The Change Personal Information page (edit_registration.jsp) displays.

Commands

register.jsp uses the following commands:

- UserRegistrationAdd
- PrivacyView

Beans

register.jsp uses the following beans:

- ErrorDataBean

edit_registration.jsp uses the following beans:

- DemographicsAccessBean
- UserRegistrationDataBean
- ErrorDataBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

Registering

The **Register** link from the Register or Login page (account.jsp) displays the registration form. WebSphere Commerce uses the UserRegistrationAdd command to create a new registration. If the customer is logged in, UserRegistrationAdd behaves like UserRegistrationUpdate. That is, a logged-in customer cannot register again and create a new account. Since the sample store allows the user to register multiple times and create multiple accounts, a logged-in customer must be logged off before re-registering. To achieve this, the Register link is automatically set to the Logoff command for logged-in customers and to the UserRegistrationForm view for guest customers. The Logoff command automatically calls the LogoffView task. The LogoffView task is registered with Logoff.jsp in the database. The Logoff.jsp checks the state URL parameter. If it is set to forgetpassword, forgetpassword.jsp loads; if not, UserRegistrationForm.jsp loads. The UserRegistrationForm view is associated with in the UserRegistrationForm.jspfile in the VIEWREG table. UserRegistrationForm.jspchecks the "new" URL parameter. If it is set to Y, the new registration form is displayed by including register.jsp. Otherwise, an update registration form is displayed by including edit_registration.jsp.

register.jsp contains a new registration form. The new registration form is submitted to the UserRegistrationAdd command. By default, the UserRegistrationAdd command expects several mandatory fields that are not required for the sample store. As a result, these fields are set up as hidden HTML fields and the values are set to "-", as in the following example:

```
<INPUT TYPE="hidden" NAME="personTitle" Value="-">
```

Note:In the NewFashion store, the name-value pair is not submitted. If the name-value pair is not submitted, the UserRegistrationAdd command will not check the URL parameter.

When customers register and fill out their profile, it can match one of the profiles in the WebSphere Commerce Accelerator. If the WebSphere Commerce Accelerator profile has a discount associated with it, the customers will receive that discount.

E-mail address as login

The sample stores require customers to type in their e-mail addresses as their login ID, but WebSphere Commerce requires a user ID as a login. As a solution, the sample store asks the customers to enter their e-mail addresses on the registration form. Then, before submitting the form, both e-mail and user ID fields are set to the value of the e-mail address using the following JavaScript:

```
function prepareSubmit(form)
{
form.<%=ECUserConstants.EC_ADDR_EMAIL1%>.value =
form.<%= ECUserConstants.EC_UREG_LOGONID%>.value.toLowerCase()
form.<%=ECUserConstants.EC_UREG_LOGONID%>.value =
form.<%= ECUserConstants.EC_UREG_LOGONID%>.value.toLowerCase()
form.submit()
}
```

Change personal information

Clicking **Change personal information** calls the UserRegistrationForm command, which is associated in the database with UserRegistrationForm.jsp. If the new parameter is not null, it will load register.jsp. Otherwise, it will load edit_registration.jsp.

The UserRegistrationUpdate command is used in edit_registration.jsp to update the user's registration information. If the customer does not complete the password or verify password fields, the system will provide the registration password, using the following code:

```
function prepareSubmit(form)
{
form.<%= ECUserConstants.EC_ADDR_EMAIL1
%>.value=form.<%=ECUserConstants.EC_UREG_LOGONID%>.value
form.<%=ECUserConstants.EC_UREG_LOGONID%>.value =
form.<%= ECUserConstants.EC_UREG_LOGONID%>.value.toLowerCase()
}
```

```

if (form.<%=ECUserConstants.EC_UREG_LOGONPASSWORD%>.value.length == 0)
{
form.<%=ECUserConstants.EC_UREG_LOGONPASSWORD%>.value = '<%=strPassword%>'
}
if (form.<%=ECUserConstants.EC_UREG_LOGONPASSWORDVERIFY%>.value.length == 0)
{
form.<%=ECUserConstants.EC_UREG_LOGONPASSWORDVERIFY%>.value = '<%=strPassword%>'
}
form.submit()
}

```

If a customer previously entered gender and age information, the gender and age fields will be prefilled. The `DemographicsAccessBean` extracts the gender and age information from the database.

Error Handling

On any error, the `UserRegistrationAdd` command calls `UserRegistrationErrorView`, which is registered with `UserRegistrationForm.jsp` in the database. However, the same error view is also invoked by `UserRegistrationUpdate`. To differentiate a new registration form from an edit form, the new registration form contains a hidden parameter called `new`. If that parameter is present, `UserRegistrationForm.jsp` includes `register.jsp`. Otherwise, it loads `edit_registration.jsp`.

`register.jsp` is used in both normal and error conditions. `ErrorDataBean` and error checking are used to determine under what condition `register.jsp` is being executed. If an error exists, `register.jsp` displays the error message.

Note: Password errors are caught through the `UserRegistrationAdd` and `UserRegistrationUpdate` commands, however they call `AuthenticationPolicyErrorView` when a password error occurs.

Sample store new arrivals page

The New Arrivals page (`newarrivals.jsp`) displays when the customer clicks **New Arrivals**.

Commands

`newarrivals.jsp` uses the following commands:

- `ProductDisplay`

Beans

`newarrivals.jsp` uses the following beans:

- `CategoryDataBean`
- `ProductDataBean`
- `CatalogDataBean`
- `EMarketingSpotBean`

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see `Sample store pages: common implementation techniques`.

The New Arrivals page (`newarrivals.jsp`) is meant to display a list of promoted items. The New Arrivals contained in `InFashion` were created in the catalog. As a result, a top-level category called `New Arrivals` was created and all promoted products were added to this category. Since `InFashion` displays the contents of this category differently than the other top level categories, the `newarrivals.jsp` file is registered in the

database (in DISPCGPREL table) to display this category. When the CategoryDisplay command is invoked with this category ID, newarrivals.jsp is loaded to display the products designated as new arrivals.

The newarrivals.jsp page uses the CategoryDataBean to display category information. For NewFashion and InFashion, the products in this page come from a special category with the identifier "Whats Hot." The list of promoted products is retrieved using the CategoryDataBean's getProducts() method.

In WebFashion, the New Arrivals page displays the Hot Sales campaign. The goal of this campaign is to sell the season's top trends. The Hot Sale campaign uses a suggestive selling initiative, that targets registered customers by suggesting items based on a customer's age. Registered customers are over 29, see one set of three products; customers under 29 see another set. Guest customers and registered customers who did not supply age information will see the default products. (Whats Hot)

The Hot Sales campaign is created using the WebSphere Commerce Accelerator and an e-Marketing Spot. The e-Marketing Spot is called NewArrivalsPage and is positioned on the New Arrivals page. To activate the e-Marketing Spot, you must set up a campaign in the WebSphere Commerce Accelerator.

NewFashion Sample store order confirmation page

After the customer completes the order in the fourth step of the sample store checkout process, the customer clicks **Order Now**. The Order confirmation page (confirmation.jsp) is displayed, allowing the customer to view the order number, sub-total, total tax, shipping charges, grand total, and price for the whole order. If two orders are processed, the information of both orders will be displayed on the confirmation page.

For more information, see the Checkout shopping cart use case.

Beans

confirmation.jsp uses the following beans:

- PayStatusPMDDataBean
- OrderDataBean

Implementation

The PayStatusPMDDataBean gets the order information from the system. The OrderDataBean takes the information, formats it, and displays it to the system.

WebFashion and NewFashion E-mail notification page

After a registered customer has completed an order by clicking **Order Now** (for more details see the Checkout shop cart use case), and the order is authorized by the Payment Manager, the system sends the customer an e-mail, OrderAuthorized.jsp, stating that the order has been accepted and payment is authorized.

In the NewFashion store, the following e-mail notices are sent:

- Order canceled e-mail (OrderCanceledNotification.jsp). The order cannot be completed since the credit card was declined.
- Shipping notification e-mail (ReleaseShipNotify.jsp). The order has been shipped to the shopper.
- Order submission e-mail (OrderReceived.jsp). After a registered customer has completed an order by clicking **Order Now**, the order submission e-mail is sent.

Beans

OrderAuthorized.jsp uses the following beans:

- StoreDataBean
- OrderDataBean
- OrderItemDataBean
- OrderBean
- StoreEntityDescriptionAccessBean
- AddressDataBean

Note: The above beans are used in the NewFashion store only.

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

The system retrieves the information about the customer's order using the OrderDataBean. Store information is retrieved using the StoreDataBean. StoreEntityDescriptionAccessBean gives the system the name of the store. AddressDataBean gives the system the shipping address.

Sample store order summary page

In the fourth step of the sample store checkout process, the Checkout 4, Order Summary page (`orderdisplaypending.jsp`), customers can review their detailed order information, including a description of the items bought, as well as quantity, unit price and total price, shipping address, and shipping costs. The customers must then complete the order by providing payment information and clicking **Order Now**. In the NewFashion store, the expected shipping date is displayed.

For more information, see the Checkout shop cart use case.

Commands

`orderdisplaypending.jsp` uses the following commands:

- OrderProcess
- PrivacyView
- ContactView
- MultiOrderProcess (NewFashion only) (MultiOrderProcess is associated with `MultiOrderProcess.jsp`)

Beans

`orderdisplaypending.jsp` uses the following beans:

- OrderDataBean
- OrderItemDataBean
- AddressDataBean
- ErrorDataBean
- ShippingModeDescriptionDataBean
- UsablePaymentTCListDataBean

Implementation details

NewFashion only

When customers click **Order Now**, the MultiOrderProcess command is called. MultiOrderProcess is a view command registered in the VIEWREG table, and is associated with `MultiOrderProcess.jsp`.

MultiOrderProcess.jsp executes OrderProcess multiple times depending on the numbers of orders in the order summary page. In NewFashion, the checkout flow only allows up to two orders in the order summary page.

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

For all sample stores

When customers initiate the checkout process by clicking **Checkout** in the Shopping Cart page, they move through a series of checkout pages, the fourth of which is Checkout 4. Order Summary page (orderdisplaypending.jsp). This page includes a form that allows customers to submit their credit card information. The UsablePaymentTCListDataBean is used to get the available credit card names from the Payment Manager, and the action for the form is set to OrderProcess. After the form is submitted, and if the order process is successful, the OrderOKView is called. The OrderOKView command is registered in the VIEWREG table in the database, and is associated with the confirmation.jsp that displays information to confirm the order. In the NewFashion store, if two orders are processed the information of both orders will be shown on the confirmation page.

If there is an error, the DoPaymentErrorView is called. DoPaymentErrorView is associated in the database with OrderDisplayPending.jsp. As a result, when there is an error, the Checkout 4. Order Summary page (orderdisplaypending.jsp) redisplay with an error message.

NewFashion Sample store check product availability page

In the third step, part 3a, of the sample store checkout process, as described in the Checkout shop cart use case, the customer can check product availability and see the expected shipping date of each item in the order (ProductAvailability.jsp).

Note: This page only shows up if some of the order items are unavailable.

If some of the items are not currently in stock, customers can choose a shipping preference allowing them to split the order, or backorder the items. Customers also have the option to remove an item from the order.

Commands

ProductAvailability.jsp uses the following commands:

- ProductDisplay
- OrderItemDelete
- OrderPrepare
- OrderItemMove
- OrderItemDisplay

Beans

ProductAvailability.jsp uses the following beans:

- OrderDataBean
- OrderItemDataBean
- CatalogEntryDataBean

Implementation

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

The Checkout 3a, Check Product Availability page (ProductAvailability.jsp) only shows up if some of the order items are not available. AllocationCheck.jsp checks to see if any items are not available. If all items are unavailable or all items are available, it forwards to OrderDisplayPending.jsp; otherwise it includes ProductAvailability.jsp. ProductAvailability.jsp allows customers to view the availability date for each item in the order. If some items in the order are not in stock, the customer selects a shipping option under the **Choose shipping preference** field for the order.

Customers can choose three shipping options:

1. Wait until the entire order is ready for shipping.
2. Ship the in-stock items now and ship the rest when they are available.
3. Ship the in-stock items now and leave the rest in the shopping cart.

If the customer chooses options 2 or 3, the OrderItemMove command splits the order into two separate orders.

WebFashion and NewFashion Sample store package display pages

A package display page features promotional groups of products in an online store. It typically includes a description, an image of the package, a list of components that make up the package and an image of each, the price for the package, and a list of attributes (size and color) if the products that compose the package have variations, and a list of values for each attribute (red and blue for color and Large and X-Large for size).

Customers can add the package to the shopping cart or wish list with a single click. The package displays as a single line item in the shopping cart, unlike a bundle, which displays a separate line for each item in the bundle. Products in a package cannot be purchased individually.

For more information on the sample store package display pages and how they work, see the Display package page use case.

Commands

PackageDisplay.jsp uses the following commands:

- OrderItemAdd
- InterestItemAdd

Beans

PackageDisplay.jsp uses the following beans:

- PackageDataBean
- CompositeProductDataBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

packagedisplay.jsp is registered in the database (in DISPENTREL table) to display all products in the store. packagedisplay.jsp displays the following:

- Description (including list of components that comprise the package), image, and price of the package
- Attributes and attribute values of the product comprising the package

- Add to shopping cart and Add to wish list links
- Quantity text box (NewFashion only)

Description (including list of components that comprise the package), image, and price of the package

The description, image, and price of the package are retrieved using the PackageDataBean.

Attributes and attribute values of the product comprising the package

The PackageDataBean retrieves ProductDataBeans to display information about each product in the package. For more information, see Sample store product pages.

In order for the package to display correctly, you must use the same parameter name for the attribute value for each product in the package. For example:

```
<SELECT NAME="attrValue" >
<!-- Display product attribute values !-->
<%
Object values[] = attribute.getDistinctAttributeValues();
for (int j = 0; j < values.length; j++)
{
%>
<option><%=values[j] %></option>
<% } %>
</select>
```

Add to shopping cart and Add to wish list

For more information, see Sample store product pages.

Quantity text box

Allows customers to specify the number of packages to add to the shop cart or wish list.

Sample store left navigation frame

The left navigation frame (`sidebar.jsp`) in the sample stores allows customers to select the language in which the store will display. The navigation frame also includes links to the Registration and Help pages.

For more information, see the Home page use case.

Commands

`sidebar.jsp` uses the following commands:

- StoreCatalogDisplay
- LogonForm
- HelpView

Beans

`sidebar.jsp` uses the following beans:

- StoreLanguageAccessBean
- LanguageDescriptionAccessBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see [Sample store pages: common implementation techniques](#).

When a customer selects country/region and language from the CHOOSE A COUNTRY/REGION drop-down list, and clicks **GO!**, store catalog pages are reloaded according to the `languageId`, using the following code:

```
String storeLangId = storeLang.getLanguageId();  
.br/>.br/><option value="<%= storeLangId %>" SELECTED><%=langDesc.getDescription()%></option>
```

JavaScript code is dynamically created in order to chain the `StoreCatalogDisplay` and `SetCurrencyPreference` commands together. This is done to link a language to a default currency. When the customer selects a language, they indirectly select the default currency for that language, and as a result the `OrderPrepare` command does not have to be called in the `shoppingcart.jsp` page. This is shown in the following code:

```
<select NAME="currency"> ... <option value="<%= (String)currencyId.elementAt(iElementNum) %>"  
SELECTED> <%= (String)currencyid.elementAt(iElementNum+1)%> </option>
```

The following is an example of a dynamically generated javascript that links to a language, based on the selected currency:

```
<SCRIPT language="javascript"> function ChangeLanguage(form) { if (form.currency[0].selected ==  
true) {  
form.URL.value = "StoreCatalogDisplay?storeId=10151&catalogId=10151&langId=-1"; } if  
(form.currency[1].selected == true) {  
form.URL.value = "StoreCatalogDisplay?storeId=10151&catalogId=10151&langId=-5"; } form.submit();  
} </SCRIPT>
```

NewFashion Sample store left navigation frame

The left navigation frame (`sidebar.jsp`) in the NewFashion store allows customers to select the language and currency in which the store will display, and search the catalog for items. The navigation frame also includes links to the Registration, Help, Advanced Search pages, and Live Chat with Customer Care.

Note: The **Live Chat with CustomerCare** link can only be seen if enabled in the store through Store Services.

For more information, see the [Home page use case](#).

Commands

`sidebar.jsp` uses the following commands:

- `StoreCatalogDisplay`
- `SetCurrencyPreference`
- `LogonForm`
- `HelpView`
- `CatalogSearchResultView`
- `AdvancedSearchView`

Beans

sidebar.jsp uses the following beans:

- SupportedLanguageAccessBean
- LanguageDescriptionAccessBean
- CurrencyDescriptionAccessBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see [Sample store pages: common implementation techniques](#).

When customers select a country/region and language from the CHOOSE A LANGUAGE drop-down list, and click **GO!**, store catalog pages are reloaded according to the languageId. The code below gets the store's supported languages and displays them in the drop-down box:

```
<jsp:useBean id="supportedLanguageDataBean"
class="com.ibm.commerce.common.objects.SupportedLanguageAccessBean" scope="page" />
<%
Enumeration enStoreLangList =
supportedLanguageDataBean.findByStore(new Integer(storeId));
while (enStoreLangList.hasMoreElements()) {
SupportedLanguageDataBean storeLang =
(SupportedLanguageDataBean) enStoreLangList.nextElement();
String storelangId = storeLang.getLanguageId();
//Get the display name of the language in the language
//currently selected by the shopper.
LanguageDescriptionDataBean langDesc =
new LanguageDescriptionDataBean();
langDesc.setInitKey_languageId(languageId);
langDesc.setInitKey_descriptionLanguageId(storelangId);

//If this language is currently selected, select it
//in the drop down list.
if (languageId.equals(storelangId))
{
%>
<option value="<%= storelangId %>" SELECTED><%=langDesc.getDescription()%></option>
```

When customers select a currency from the CHOOSE A CURRENCY drop-down list, and clicks **GO!**, store catalog pages are reloaded according to the supportedCurrencies. The code below gets the supported currencies and displays them in the drop-down box:

```
CurrencyManager cm = CurrencyManager.getInstance();
String [] supportedCurrencies = (String []) cm.getSupportedCurrencies(cmdcontext.getStore());
for (int i = 0; i < supportedCurrencies.length; ++i)
{
CurrencyDescriptionDataBean currDesc = new CurrencyDescriptionDataBean();
currDesc.setInitKey_languageId(languageId);
currDesc.setInitKey_currencyCode(supportedCurrencies[i]);
String currency = (String) cmdcontext.getCurrency();
// pre-select the appropriate value in the in the drop down list.
if (currency.equals(supportedCurrencies[i]))
{
%>
<OPTION Value="<%=supportedCurrencies[i]%>" SELECTED><%=currDesc.getDescription()%></OPTION>
<%
```

When customers enter a keyword in the Search field and clicks **GO**, the `CatalogSearchResultView` command submits the search criteria and the `resultList.jsp` page is displayed with the results of the search.

If customers click **Live Chat with Customer Care**, a pop-up window is displayed on the screen, and customers can chat online in real-time with a customer service representative. This link will only be displayed if this feature is enabled in Store Services. The **Live Chat with Customer Assistance** link is contained within the body of a pair of custom tags (the `<flow:ifEnabled feature="customerCare">` tag and `</flow:ifEnabled>` tag), and can be enabled or disabled based on what option is selected in Store Services. As long as the custom tag remains in place, Store Services can be used to automatically switch between a site that has collaboration support and one that does not, without having to modify the JavaServer page. To permanently enable or disable collaboration support in the page, the custom tags and the **Live Chat with Customer Care** link can be removed from the JavaServer page by clicking **Apply Permanently**, in the Store Services GUI. It is not recommended to manually remove or alter the custom tags or the code it surrounds. Instead, use the **Apply Permanently** button in Store Services.

Note: The custom tag that surrounds the **Live Chat with Customer Care** link cannot be copied into the JavaServer pages of other stores. These tags are only intended to work in the store that originally contained them. For more information on collaboration, see the related links below.

Sample store select billing address page

In the first step of the sample store checkout process, the Checkout 1, Select Billing Address page (`billingaddress.jsp`), the customer can select an existing address as the billing address, or create a new address to be used as the billing address.

For more information, see the Checkout shop cart use case.

Commands

`billingaddress.jsp` uses the following commands:

- `AddressForm`
- `OrderItemDisplay`
- `OrderCopy`
- `AddBillAddressView`

Beans

`billingaddress.jsp` uses the following beans:

- `OrderDataBean`
- `AddressAccessBean`
- `OrderItemAccessBean`
- `ErrorDataBean`

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see [Sample store pages: common implementation techniques](#).

When a customer initiates the checkout process by clicking **Checkout** in the Shopping Cart page, they move through a series of checkout pages, the first of which is the Checkout 1. Add billing address page (`billingaddress.jsp`). `billingaddress.jsp` checks whether the customer has existing addresses in the address book. If addresses currently exist in the address book, they will be displayed, allowing the customer to select one as the billing address.

The customer can also create a new address, by clicking **Create new address**. Clicking **Create new address** calls the AddressForm command, which is associated in the database with AddressForm.jsp. AddressForm.jsp calls address.jsp, which loads the Add Address page. address.jsp checks for the page parameter to determine the next page to load. If the page value is set to billingaddress the URL value in the AddressAdd form will be set to OrderItemDisplay. OrderItemDisplay calls the billingaddress.jsp, which takes the customer back to the Checkout 1. Select Billing Address page when the customer clicks **Submit**. checks the page parameter to determine which address form to load.

If there aren't any existing addresses in the address book, the Add billing address form will be displayed, prompting the customer to enter a new address. The Add billing address form in this case is also generated by billingaddress.jsp. The action for this HTML form is set to AddBillAddressView, which is registered in the VIEWREG table. AddBillAddressView is associated with AddBillAddress.jsp. When the form with AddBillAddressView is submitted, AddBillAddress.jsp is called.

AddBillAddress.jsp executes the following commands:

- AddressAdd
- OrderCopy

After AddressAdd is executed, AddBillAddress.jsp uses the address ID returned by AddressAdd as part of the input to the OrderCopy command. Then, the OrderCopy assigns the address ID to the billing address of the current order, and OrderItemDisplay.jsp is called. The value of the page parameter is set to newshipaddress, so OrderItemDisplay.jsp calls shipaddress.jsp.

Note: During registration, WebSphere Commerce requires an address to be created. Since the sample store doesn't ask for an address during customer registration, some of the required fields, such as address1, are set to unused. When checking for addresses, **billingaddress.jsp** checks whether the value of address1 is unused. If it is, the address is not displayed.

NewFashion Sample store select billing address page

In the first step of the sample store checkout process, the Checkout 1, Select Billing Address page (billingaddress.jsp), customers can select an existing address as the billing address, or create a new address to be used as the billing address.

For more information, see the Checkout shop cart use case.

Commands

billingaddress.jsp uses the following commands:

- AddressForm
- OrderItemDisplay
- OrderCopy
- AddressAdd

Beans

billingaddress.jsp uses the following beans:

- OrderDataBean
- AddressDataBean
- OrderItemDataBean
- ErrorDataBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see *Sample store pages: common implementation techniques*.

When customers initiate the checkout process by clicking **Checkout** in the Shopping Cart page, they move through a series of checkout pages, the first of which is the Checkout 1, Add Billing Address page (`billingaddress.jsp`). `billingaddress.jsp` checks whether the customers have existing addresses in the address book. If addresses currently exist in the address book, they will be displayed, allowing customers to select one as the billing address.

The `OrderItemDisplay` command is used to determine which page will be loaded next. The `OrderItemDisplay` returns the `OrderItemDisplay.jsp` in the database. `OrderItemDisplay.jsp` includes different JSP files based on the page parameter. If a value for page is `shipmethod`, the third page, Checkout 3, Select Shipping Method (`shipping.jsp`), is loaded.

Customers can also create a new address, by clicking **Create new address**. Clicking **Create new address** calls the `AddressForm` command, which is associated in the database with `AddressForm.jsp`. `AddressForm.jsp` calls `address.jsp`, which loads the Add Address page. The Add Address page will take the customer back to the Checkout 1, Add billing address page when they click **Submit**.

If there are no existing addresses in the address book, the Add billing address form will be displayed, prompting the customer to enter a new address. The Add billing address form in this case is also generated by `billingaddress.jsp`.

After `AddressAdd` executes and calls the `OrderCopy` command, `OrderCopy` assigns the address ID to the billing address of the current order, and calls the `OrderItemDisplay` command which calls the `OrderItemDisplay.jsp`. The value of the page parameter is set to `newshipaddress`, so `OrderItemDisplay.jsp` calls `shipaddress.jsp`.

Note: In the *InFashion* and *WebFashion* stores, the `AddBillAddress.jsp` is used to call the `AddAddress` and `OrderCopy` commands. In the *NewFashion* store the `AddBillAddress.jsp` has been eliminated by chaining the `AddAddress` and `OrderCopy` commands using the `URL` parameter. This method is easier to implement as it does not require an extra JSP, however it has a longer system execution path because of extra the re-direction.

Sample store shopping cart

Customers can view and edit the items they have selected in the shopping cart (`shoppingcart.jsp`), as described in the Display shopping cart use case.

Commands

`shoppingcart.jsp` uses the following commands:

- `OrderItemUpdate`
- `OrderItemDelete`
- `StoreCatalogDisplay`
- `QuickCheckoutView`

Beans

`shoppingcart.jsp` uses the following beans:

- `OrderDataBean`

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see *Sample store pages: common implementation techniques*.

When a customer clicks **SHOPPING CART** in the header or footer, the `OrderItemDisplay` command is called, which returns the `OrderItemDisplayViewShiptoAssoc` view command.

`OrderItemDisplayViewShiptoAssoc` view command is registered with `OrderItemDisplay.jsp` in the database. `OrderItemDisplay.jsp` includes different JavaServer Page files based on the page parameter. If a value for page is not supplied, the Shopping Cart page, (`shoppingcart.jsp`), is loaded.

Note: `OrderItemDisplay.jsp` runs `OrderPrepare`, which recalculates the order total and converts the default currency to the currency in use by the customer. Single currency stores do not need to run `OrderPrepare`.

The Shopping Cart page includes an **Update Totals** button and a **Checkout** button. Clicking **Update Totals** updates the quantity of items ordered, and then redisplay the Shopping Cart page. Clicking **Checkout** updates the quantity of items ordered and then displays the Checkout 1: Select Billing Address page.

Both the **Checkout** and **Update Totals** buttons use the same HTML form. However, when a customer clicks **Checkout**, before the form is submitted, JavaScript is used to add an extra page parameter with value set to `billingaddress`.

The command used to complete the quick checkout option is `QuickCheckoutView`, which is a view command registered in `VIEWREG` and associated with `QuickCheckout.jsp`. `QuickCheckout.jsp` runs the following server commands:

- `OrderItemUpdate` (update shipping address)
- `OrderItemUpdate` (update shipping mode)
- `OrderCopy` (update billing address and payment information)
- `OrderPrepare`

`QuickCheckout.jsp` retrieves the billing and shipping addresses, shipping method, and payment information from the customer's quick checkout profile using the `OrderAccessBean`. Then, it assigns this information to the order specified in the `orderId` and executes the `OrderPrepare` command.

After the quick checkout process is complete, the command forwards to a view specified in the URL. `WebFashion` specifies `QuickCheckoutSummaryView` as the URL for the `QuickCheckout` command. So, when quick checkout is complete, the Quick checkout summary page displays.

Error handling

If the store does not have a fulfillment center attached to it, or if the product is not in stock, the `OrderItemAdd/OrderItemUpdate` command calls the `ResolveFulfillmentCenterErrorView`, which is registered with `shoppingcart.jsp` in the database. If customers enter an invalid character in the quantity field, the `InvalidInputErrorView` is called. `InvalidInputErrorView` is also registered with `shoppingcart.jsp` in the database.

Since errors can be caused by both the `OrderItemUpdate` command, and the `OrderItemAdd` command, `shoppingcart.jsp` checks which command caused the problem and displays an error accordingly. If the last command is `OrderItemUpdate`, the Shopping cart page is redisplayed with an error message. Otherwise, a separate error page is displayed with an error message. This is shown in the code below:

```
String lastCmdName = cmdcontext.getCommandName().trim();
```

shoppingcart.jsp is used in both normal and error conditions. The ErrorDataBean and error checking determine under what condition shoppingcart.jsp is being displayed. If an error exists, shoppingcart.jsp displays an appropriate error message.

If the QuickCheckout.jsp(QuickCheckoutView) cannot find the quick checkout profile, it calls QuickCheckoutError.jsp to show a error message. As a result, quickcheckouterrorview.jsp displays.

quickcheckouterrorview.jsp checks if the customer is registered. If the customer is registered, the system displays a message prompting the customer to create a quick checkout profile. If not, the system prompts the customer to register first, and then create a quick checkout profile.

NewFashion Sample store shopping cart

Customers can view and edit the items they have selected in the shopping cart (shoppingcart.jsp), as described in the Display shopping cart use case. Up to two orders can be present in the shopping cart.

Commands

shoppingcart.jsp uses the following commands:

- OrderItemUpdate
- ProductDisplay
- StoreCatalogDisplay
- OrderItemMove

Beans

shoppingcart.jsp uses the following beans:

- OrderItemDataBean
- ErrorDataBean
- CatalogEntryDataBean
- OrderDataBean
- FormattedMonetaryAmountDataBean
- OrderAccessBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

The shopping cart appears to always display as one order, however there can be multiple orders in the shopping cart that have been split up. This occurs if the customer has gone to the checkout flow previously, and split the order. When the customer clicks the **Checkout**, OrderItemMove is called to move all the orders into one order. OrderItemMove uses the deleteIfEmpty parameter to delete all the empty orders left over after the move.

When a customer clicks **SHOPPING CART** in the header or footer, OrderItemDisplay command is called, which is registered with OrderItemDisplay.jsp in the database. OrderItemDisplay.jsp includes different JSP files based on the page parameter. If a value for page is not supplied, the Shopping Cart page, (shoppingcart.jsp), is loaded.

The OrderItemUpdate command updates the quantity for each item in the shopping cart. The FormattedMonetaryAmountDataBean formats the total price and amounts. The OrderDataBean gets the shopping cart items for each order. Each shopping cart item is an OrderItemBean.

The Shopping Cart page includes an **Update Totals** button and a **Checkout** button. Clicking **Update Totals** updates the quantity of items ordered, and then redisplay the Shopping Cart page. Clicking **Checkout** updates the quantity of items ordered, and then displays the Checkout 1, Select Billing Address page.

Note: Both the **Checkout** and **Update Totals** buttons use the same HTML form.

Error handling

If the store does not have a fulfillment center attached to it, the `ErrorDataBean` will provide error information. `ErrorDataBean` also provides information if the customer enters an invalid quantity, such as a non-numeric character.

`shoppingcart.jsp` is used in both normal and error conditions. The `ErrorDataBean` and error checking determine under what condition `shoppingcart.jsp` is being displayed. If an error exists, `shoppingcart.jsp` displays an appropriate error message. For more information on error handling refer to the related concepts below.

Sample store shipping method page

In the third step of the sample store checkout process, as described in the Checkout shop cart use case, the customer must select a shipping method (`shipping.jsp`).

Commands

`shipping.jsp` uses the following commands:

- `AddShipModeView`
- `OrderItemDisplay`

Beans

`shipping.jsp` uses the following beans:

- `OrderBean`
- `OrderItemAccessBean`
- `ShipModeAccessBean`

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see [Sample store pages: common implementation techniques](#).

When a customer initiates the checkout process by clicking **Checkout** in the Shopping cart page, they move through a series of checkout pages. The `OrderItemDisplay` command is used to determine which page will be loaded next. The `OrderItemDisplay` returns the `OrderItemDisplay.jsp` in the database. `OrderItemDisplay.jsp` includes different JSP files based on the page parameter. If a value for page is `shipmethod`, the third page, Checkout 3.Select shipping method (`shipping.jsp`), is loaded.

The Checkout 3.Select Shipping Method page (`shipping.jsp`) includes a form that allows customers to select a shipping method. The action for the form is set to `AddShipModeView`, which is registered in the `VIEWREG` database table to associate with `AddShipMode.jsp`. When the form with `AddShipModeView` is submitted, the `AddShipMode.jsp` is called.

`AddShipMode.jsp` executes the following commands:

- `OrderItemUpdate`

- OrderPrepare.

The OrderItemUpdate command updates the order item with the selected shipping method. Then, the OrderPrepare command is called to preprocess the order. Afterwards, OrderDisplay is called to display the next page in the checkout process. If the status parameter is set to P, the Checkout 4: Order Summary (OrderDisplayPending.jsp) displays next.

The Checkout 3. Select Shipping Method page (shipping.jsp) displays the cost structure and approximate delivery time for each shipping method. This information is stored in the SHPMODEDSC table in the following fields:

- DESCRIPTION stores the description of the shipping method.
- SHPMODEDSC.FIELD1 stores the cost structure description.
- SHPMODEDSC.FIELD2 stores the approximate delivery time.

If you change the shipping charge in the database, make sure you also change the description in SHPMODEDSC table so that the updated values display on this page.

Note: SHPMODEDSC.FIELD1 and SHPMODEDSC.FIELD2 were created using the custom fields in the SHPMODEDESC table.

NewFashion Sample store select shipping method page

In the third step of the sample store checkout process, as described in the Checkout shop cart use case, the customer must select a shipping method (shipping.jsp). The customer can select a separate shipping method for each item in the order.

Commands

shipping.jsp uses the following commands:

- OrderItemUpdate
- Product Display
- OrderItemDisplay
- AllocationCheck

Beans

shipping.jsp uses the following beans:

- OrderDataBean
- ShippingDataBean
- OrderItemDataBean
- ShippingModeDescriptionDataBean
- CatalogEntryDataBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

When customers initiate the checkout process by clicking **Checkout** in the Shopping Cart page, they move through a series of checkout pages. The OrderItemDisplay command is used to determine which page will be loaded next. The OrderItemDisplay returns the OrderItemDisplay.jsp in the database. OrderItemDisplay.jsp includes different JSP files based on the page parameter. If a value for page is shipmethod, the third page, Checkout 3, Select Shipping Method (shipping.jsp), is loaded.

The Checkout 3, Select Shipping Method page (`shipping.jsp`) allows customers to select a shipping method for each item in the order. Customers select a shipping method under the **Shipping method** field for each item.

The `OrderItemUpdate` command updates the order item with the selected shipping method. The `OrderItemDisplay` command is called to display the next page in the checkout process when customers click **Next**. If some of the items are not available the Checkout 3a, Product Availability (`ProductAvailability.jsp`) page displays next. The Product Availability page is directed to `AllocationCheck.jsp` which determines which page to go to, depending on whether there is available inventory.

The Checkout 3, Select Shipping Method page (`shipping.jsp`) displays the cost structure for each shipping method. This information is stored in the `SHPMODEDESC` table in the following fields:

- `DESCRIPTION` stores the description of the shipping method.
- `SHPMODEDESC.FIELD1` stores the cost structure description.

If you change the shipping charge in the database, make sure you also change the description in `SHPMODEDESC` table so that the updated values display on this page.

Note: `SHPMODEDESC.FIELD1` is a custom field in the `SHPMODEDESC` table.

Sample store select shipping address page

In the second step of the sample store checkout process, the Checkout 2. Select shipping address page (`shipaddress.jsp`), the customer can select an existing address as the shipping address, edit that address, or create a new address to be used as the shipping address.

For more information, see the Checkout shopping cart use case.

Commands

`shipaddress.jsp` uses the following commands:

- `OrderItemUpdate`
- `AddressForm`
- `OrderItemDisplay`

Beans

`shipaddress.jsp` uses the following beans:

- `OrderBean`
- `AddressAccessBean`
- `OrderItemDataBean`

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see [Sample store pages: common implementation techniques](#).

When a customer initiates the checkout process by clicking **Checkout** in the Shopping Cart page, they move through a series of checkout pages, the second of which is the Checkout 2. Select shipping address page (`shipaddress.jsp`). `shipaddress.jsp` displays all existing addresses, allowing the customer to select one as the shipping address.

The customer can also create a new address, by clicking **Create new address**. Clicking **Create new address** calls the `AddressForm` command, which is associated in the database with `AddressForm.jsp`. `AddressForm.jsp` calls `address.jsp`, which loads the Add Address page. `address.jsp` checks for the page parameter to determine the next page to load. If the page value is set to `shipaddress` the URL value in the `AddressAdd` form will be set to `OrderItemDisplay`. `OrderItemDisplay` calls the `shipaddress.jsp`, which takes the customer back to the Checkout 2. Select shipping address page when the customer clicks **Submit**.

Note: During registration, WebSphere Commerce requires an address to be created. Since the sample store doesn't ask for an address during customer registration, some of the required fields, such as `address1`, are set to "-". When checking for addresses, `billingaddress.jsp` checks whether the value of `address1` is "-". If it is, the address is not displayed.

NewFashion Sample store select shipping address page

In the second step of the sample store checkout process, the Checkout 2, Select Shipping Address page (`shipaddress.jsp`), customers can select an existing address as the shipping address, edit that address, or create a new address to be used as the shipping address. The existing address is selected by choosing a nickname under the **Ship to** field. Each item in the order can have a separate shipping address.

For more information, see the Checkout shop cart use case.

Commands

`shipaddress.jsp` uses the following commands:

- `OrderItemUpdate`
- `AddressForm`
- `AddressBookForm`
- `Product Display`
- `OrderItemDisplay`

Beans

`shipaddress.jsp` uses the following beans:

- `OrderBean`
- `AddressDataBean`
- `OrderItemDataBean`

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see [Sample store pages: common implementation techniques](#).

When customers initiate the checkout process by clicking **Checkout** in the Shopping Cart page, they move through a series of checkout pages, the second of which is the Checkout 2, Select Shipping Address page (`shipaddress.jsp`). `shipaddress.jsp` displays all items in order, and nicknames linked to existing addresses, allowing the customer to select a separate nickname as the shipping address for each item.

The `OrderItemDisplay` command is used to determine which page will be loaded next. The `OrderItemDisplay` returns the `OrderItemDisplay.jsp` in the database. `OrderItemDisplay.jsp` includes different JSP files based on the page parameter. If a value for page is `shipmethod`, the third page, Checkout 3, Select Shipping Method (`shipping.jsp`), is loaded.

Customers can create new addresses, by clicking **Create new address**. Clicking **Create new address** calls the AddressForm command, which is associated in the database with AddressForm.jsp. AddressForm.jsp calls address.jsp, which loads the Add Address page. The Add Address page will take the customer back to the Select shipping address page when they click **Submit**.

Customers can also edit an address by clicking **Edit address book**. Clicking **Edit address book** calls the AddressBookForm command, which is associated in the database with AddressBookForm.jsp. AddressBookForm.jsp calls AddressBook.jsp, which loads the Address Book page. All the addresses associated with that customer are displayed. Customers can then click **Edit** under the chosen address. Clicking **Edit** displays the Update address page where changes can be made. The customers are taken back to the Address Book page when they click on **Submit**. The Address Book page takes customers back to the Checkout 2, Select Shipping Address page when customers click **Return to checkout**.

Note: During registration, WebSphere Commerce requires an address to be created. Since the sample store does not ask for an address during customer registration, the fields of the address created during registration are empty. When checking for addresses, shipaddress.jsp checks whether LogonID equals NickName. If it is, this means the address is a self address i.e. address created during registration, and the address is not displayed. If it is, the address is not displayed.

Sample store quick checkout summary page

When a customer clicks **Quick checkout** from the Shopping cart, the Quick Checkout: Order summary page (quickcheckoutsummary.jsp) displays, with the order details and pre-filled shipping, billing and payment information. The customer must then complete the order by clicking **Order Now**.

For more information, see the Quick checkout use case.

Commands

quickcheckoutsummary.jsp uses the following commands:

- OrderProcess
- OrderItemDisplay
- PrivacyView
- ContactView

Beans

quickcheckoutsummary.jsp uses the following beans:

- OrderBean
- AddressDataBean
- ErrorDataBean
- ProfileCassetteAccountDataBean
- OrderPaymentInfoAccessBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

The quick checkout summary page behaves the same way as does the Sample store order summary page, with the following differences:

- Payment information fields are pre-filled with information from quick checkout profile.
- Payment information is retrieved using the OrderPaymentInfoAccessBean.

WebFashion and NewFashion sample store view orders pages

After registered customers have placed an order, they can view the status of that order at any time. To view an order, the customer clicks **My Account**, and then from the My Account page, clicks **View orders**. The Order Status page (`trackorderstatus.jsp`) displays with the list of orders that the customer has placed. For more information about a particular order, customers click on the order. The Order Details page (`orderdetail.jsp`) displays.

Note: Temporary, pending, CSR edit, quick order profile, private requisition list, shareable requisition list, no inventory orders, and cancelled orders, cannot be viewed.

For more information, see the View orders use case.

Beans

`trackorderstatus.jsp` uses the following beans:

- OrderDataBean
- PayStatusListPMDDataBean
- OrderAccessBean
- PriceDataBean

`orderdetail.jsp` uses the following beans:

- OrderDataBean
- OrderItemDataBean

Implementation details

Note: For information on implementation techniques common to all sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

When a customer clicks **View orders**, the Order Status page (`trackorderstatus.jsp`) displays with a list of orders the customer has placed. The OrderDataBean retrieves all the orders placed by the customer, using the following code to loop through all orders in the list:

```
<jsp:useBean id="orderABFinder" class="com.ibm.commerce.order.objects.OrderDataBean"
scope="page" />
Enumeration ordersABList = orderABFinder.findByMemberForUpdate(userId);
```

The OrderDataBean displays the following order information:

- Order number
- Order date
- Total amount
- Payment status

The OrderDataBean retrieves the payment status by calling the `.getStatus()` method. The `.getStatus()` method returns a one-character status string which represents the payment status as seen in the table below.

Status	Brief Description	Meaning
P	Pending	The customer can modify the Order.
I	Submitted	The customer cannot modify the Order.

W	Pending approval	The CheckOrderApproval task command indicated that some of the order items are not approved.
N	Approval denied	The CheckOrderApproval task command indicated approval has been denied for some order items.
M	Pending payment authorization	Waiting for payment authorization.
A	Payment authorization requires review	Payment authorization has encountered an unusual circumstance, such as an address verification warning. The payment authorization should be reviewed and accepted, or the order cancelled, using the Order Management user interface. If the authorization is accepted, the user interface will change the order status to either 'B' or 'C' as appropriate.
B	Back-ordered	Payment authorization may need to be re-done, since the Order amount may change as a result of back-order inventory allocation.
C	Payment authorization complete	Payment authorization has completed. All order items are allocated from existing inventory. The order amount will not change.
E	CSR Edit	A customer service representative is working with the order.
R	Released	All order items have been released for fulfillment.
S	Shipped	All order items have been shipped.
D	Deposited	Payment has been captured.
L	No inventory	Inventory is not available for one or more order items.
T	Temporary	Used by the Order Management user interface to temporarily back-up an order.
Q	Quick order profile	The order holds default order information that can be copied to quickly create new orders.
F	Ready for remote fulfillment	The order is ready to be sent to a remote system for fulfillment. This status is used by the MQAdapter feature.
G	Pending remote fulfillment	The order has been sent to a remote system for fulfillment. This status is used by the MQAdapter feature.
Y	Private requisition list	The order is a private requisition list.
Z	Shareable requisition list	The order is a shareable requisition list.
X	Cancelled	The order has been cancelled.

Note: The Payment Manager checks to see if the payment has been declined. If the payment status is not declined, a message is displayed, depending on what the `.getStatus()` method returned.

```
if (payStatusBean.getPaymentState(sOrderId).equalsIgnoreCase("PAYMENT_VOID") ||
    payStatusBean.getPaymentState(sOrderId).equalsIgnoreCase("PAYMENT_DECLINED")) {
```

Once customers have retrieved their list of orders, they can click on a specific order for more information (`orderdetail.jsp`). The `OrderDataBean` retrieves all the order items for the customer, and the `OrderItemDataBean` retrieves the details about each item. The `OrderItemDataBean` retrieves the following details:

- Price of the order
- Contents of the order
- Estimated or actual shipping date of the order
- Tracking number for each item

Each item may consist of multiple pieces shipped in different boxes and may have more than one tracking number. Multiple items may share the same tracking numbers.

The `OrderDataBean` retrieves the estimated or actual shipping date of the order as seen in the following code:

```
orderDate = orderABFinder.findByOrderForUpdate(new Long(orderId)).getActualShipDate();
```

WebFashion and NewFashion Sample store wish list pages

The sample store wish list pages allow registered customers to add items to their wish list (interest list), then view and edit the list (`interestitemdisplay.jsp`), adding products to their shopping cart as desired. Customers can also send their wish list to family and friends via e-mail (`sendwishlistmsg.jsp`). In the NewFashion store, customers can send a personal message with their wish list.

Note: NewFashion allows both guest shoppers and registered customers to add items to the wish list. WebFashion allows only registered customers to add items to the wish list.

When customers receive a wish list via e-mail, they see the wish list page (`sharedwishlist.jsp`). This page is identical to `interestitemdisplay.jsp`, without the **Send wish list** and **Remove item** buttons. Customers can select an item to purchase and add it to their shopping cart.

For more information, see the View wish list use case and the Add item to the wish list use case.

The sample store wish list procedures uses the following JSP files:

- `interestitemdisplay.jsp` (Wish list page)
- `sharedwishlist.jsp`
- `sendwishlistmsg.jsp`

Commands

`interestitemdisplay.jsp` uses the following commands:

- `OrderItemAdd`
- `SendWishListMsg`

`sharedwishlist.jsp` uses the following commands:

- `OrderItemAdd`

sendwishlistmsg.jsp uses the following commands:

- SendMsgCmd

Beans

interestitemdisplay.jsp uses the following beans:

- InterestItemListDataBean
- InterestItemDataBean
- CatalogEntryAccessBean
- UserRegistrationDataBean

sharedwishlist.jsp uses the following beans:

- InterestItemListDataBean
- InterestItemDataBean
- CatalogEntryAccessBean

sendwishlistmsg.jsp uses the following beans:

- UserRegistrationDataBean (WebFashion only)

Implementation details

Note: For information on implementation techniques common to all Sample store pages, including multicultural information, see Sample store pages: common implementation techniques.

Add to wish list

When customers click **Add to wish list**, the InterestItemAdd command is called. For more information, see Sample store product pages.

Before adding the item to the wish list, the system uses the UserRegistrationDataBean to check that the customer is registered with the store. If the customer is not registered, the JSP includes registerfirst.jsp, which then displays a message prompting the customer to register and try again.

View wish list

When customers click **View wish list**, the Wish list page (interestitemdisplay.jsp) displays with a list of the wish list contents. InterestItemListDataBean and InterestItemDataBean retrieve the information about contents of the wish list.

Add to shopping cart and Remove items

From the Wish list page (interestitemdisplay.jsp), customers can choose to add items to their shopping cart or delete items from the wish list. When a customer clicks **Add selected items to shopping cart**, the OrderItemAdd command is called. When a customer clicks **Remove item**, the InterestItemDelete command is called.

Send wish list

When a customer completes the **Name** and **E-mail address** fields and clicks **Send wish list**, the SendWishListMsg command is called. SendWishListMsg is registered in the VIEWREG table to display sendwishlistmsg.jsp.

Before the SendWishListMsg command can run, you must create an instance. WebFashion and NewFashion create an instance using the following:

```
SendMsgCmd sendMsgCmd = (SendMsgCmd) CommandFactory.createCommand(cmdEntry);
```

For example:

```
CommandRegistryEntry cmdEntry =  
CommandFactory.locateCommandEntry("com.ibm.commerce.messaging.commands.SendMsgCmd", new
```

```
Integer(storeId));  
SendMsgCmd sendMsgCmd = (SendMsgCmd) CommandFactory.createCommand(cmdEntry);
```

You can set the parameters of the “sendMsgCmd” task command to send the message immediately:

```
sendMsgCmd.sendImmediate();
```

Note: WebFashion has a recipient’s E-mail address field only. NewFashion has a recipient’s E-mail field, sender’s Name field, and Personal message field. The recipient’s E-mail field, and sender’s Name fields are mandatory.

Error handling

If the customer has not selected any items in the wish list and clicks **Add selected items to shopping cart**, an error message displays. The following code performs this action:

```
function checkForm(form)  
{  
  var hasItem  
  var i, e  
  hasItem = false  
  for (i = 0; i < form.elements.length; i++)  
  {  
    e = form.elements[i]  
    if (e.type == "checkbox")  
    {  
      if (e.checked)  
      {  
        hasItem = true  
        break  
      }  
    }  
  }  
  if (hasItem)  
  form.submit()  
  else  
  alert("<%=infashiontext.getString("SELECTITEMS")%>")  
}  
</script>
```

Sample store use cases

Use cases detail the flow of each user interaction in a store, for example, registration or checkout. A set of use cases, that detail the interactions for the sample stores InFashion, WebFashion, and NewFashion is provided in the online help. These use cases can help you to more thoroughly understand the flow of the sample stores, and can be used as a guide for creating use cases for your own store.

Notes:

1. WebFashion is the Professional Edition version of the InFashion store. WebFashion only pages and features are denoted by

▶ Professional

.

2. Pages listed below not denoted by

▶ Professional

may contain some Professional Edition features.

The following use cases are provided:

- Home page use case
- Registration use case
- Logon use case
- Manage Personal Account use case
- Change personal information use case
- View product category use case
- Display product page use case
-

► Professional

Display package page use case

•

► Professional

Display bundle page use case

- Display shopping cart use case
- Checkout shop cart use case

•

► Professional

Quick checkout use case

- Edit an address use case
- Add new address use case

•

► Professional

View orders use case

•

► Professional

Add item to the wish list use case

•

► Professional

View wish list use case

•

► Professional

Create quick checkout profile use case

For more information on how the pages work together, see the sample store shopping flow diagram.

Chapter 5. Sample store use cases

Add new address use case (Business Edition)

Business

Customers can add a new address to their address book.

Actor

Customer

Main flow

The customer clicks **Add a new address**. The system displays a page with the following fields:

- Nickname (NewFashion only)
- First name
- Last name
- Street address (composed of two text boxes)
- City
- State or Province
- ZIP code or Postal Code
- Country/Region
- Phone number

The customer types the information in the fields and clicks **Submit**. The system adds a new address to the address book (E1).

Alternative flow

None

Exception flows

E1: Missing Mandatory Field

If any of the following fields are missing, system issues an error message.

- First name
- Last name
- Street address
- City
- State or Province
- ZIP code or Postal Code
- Country/Region

If the nickname entered already exists in the customer's address book, the system will report an error message.

The use case resumes from the beginning.

WebFashion and NewFashion add item to the wish list use case

Customers can view and edit the products they have selected to purchase through the shopping cart.

In the WebFashion store a wish list (or interest list) allows registered customers to add products to a list that they would like to order in the future. In the NewFashion store, customers do not have to be registered to add an item to the wish list. A wish list can be e-mailed to family or friends, who can then purchase the items as gifts for the customer. A wish list differs from a shopping cart in that customers plan to purchase the products in the shopping cart during the current shopping session.

Actor

Registered customer

Main flow

The customer views a product, package, or bundle (for more information see Display product page use case, Display bundle page use case, and Display package page use case) and then clicks **Add to wish list**. The system adds the product, bundle, or package to the wish list (E1) and then displays the wish list page, as described in View wish list use case.

Exception flows

E1: Guest shopper tries to add an item to the wish list

If a guest shopper tries to add an item to the wish list, the system displays the following message:
To save items to your wish list, register with WebFashion, and then try again.

In the NewFashion store, guest shoppers can add to the wish list without registering.

View product category use case

Catalog group pages display a list of subcategories and products. Catalog group pages help customers navigate through the products. The beginning catalog group pages lead to broad areas, and subsequent catalog group pages narrow the search.

There are usually three types of catalog group pages:

- Pages that display subcategories within a parent category
- Pages that display products within a subcategory
- Pages that display both subcategories and products

Actor

Customer

Main flow

Top-level product categories are listed in the Home page, for example, Men's Fashion, Women's Fashion and New arrivals. When the customer clicks on Men's Fashion or Women's Fashion, the system retrieves the subcategories within the selected category from the database, and displays the corresponding category information. When the customer clicks on the New arrivals link, new products are displayed. The following information is displayed for each category and product:

- A thumbnail image (products only)
- The name of each category or product
- A brief description to identify product.
- The price of the product.

The category pages for NewFashion and WebFashion also include a featured special, in the form of a package or bundle. The featured special includes an image and a brief description of the package or bundle. InFashion's featured special is a single product only.

The customer clicks the thumbnail image or the name. Then, the system displays the corresponding page (A1, A2, A3, A4).

Alternate flow

A1: Customer selects category

When the customer selects a category, the system displays another category page by resuming this use case from the beginning.

A2: Customer selects product

When the customer selects a product, the product page displays as described in Display product page use case.

A3: Customer selects bundle (WebFashion and NewFashion only)

When the customer selects a bundle, the bundle page is displayed as described in Display bundle page use case.

A4 Customer selects package (WebFashion and NewFashion only)

When the customer selects a package, the package page is displayed as described in Display package page use case.

Exception flows

None

Checkout shopping cart use case

Customers check out and submit an order when they pay for products listed in their shopping carts.

Actor

Customer

Main flow

The customer clicks **Checkout** to initiate the use case.

The system displays the Billing Address page. If the customer has at least one address in the address book, the system prompts the customer to select one of the addresses from the address book as the billing address. The customer can select an address or click **Create new address**. If the customer clicks **Create new address**, a new address is added using the Add new address use case. The customer then selects an address. The system sets up the selected address as the billing address for the order. If the customer does not have any addresses in the address book, A2 Enter Billing Address is performed.

The system displays the Shipping Address page. A list of addresses in the address book displays. If the customer finds the appropriate shipping address in the list, the customer selects the address. Otherwise, the customer clicks **Create New Address** and adds a new address using the Add new address use case. The system sets up the selected address as the shipping address of the order.

The system displays a list of shipping methods appropriate for the shipping address. The following information is displayed for each shipping method:

- Short description
- Cost of shipping:

- Fixed shipping cost per order
- Shipping cost per item ordered
- Approximate delivery time (InFashion and WebFashion only)

The customer selects the shipping method. The system sets the selected method as the shipping method of the order.

Note: In the NewFashion store, the shipping address and shipping method are specified for each order item.

For NewFashion only

In the NewFashion sample store, the system displays the availability of each item in the order, as well as several choices for shipping the items based on their availability. For each product in the order the system displays:

- Quantity
- Short product description
- Attribute values (for example, size: x = large; color: blue)
- Expected availability date based on future inventory
- Remove button (to remove the item from the order)

If the quantity requested for a specific item is not in stock, the system splits the order into two orders, one containing the available order items, and the other containing the non-available portions of the inventory. The customer chooses one of three Shipping preferences:

- Option 1, wait until the entire order is ready before shipping. The system shows the expected availability date of the whole order.
- Option 2, ship the available items now and ship the rest later.
- Option 3, ship the available items now and leave the rest in the shopping cart for purchase at a later date.

The customer then selects the shipping preference for the order.

Note: If all the items are available the customer will not see the availability date for each item.

For more information on the NewFashion inventory subsystem see Inventory subsystem.

For InFashion, WebFashion, and NewFashion

As well as displaying the product information, the system also displays the details of the order information. For each item in the order the system displays:

- Short product description
- Attribute values (for example, size: x = large; color: blue)
- Quantity
- Unit price
- Total price
- The system also displays:
 - Billing address
 - Shipping address
 - Subtotal (total cost of all products ordered)
 - Discounts, if any
 - Total tax, if any (multiple taxes should be displayed separately)

- Duties, if any
- Shipping cost including the shipping method
- Grand total (what the customer will be charged)
- Estimated shipping date (displayed at the top) (NewFashion only)

Note: In the NewFashion store, shipping address and shipping method are shown for each item in the order.

The system prompts the user to enter the credit card information. The following information is requested:

- Credit card type (for example, Visa(R) or MasterCard(R))
- Card number
- Expiration month
- Expiration year

The customer enters credit card information and clicks **Order Now**. The system stores the payment information (E1) and displays a confirmation page with the following information about the order:

- Order number
- Sub total (total cost of all products ordered)
- Total tax
- Shipping
- Discounts, if any
- Grand total

The customer prints out the page for future reference.

Alternative flow

A1: Add billing address

The customer is prompted to add a new address. The customer enters an address using the Add new address use case. The system sets up the new address as the billing address for the order. The use case continues.

Exception flows

E1: Invalid credit card number

The system checks the validity of the credit card number. If the check fails, the system displays an error message stating:

Invalid credit card number

and prompts the customer to re-enter the information. The use case continues.

E2: Credit card expired

If the expiry date of the credit card is earlier than the current date, the system displays an error message. The use case resumes from the beginning.

Note: If an item is not available the Product Availability Page will be displayed to the customer.

WebFashion and NewFashion display bundle page use case

The bundle page displays more details about the products or items that compose a bundle.

Actor

Customer

Main flow

The customer selects a bundle in the category page. The system then retrieves the bundle information from the database and then displays a page with detailed information about the selected bundle. The bundle page displays the following information:

- Short description
- Long description
- Full-sized image of the bundle
- Images of the products that compose the bundle
- Price for each product in the bundle
- A list of attributes (size and color) if the products that compose the bundle have variations, and a list of values for each attribute (red and blue for color and Large and X-Large for size)

The customer then selects the appropriate values for each attribute and clicks **Add to shopping cart**. In the NewFashion store, the customer can specify the quantity to add to the shopping cart. The system adds the bundle to the shopping cart (E-1), and displays the Shopping Cart page as described in the Display shopping cart use case.

Note:The default quantity is 1. Customers can modify the quantity after the bundle has been added to the shopping cart.

The customer also has the option of clicking **Add to wish list**. If the customer does so, the system adds the bundle to the wish list and displays the wish list as described in the Add item to the wish list use case.

Alternative flow

None

Exception flows

None

WebFashion and NewFashion Display package page use case

The package page displays more details about the products or items that compose the package.

Actor

Customer

Main flow

The customer selects a package in the category page. The system retrieves the package information from the database and then displays a page with detailed information about the selected package. The package page displays the following information:

- Short description

- Long description
- Images of the products that compose the package
- Price for the package
- A list of attributes (size and color) if the products that compose the package have variations, and a list of values for each attribute (red and blue for color and Large and X-Large for size)

The customer then selects the appropriate values for each attribute and clicks **Add to shopping cart**. The system adds the package to the shopping cart (E-1), and displays the Shopping Cart page as described in the Display shopping cart use case. In the NewFashion store, the customer can specify the quantity to add to the shopping cart. The default quantity is one.

Note: Since all items in a package cannot be sold separately, a package counts as one item in the shopping cart.

The customer also has the option of clicking **Add to wish list**. If the customer does so, the system adds the package to the wish list and displays the wish list as described in the Add item to the wish list use case.

Alternative flow

None

Exception flows

E1: Attribute is not selected

For products with a variety of attributes, the customer must select a value for each attribute. If the customer does not select a value, the system selects a default value and adds the package to the shopping cart or wish list.

Edit an address use case (Business Edition)

Business

Customers can edit addresses in their address book.

Actor

Customer

Main flow

The customer selects an address from address book to edit. The system retrieves and displays the selected address details:

- Nickname (NewFashion only)
- First name
- Last name
- Street address
- City
- State or Province
- ZIP code or Postal Code
- Country or Region
- Phone number

In NewFashion, the nickname entered by the customer is displayed at the top of the page.

The customer makes the desired changes and clicks **Submit**. The system updates the address (E1).

Alternate flow

None

Exception flows

E1: The system notifies the customer of missing mandatory information, and requests the missing information which can include the following:

- First name
- Last name
- Street address
- City
- State or Province
- ZIP code or Postal Code
- Country or Region

If the nickname already exists in the address book, an error message will be displayed.

Customer enters missing mandatory information.

Home page use case

The home page acts as the storefront, and brings customers into your store.

Actor

Customer

Main flow

A customer enters the store URL in the Web browser. The system then displays the home page.

The home page includes the following:

- A navigation bar with links to the following pages:
 - Home page
 - Contact information page
 - Security and privacy page
 - Shopping cart page
 - Registration page
 - Account page
 - Help page
- Links to primary or top-level categories. For each category the following information is displayed:
 - An image
 - Links to targeted products
 - If customers have specified their gender when registering, products targeted to the customers gender display. If not, a generic set of products displays.
 - For each product the following information is displayed:

- An image
- A short description

The customer clicks the image. Then, the system displays the corresponding page (A1, A2).

Alternative flow

A1: Customer selects category

When the customer selects a category, the system displays a category page as described in the View product category use case.

A2: Customer selects product

When the customer selects a product, the product page displays as described in Display product page use case.

Logon use case

The logon process allows registered shoppers to access their accounts by typing their user name and password.

Actor

Customer

Main flow

The customer selects **My Account**. The system then displays a page with the following fields:

- E-mail address
- Password

Customers enter the appropriate information in the above fields, and select **Login**. The system ensures that customers e-mail addresses and passwords are correct and then allows the customer into their accounts. If customers forget their passwords, they select **Forgot your Password** and alternative flow A1 is performed.

Alternative flow

A1: Forget your Password

If customers forget their password, they select **Forgot your Password**. The system then displays a page prompting the customers to enter their e-mail addresses. The customers type their e-mail addresses and click **Send me my password**. The system then sends the password to the e-mail address (E1).

Exception flows

E1: No matching e-mail address in the system

If the system cannot locate a customer with the matching e-mail address, the following error message displays: Can not locate a customer with that e-mail. The use case aborts.

If the customer attempts to login with the same user name and fails 6 times, the customer will be locked out. A message will be displayed noting that the account is locked, and that the customer must contact a store representative to re-activate it.

Manage Personal Account use case

Customers manage their account through the account pages.

Actor

Customer

Main flow

The customer selects **My Account**. The system then displays the My Account page with the following options:

- Change personal information
- Edit my address book
- Create or update Quick Checkout Profile (WebFashion only)
- View wish list
- View orders

If the customer selects **Change personal information**, alternate flow A1: Change e-mail and password is performed.

If the customer selects **Edit my address book**, alternate flow A2: Edit address book is performed.

If the customer selects **Create or update profile** (the Quick Checkout profile), the Quick Checkout Profile page displays, as described in the Create quick checkout profile use case.

If the customer selects **View wish list**, the Wish list page displays, as described in the View wish list use case.

If the customer selects **View orders**, the Order Status page displays, as described in the View orders use case.

Alternative flow

A1: Edit e-mail and password

The system prompts the customer to modify the e-mail address and password using the procedure described in the Change personal information use case. The use case resumes from the beginning.

A2: Edit address book

The system displays a page that lists all the address already added to the address book. Next to each address are two buttons: **Edit** and **Delete**. Below the list of addresses is an **Add a new address** button.

If the customer clicks **Delete**, the system deletes the corresponding address from the database and then displays a message indicating that the address was deleted properly. The alternate flow resumes from the beginning.

If the customer clicks **Edit**, the system prompts the customer to edit the address using the procedure described in the Edit Address use case. The system then displays a message confirming that the address was updated properly, and the alternate flow resumes from the beginning.

If the customer clicks **Add a new address**, the system prompts the customer to enter a new address using the procedure described in the Add new address use case. This alternate flow resumes from the beginning.

Exception flows

None.

WebFashion and NewFashion View orders use case

Customers can track and view the status of their order.

Actor

Customer

Main flow

The customer clicks **My Account** and then **View orders**. The system displays the Order Status page, which lists all of the orders placed by the customer. If the customer has not placed any orders, E1: No orders found is performed.

The following information is displayed for each order:

- Order number (linked to the Order Details page)
- Order date
- Order status
- Total amount

To find out more information about the order, the customer can click the order number. The system displays the Order Details page. The Order Details page displays the following information about each product in the order:

- Quantity
- Short product description
- The attributes (size and color) and values for each attribute (red and blue for color, and Large and X-Large for size)
- Unit price
- Total price
- Shipping date for the whole order (NewFashion only)
- Tracking ID numbers (NewFashion only)

Exception flows

E1: No orders found

If the system cannot find any orders associated with this customer, the system displays an error message stating:

There are no orders found

The customer can then click **Return to Home Page** or **My Account**.

Display product page use case

The product page displays more details about a product.

Actor

Customer

Main flow

The customer selects a product in the category page. The system then retrieves the product information from the database and displays a page with detailed information about the selected product. The Product page displays the following information:

- Product name
- Detailed description
- Price
- Full-sized image of the product
- A list of attributes (size and color) if the product has variations, and a list of values for each attribute (red and blue for color and Large and X-Large for size)

The customer then selects the appropriate values for each attribute and clicks **Add to shopping cart**. The system adds the selected item in the shop cart (E1), and displays the shopping cart page as described in the Display shopping cart use case. In the NewFashion store, the customer can specify the quantity to add to the shopping cart. The default quantity is 1.

In WebFashion and NewFashion, the customer has the option of clicking **Add to wish list**. If the customer does this, the system adds the package to the wish list and displays the wish list as described in the Add item to the wish list use case.

Alternative flow

None

Exception flows

None

Change personal information use case

Customers can change personal information such as e-mail address and password.

Actor

Customer

Main flow

The customer clicks **Change Personal Information** (E1). The system displays the Change personal information page, which includes the following fields:

- **First name**, pre-filled with the current first name
- **Last name**, pre-filled with the current last name
- **E-mail address**, pre-filled with the current e-mail address
- **Password**
- **Verify password**
- **Age**, pre-filled with the age (WebFashion only)
- **Gender**, pre-filled with the gender (WebFashion only)
- **Preferred language** (NewFashion only)
- **Preferred currency** (NewFashion only)

The customer alters the fields as needed. The password field can be left empty. If the customer chooses to leave the password field empty, the system does not modify the current password. The customer then

selects **Submit**, and the system updates the e-mail address and password (E2). If the password field is empty, the system leaves the previous password unchanged.

Alternative flow

None

Exception flows

The password entered must be at least 6 characters, contain at least one digit, contain at least 1 letter, and must not contain the same character 5 times in total, or 4 consecutive times, or an error message is displayed. An error message is also displayed if the password is the same as the e-mail address, or if the customer decides to change the password, but the password is the same as the previous one. For more information see the related tasks below.

E1: Customer not logged in

Customers must be logged in to change their e-mail addresses and passwords. If customers are not logged in, the system prompts the customers to log in using the procedure described in the Logon use case. This use case then resumes from the beginning.

E2: E-mail already exists for a different customer

The entered e-mail address cannot match the e-mail address of any other existing customer. If the system finds a match it prompts the customer to select a different e-mail address. The use case resumes from the beginning.

E3: Can not verify password

The password entered must match the password specified as the verify password. Otherwise, the system displays the following error message: Password does not match the verify password, please try again.

The use case resumes from the beginning.

Registration use case

The registration process allows customers to enter personal information in the database.

Actor

Customer

Main flow

The customer selects **Register**. The system then displays a page with the following fields:

- E-mail
- Password
- Verify password
- First name
- Last name
- Age (optional) (WebFashion only)
- Gender (optional) (WebFashion only)
- Preferred language (NewFashion only)
- Preferred currency (NewFashion only)

The customer enters the appropriate information in the above fields, and selects **Submit**. The system creates a new customer in the system and saves the customer's information (E1, E2, E3).

The system prompts customers to manage their accounts following the process in the Manage Personal Account use case.

Alternative flow

None.

Exception flows

E1: E-mail address already exists

If the e-mail address already exists in the system, the system displays an error message asking the user to enter another e-mail address. The use case resumes from beginning.

E2: Missing mandatory fields

If any of the following fields (E-mail, Password, Verify password, First name, Last name) are not completed, the system issues an error message. The use case resumes from beginning.

E3: Invalid password

If the password does not match the verification password, the system issues a warning.

Note:The password entered must be at least six characters, contain at least one digit and one letter, and must not contain the same character five times in total, or four consecutive times, or an error message is displayed. An error message is also displayed if the password is the same as the e-mail address.

Display shopping cart use case

Customers can view and edit the products they have selected to purchase through the shopping cart.

Actor

Customer

Main flow

The customer clicks **Shopping cart**. The system then displays a page showing the contents of the shopping cart. A list of products in the cart is displayed (E1). For each such product the following information is displayed.

- Short product description
- Attribute name and attribute value pairs of the item, such as size and large.
- Quantity
- Unit price
- Total price

The system also displays the total cost of the order before tax and shipping.

If the customer clicks **Update totals**, A1: Update totals is performed.

If the customer clicks **Remove item**, A2: Remove an item from the cart is performed.

If the customer clicks **Checkout**, A3: Update cart and checkout is performed.

If the customer clicks **Return to shopping**, the system displays the same page from where the customer came to the shopping cart page. The use case terminates.

If the customer clicks Quick checkout the system displays the Quick checkout: Order summary page as described in the Quick checkout use case

Sub flows

A1: Update totals

To update the quantity of any item, the customer types the new quantity in the text box, then clicks **Update totals**. The system updates the quantity of the items in the shop cart. If the new quantity of an item is 0, the item is removed from the cart.

A2: Remove an item from the cart

To delete an item from the cart, the customer clicks **Remove item** for the item. The system removes the item from the cart.

A3: Update cart and checkout

The system updates the quantity of the items in the shop cart and initiates the checkout process. Refer to Checkout shopping cart use case, Checkout shopping cart use case.

Exception flows

E1: Empty shopping cart

If there are no items in the shopping cart, the system displays the following message: Your cart is empty.

If an invalid number is entered into the quantity field such as a character, the shopping cart page will be re-displayed with an error message.

WebFashion and NewFashion View wish list use case

The customers can view and edit the products they have added to their wish list (or interest list).

Actor

Customer

Main flow

Customers click **My Account**, and then **View wish list**. The system then displays a page showing the contents of the wish list. A list of products in the wish list is displayed (E1). For each product, bundle, or package in the wish list, the following information is displayed.

- Short product description
- Attribute and value of the item, for example, size and color values
- Unit price
- **Remove** link

The system also displays the following items in the page:

- **Return to shopping** button
- **Add selected items to shopping cart** button.
- **Send wish list** button
- **E-mail address** field (To: / recipient)
- **E-mail message** field (NewFashion only)
- **From:/sender Name**(NewFashion only)
- **E-mail address**(NewFashion only)

To delete an item from the wish list, the customer, clicks **Remove**. The system completes A1, Remove item.

To send the wish list to friends and family via e-mail, customers enter the recipients E-mail addresses in the **E-mail address** field and click **Send wish list**. The system completes A2, Send wish list. In the NewFashion store, customers can send an e-mail message using the **E-mail message** field. Customers need to specify their name and can optionally specify their e-mail address. Registered customers have the **Name** field pre-filled.

To add items in the wish list to the shopping cart, customers select each item to add to the shopping cart and click **Add selected items to shopping cart**. The system completes A3, Add to cart.

Alternative flow

A1: Remove item

To delete an item from the cart, customers click **Remove** for the item. The system removes the item from the wish list. The use case resumes from the beginning.

A2: Send wish list

To send the wish list to friends and family via e-mail, customers enter the recipients e-mail addresses in the **E-mail address** field. To send the wish list to more than one friend, each e-mail address must be separated with a comma, for example: info@infashion.com, wcs@infashion.com. The customers then click **Send wish list**.

The system composes an e-mail message containing the following information:

- instructions on how to purchase the wish list items
- a link to the wish list, which contains the following information:
 - short description of each product, package or bundle in the wish list
 - a link to the product, bundle, or package display page
 - price
 - **Add selected items to shopping cart** button

In the NewFashion store, customers can send a personalized e-mail message along with the system message by entering the message in the **E-mail** message field.

A3: Add to cart

To add an item in the wish list to the shopping cart, customers select each item to add, then click **Add selected items to shopping cart**. The system adds all selected items to the shopping cart, then displays the shopping cart, as described in the Display shopping cart use case.

Exception flows

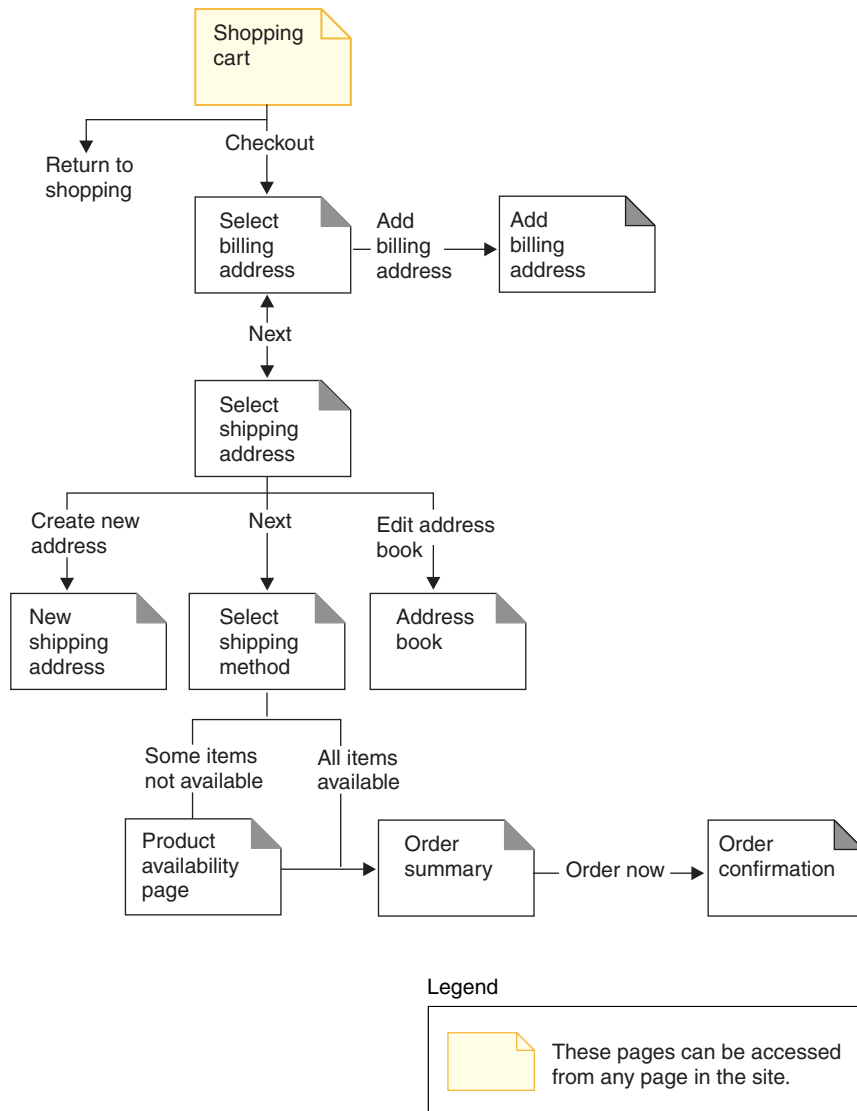
E1: Empty wish list

If there are no items in the wish list, the system displays the following message: Empty wish list. The use case terminates.

Chapter 6. NewFashion shopping flows

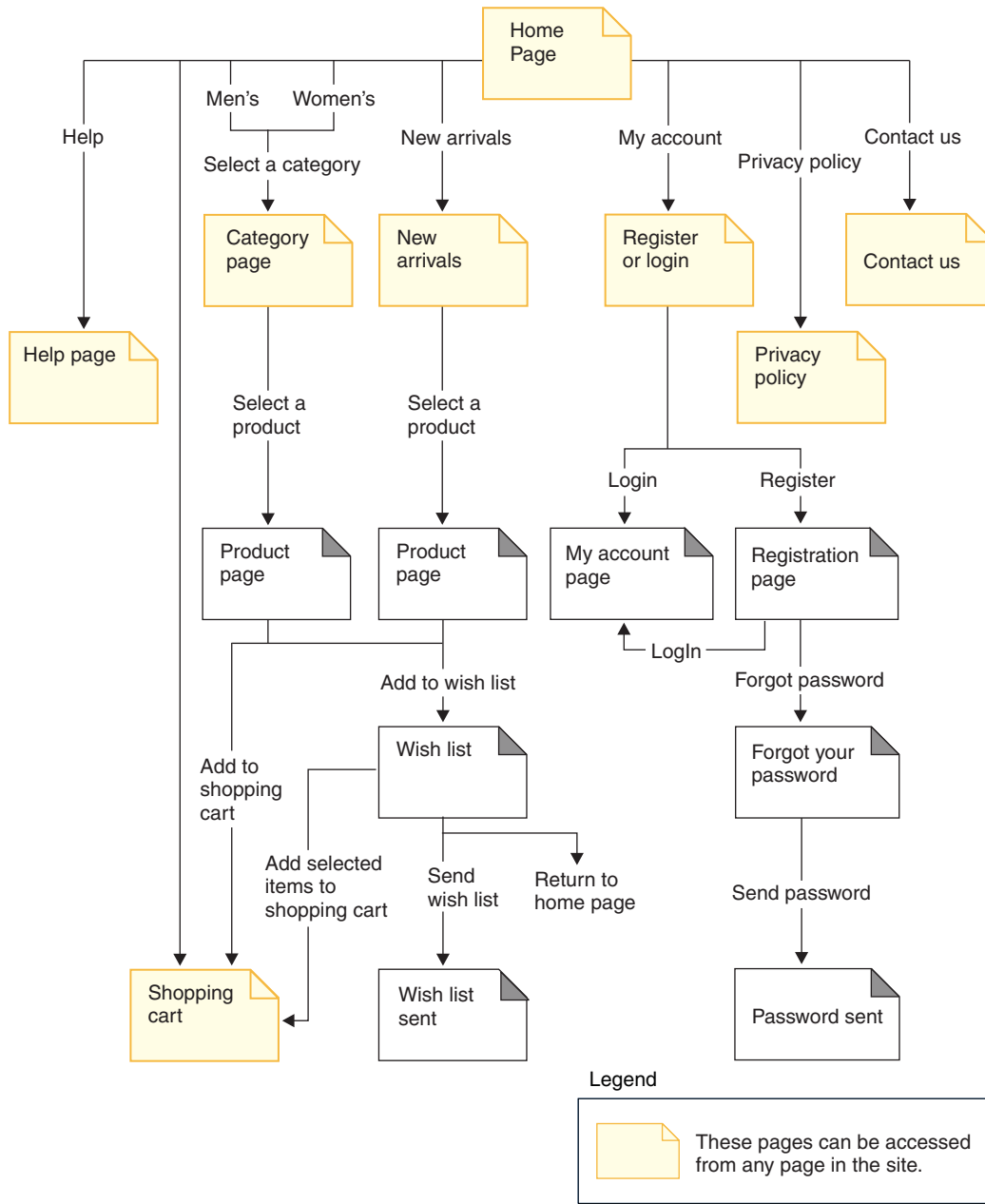
NewFashion Sample store shopping cart flow

The following diagram details the NewFashion Shopping cart flow. For more information on the complete shopping flow, see the related references below.



NewFashion Sample store shopping flow

The following diagram details the top pages in the NewFashion shopping flow. For more information on the shopping cart and my account flows, see the related references section below.



Notices

Any reference to an IBM licensed program in this document is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Director of Licensing
Intellectual Property & Licensing
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independent created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd. Laboratory
8200 Warden Avenue
Markham, Ontario
L6G 1C7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

This document may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

This document may contain information about other companies' products, including references to such companies' Internet sites. IBM has no responsibility for the accuracy, completeness, or use of such information.

This product is based on the SET protocol.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Trademarks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries or both:

AIX	CICS	DB2
DB2 Extenders	Encina	HotMedia
IBM	iSeries	MQSeries

SecureWay
400

VisualAge

WebSphere

Blaze Advisor is a trademark of HNC Software, Inc. in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Lotus and Domino are trademarks of Lotus Development Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Oracle is a registered trademark of Oracle Corporation.

SET and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC. For further information see <http://www.setco.org/aboutmark.html>.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.