IBM® WebSphere Commerce

Store Developer's Guide

Version 5.4

IBM® WebSphere Commerce

Store Developer's Guide

Version 5.4

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

Second Edition (May 2002)

This edition applies to the following products:
- IBM® WebSphere® Commerce Business Edition for Windows NT® and Windows® 2000, Version 5.4
- IBM WebSphere Commerce Business Edition for AIX®, Version 5.4
- IBM WebSphere Commerce Business Edition for Solaris, Version 5.4
- IBM WebSphere Commerce Business Edition for Linux, Version 5.4
- IBM WebSphere Commerce Studio, Business Developer Edition for Windows NT and Windows 2000, Version 5.4
- IBM WebSphere Commerce Professional Edition for Windows NT and Windows 2000, Version 5.4
- IBM WebSphere Commerce Professional Edition for AIX, Version 5.4
- IBM WebSphere Commerce Professional Edition for Solaris, Version 5.4
- IBM WebSphere Commerce Professional Edition for Linux, Version 5.4
- IBM WebSphere Commerce Studio, Professional Developer Edition for Windows NT and Windows 2000, Version 5.4

and to all subsequent releases and modifications of the above listed products, until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

IBM welcomes your comments. You can send your comments by any one of the following methods:
1. Electronically to the network ID listed below. Be sure to include your entire network address if you would like a reply.

   `Internet: torrcf@ca.ibm.com`
2. By mail to the following address:

   IBM Canada Ltd. Laboratory
   B3/KB7/8200/MKM
   8200 Warden Avenue
   Markham, Ontario, Canada L6G 1C7

Contents

Before you begin

The *IBM WebSphere Commerce Store Developer's Guide* provides information about the WebSphere Commerce store architecture and the store development process. In particular, it provides details on the following topics:

- Store development options
- The store archive
- Store development tools
- Developing your storefront
- Developing your store data
- Store data architecture
- Store data information model
- Adding access control to your store
- Packaging your store
- Publishing your store
- Adding WebSphere Commerce features to your store

Conventions used in this book

This book uses the following highlighting conventions:

Boldface type indicates commands or graphical user interface (GUI) controls such as names of fields, buttons, or menu choices.

`Monospaced type` indicates examples of text you enter exactly as shown, as well as directory paths.

Italic type is used for emphasis and variables for which you substitute your own values.

This icon marks a Tip — additional information that can help you complete a task.

NT indicates information specific to Windows NT.

2000 indicates information specific to Windows 2000.

AIX indicates information specific to AIX.

Solaris indicates information specific to the Solaris Operating Environment.

400 indicates information specific to the IBM eserver iSeries™ 400® (formerly called AS/400®).

Linux indicates information specific to Linux.

Business indicates information specific to WebSphere Commerce Business Edition.

Professional indicates information specific to WebSphere Commerce Professional Edition.

Where to find new information

This book may be updated in the future. Check the following WebSphere Commerce Web site for updates:

Business `http://www.ibm.com/software/webservers/commerce/wc_be/lit-tech-general.html`

Professional `http://www.ibm.com/software/webservers/commerce/wc_pe/lit-tech-general.html`

Updates may include new information.

Part 1. Overview

Chapter 1. Store architecture overview

This chapter provides an introduction to the WebSphere Commerce Server store architecture.

What is an online store?

An online store is a store that uses Internet technologies to sell or exchange goods or services. It is composed of a collection of Web pages that display your products and allow customers to purchase them. The home page brings customers into the store, and directs them to your products and services. Online catalog pages group products together, and direct customers to the product pages, where they can find detailed information about the product. In business-to-consumer stores, the Shopping Cart page fulfills the same role as a physical shopping cart: you add products you wish to purchase to it, and then pay for them in the Checkout pages. In business-to-business sites, certain pages allow you to submit orders and requests for quotes (RFQs).

Store pages are created using JavaServer Pages (JSP) technology. Each page contains HTML for static content, client-side JavaScript™ to handle input data and sophisticated data display, URLs to invoke WebSphere Commerce Server commands and other views, as well as JSP tags and Java™ code for generating dynamic content. A set of commerce data beans included with WebSphere Commerce Studio and WebSphere Commerce are available for use by your JavaServer Page files, allowing you to access information from the database, such as the price of a product, or the product's attributes.

Your store is also composed of the database assets necessary to create a functional store. For example, a functional store must include data on catalogs, taxes, shipping, and currency.

The composition of a store

An online store is composed of the following assets:

- Store front

  The external portion of your store, or the portion that displays to your customers, is known as the store front. The store front is comprised of Web assets such as HTML pages, JSP files, style sheets, images, graphics and other multimedia file types.

  This guide discusses the concepts and tasks involved in creating the JSP files that build your store pages. For more information, see Chapter 3, "Developing your store front" on page 15.
- Back office

  The portion of your store that customers don't see, the commands, customized code, and the implementation of business logic that allow a customer to purchase a product in the store front, is known as the back office.

  For more detailed information on creating business logic or customized code see the *IBM WebSphere Commerce Programmer's Guide*.
- Store data

  The data assets that compose your store. 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, a process to handle orders, the inventory to fulfill the request, and a shipping process in place. You must also have methods for processing and collecting payment.

The concepts and tasks involved in creating store data are discussed in Part 4, "Developing your store data" on page 35.

Store architecture

The WebSphere Commerce store architecture consists of the following components:

- WebSphere Commerce Server
- WebSphere Commerce Server instance
- Store configurations

WebSphere Commerce Server

The WebSphere Commerce Server is the server that handles the store-and commerce-related functions of an e-commerce solution. The store front assets and back-office business logic reside in a Web application within the WebSphere Commerce Server. WebSphere Commerce provides a default Web application (`WCS Stores`) for your use, or you can create your own.

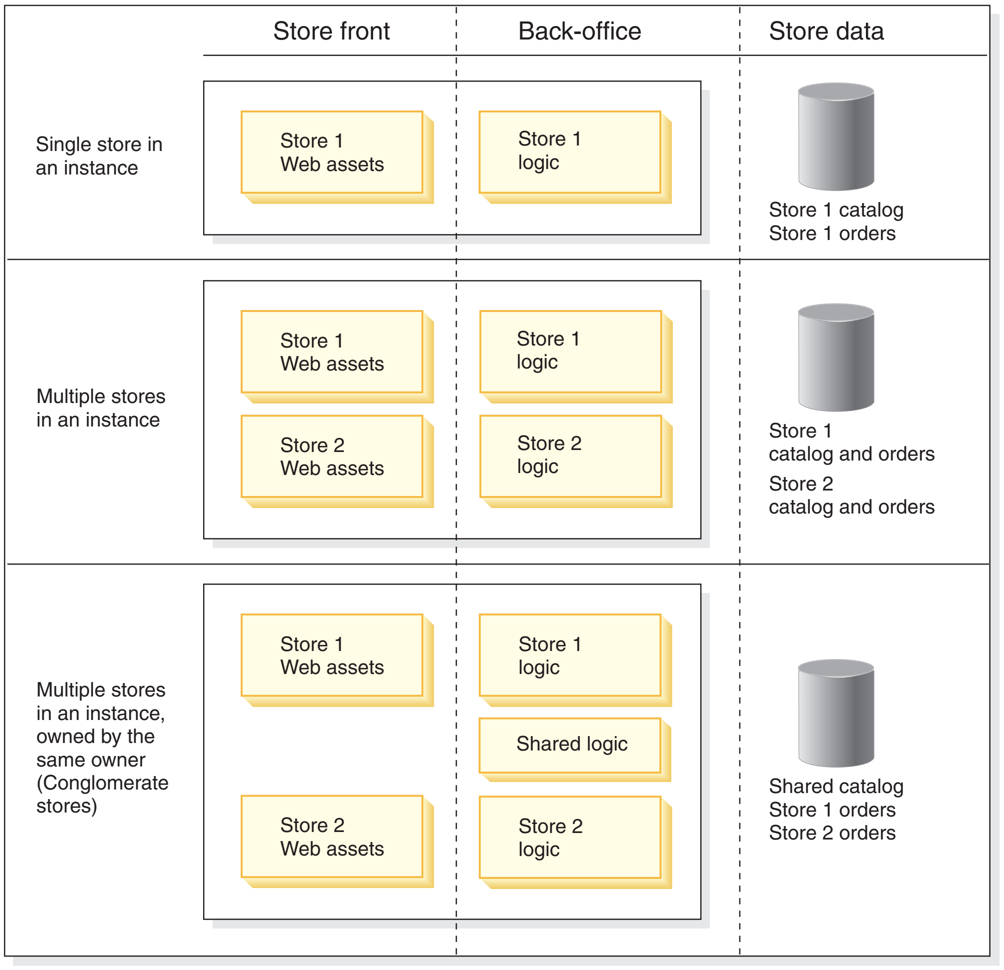
A Web application can contain the assets for one store, or the assets for multiple stores. When a Web application contains multiple store fronts and back offices, the assets for each store are separated by store directory (storedir).

WebSphere Commerce Server instance

A WebSphere Commerce Server instance is a WebSphere Application Server application with an associated database. An instance can support multiple stores. All stores in an instance share the same database and may share some types of data, for example, catalog, fulfillment, or receipts. All stores in an instance also share the same EJB container.

Store configurations

WebSphere Commerce supports several store configurations. That is, using WebSphere Commerce you can create a single store in an instance, or you can create multiple stores in an instance with separate store fronts, back offices, and store data. Or, you can create multiple stores in an instance with separate store fronts, shared back offices, and shared catalogs. The following diagram illustrates some possible store configurations.



Note: Each store has its own identifier. The licensing for WebSphere Commerce sets a specific limit on the number of stores you may create. You may purchase additional entitlement. See your licensing agreement for details.

Chapter 2. Store development

This chapter provides an overview of the store development process in WebSphere Commerce.

Store development options

WebSphere Commerce provides several options to develop your store:

- Creating a store based on a sample
- Creating a store by developing new store assets
- Creating a store using a combination of the sample store and new store assets

Creating a store based on a sample

In WebSphere Commerce, the fastest and easiest way to create an online store is to copy one of the sample stores provided with WebSphere Commerce and then customize it to meet your needs.

Sample stores

WebSphere Commerce includes several fully functional online sample stores that you can use as the basis for creating your own store. These samples, which include both business-to-consumer stores and business-to-business stores, implement many of the most commonly used features in today's top electronic commerce sites, and provide all the necessary store assets. For more information about the online stores provided with WebSphere Commerce, see the WebSphere Commerce online help.

Why start with a sample store?: Although it is possible to create an entirely new online store with WebSphere Commerce, using a copy of one of the sample stores as the base for your own store allows you to create a functional store much more quickly.

WebSphere Commerce requires that certain data be loaded into the WebSphere Commerce Server database to create a functional store, and that this data be loaded in the order determined by the schema. Since the sample stores include all the mandatory data in the order and structure that the WebSphere Commerce Server database requires, using one as a base for your own store saves you a substantial amount of time during the initial creation period.

After creating a copy of a sample store, you can edit it a lot or a little, depending on your store's needs. For example, you may only need to edit the data using the tools available with WebSphere Commerce and change the look and feel of the store pages using WebSphere Commerce Studio. Or, you may need to edit the XML files or the database directly to make more comprehensive changes to the data, and rewrite the store pages to change the store's flow and features. For more information on editing stores, see the WebSphere Commerce online help topic *"Changing store database assets."*

WebSphere Commerce also provides several reference stores that are designed to be used as code samples for the highlighted features. A reference store is an online store which contains fully functional code for selected features of an online store, for example, coupons. Reference stores are available from

▶Business `http://www.ibm.com/software/webservers/commerce/wc_be`

/downloads.html

Professional http://www.ibm.com/software/webservers/commerce/wc_pe /downloads.html

For more information on creating a store based on a sample, see the WebSphere Commerce online help.

Creating a store by developing new store assets

Not everyone will want to create their store by basing it on a sample store. For example, if the flow of your store pages is significantly different than that of any of the provided samples, or if you plan to significantly customize the WebSphere Commerce Server database schema, you may want to create your store by developing your own store front, back office and store data assets. See “Store development tools” on page 10 for a list of tools provided with WebSphere Commerce.

Creating a store using a combination of the sample store and new store assets

Using a combination of the sample store and developing new store assets may be the method of store development that works best for you. For example, if some of the database assets in one of the sample stores closely match your store’s needs, but the flow of that store’s pages does not, you can copy the database assets from the store and customize them, while developing entirely new Web assets.

The store archive

The sample stores included with WebSphere Commerce are provided in store archive format. A store archive file (`.sar`) is a compressed archive file (for example, a ZIP file) that contains all the assets necessary to create a store. It is primarily used as a vehicle for packaging and delivering stores in a format that can be easily copied, and then used as a base upon which to create new stores. A store archive only needs to be published to the WebSphere Commerce Server to create a functional store that you can view, browse, and shop.

Typically, a store archive is composed of the following files:

- Web assets: The files that create your store pages, such as HTML files, JSP files, images, graphics, and include files. Web assets are grouped together as a compressed file in the store archive.
- Property resource bundles (optional): Contains the text for your store pages. If your store supports more than one language, the resource bundle will contain multiple bundles; that is, one bundle per language.
- Store data assets: The data to be loaded into the database. Store data assets include data such as campaigns, catalog entries, currencies, fulfillment information, pricing, shipping, store, and taxation information. For a more detailed list of store data assets, see Part 4, “Developing your store data” on page 35.

  The store database assets in the sample store archives provided with WebSphere Commerce are well-formed, XML files valid for the Loader package. 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, which are resolved by the ID Resolver when the store is published. The use of these conventions allows the sample store archives to be copied and published multiple times. For more information, see Appendix B, "Creating your data" on page 305.

For more information on the Loader package, see Part 7, "Publishing your store" on page 207.

- Payment assets: Configuration information for the IBM Payment Manager.
- A descriptor: An XML file, `sarinfo.xml`, that describes the store archive, including the names of the Web assets compressed archive file, the resource bundles, and the store database asset XML files. The `sarinfo.xml` file also contains the names of include files and consistency checking files, as well as information about the archive file that is needed during the publishing process. The `sarinfo.xml` is the only mandatory file in a store archive.

Note: The Business ToolTech and NewFashion sample store archives also including the following files:

- tools_properties.zip
- tools_xml.zip
- runtime_xml.zip

These files are used by Store Services to configure stores. These files should not be changed, removed, or copied to other stores.

Sample store archives

A sample store archive file (.sar) is a store archive that is meant to be copied and used as a base upon which to create new stores. Sample store archives include a few conventions that allow them to be copied and published multiple times. These conventions include the following:

- No references to generated primary keys or foreign keys: Sample store archives do not contain generated primary keys that are specific to a particular instance of the database. Instead they use internal aliases, which are resolved by the ID Resolver when the store is published. For more information, see Appendix B, "Creating your data" on page 305.

The sample stores provided with WebSphere Commerce are sample store archives. These stores are available from the **Samples** list in the Create Store Archive page in Store Services.

Determining when to use a store archive

The store archive is designed as a vehicle for packaging and delivering stores. If you want to use your store as a sample to be delivered to others, to deploy it on another server or platform, or to use it as the basis of creating other stores, consider bundling it in the store archive form.

You may also want to use the store archive if your store is closely based on one of the sample stores and you do not need to make many changes to the archive.

If you choose to use the store archive, and have created a store using a sample store, your store will already be in store archive format. You can then maintain your new store in the store archive format by ensuring that all changes you make during store development are reflected in the store archive.

You can also package a store you have created using other methods as a store archive. For more information on creating a store archive, see Part 6, “Packaging your store” on page 199.

When wouldn’t I want to use a store archive?

You may choose not to use the store archive if you are creating a single instance of your store, or if you are making extensive changes to the existing WebSphere Commerce schema. Changes to the WebSphere Commerce schema are not immediately supported by the store archive and WebSphere Commerce store development tools out of the box. If you want to maintain your store archive after making extensive schema changes, contact an IBM representative for more information.

Store development tools

WebSphere Commerce provides a variety of tools to help you develop your store. Which tools you use depends on how you choose to develop and package your store.

Tools for developing the store front

Developing your store front assets may include customizing the sample store pages, replacing them with existing pages of your own, creating new pages, or doing a combination of all three.

WebSphere Commerce provides the following tools to create or edit store front assets:

- WebSphere Commerce Studio: Commerce Studio includes the tools required to create and edit your store front assets, including HTML, graphics, multimedia, and JavaServer Pages (JSP) files. Page Designer, included in Commerce Studio, allows you to create HTML or JSP files, as well as animated images. You can also configure WebSphere Commerce Studio to use another Web development tool of your choice. Refer to the WebSphere Studio online help for more information on registering your own tools with WebSphere Commerce Studio.

  If you plan to work with your store in store archive format, WebSphere Commerce Studio allows you to import the Web assets from the store archive into a Studio project, while keeping the store archive structure intact. After making changes to the JSP files, HTML files, and images using the Studio tools, you can export the files back to the store archive on the WebSphere Commerce Server and republish the Web assets.

  If you are not maintaining your store archive, you can publish the files directly to your functional store using WebSphere Commerce Studio.
- Store Services: The Web Assets dialog in Store Services allows you to replace the Web assets compressed archive file in the store archive with another set of Web assets, or to download the existing Web assets to a location of your choice, where you can edit them with your preferred Web development tool. If you are working with your store in store archive format, you can use the Web Assets dialog to put the changed assets back into the store archive. The Configure stores pages allow you to enable or disable different features in the JSP files in published stores. Currently Store Services only supports configuring the collaboration features: Collaborative Workspaces and Customer Care. These features are only available for configuration in stores based on the Business ToolTech and NewFashion sample stores.

For more information on using the tools in WebSphere Commerce Studio and Store Services to create and edit your store front assets, see the WebSphere Commerce online help. For more information on creating your store front in WebSphere Commerce, see Part 2, "Developing your store front" on page 13.

Tools for developing the store data

You have several options for developing and editing the database assets in the store.

- Store Services

  Store Services is a browser-based set of tools that works on store archives. Using Store Services, you can quickly create a store archive based on a sample provided with WebSphere Commerce. Once you have created a store archive, Store Services allows you to perform the following tasks:

  - Publish the store archive to create a functional store.
  - Change tax settings using the Tax notebook.
  - Change shipping settings using the Shipping notebook.
  - Change general store settings using the Store Profile notebook.

  Store Services does not allow you to edit all store data assets in the store archive. For a list of the assets you can edit using Store Services, see the WebSphere Commerce online help topic *"Changing store database assets."* To edit other assets in store archive, edit the XML assets directly.

  For more information on using Store Services, see the WebSphere Commerce online help.

  When to use Store Services: Use Store Services to copy sample store archives and to edit database assets in store archive format.

- WebSphere Commerce Loader package

  The WebSphere Commerce Loader package consists primarily of utilities for preparing and loading data into a WebSphere Commerce database. Use the Loader package to load large amounts of data and to update data in your WebSphere Commerce database. The Loader utility in this package uses valid and well-formed XML as input to load data into the database. Elements of the XML document map to table names in the database, and element attributes map to columns.

  For information on using the Loader package to develop and load data assets, see Part 7, "Publishing your store" on page 207.

  When to use WebSphere Commerce Loader package: Use the WebSphere Commerce Loader package to initially load database assets into the WebSphere Commerce database and to update them.

  Important: If you have changed the database schema, the Loader package is your only option for loading data into the database.

- WebSphere Commerce Accelerator

  WebSphere Commerce Accelerator is a workbench of online tools primarily used to maintain online stores through various store operations. However, since the WebSphere Commerce Accelerator allows you to edit data already in the database, you can use it as a store development tool once you have initially populated the database, whether it is with the sample store data, or data you have created. For a list of the database assets you can edit with the WebSphere Commerce Accelerator, see the WebSphere Commerce online help topic *"Changing store database assets."*.

When to use WebSphere Commerce Accelerator: Use the WebSphere Commerce Accelerator after you have already populated the WebSphere Commerce database.

- Editing the database directly

  You always have the option of editing the database directly using SQL inserts.

  Note: SQL is database specific. Oracle may require a different SQL syntax. Note that SQL statements will necessarily have database specific values and the SQL statements may not be reusable in another WebSphere Commerce Server instance.

Tools for developing the back office

Tools for developing the back office, including creating and extending commands, creating customized code, and implementing business logic are discussed in the *IBM WebSphere Commerce Programmer's Guide*.

The Store Developer's role

Store Developers develop all three types of store assets. They design and implement the store front assets, including JavaServer Pages files and the back office, including creating new commands and any necessary customized code. They also create the store data, and can modify any of the standard functionality included with WebSphere Commerce.

Store Developers who are creating the storefront and the store data must have programming skills in Java, JavaScript, HTML, JSP technology, and be familiar with the WebSphere Commerce store architecture, store data and store archives.

Store Developers who are creating the back office must have programming skills in Java, JavaBeans™, VisualAge® for Java, J2EE programming, and be familiar with the WebSphere Commerce programming model and object model. The *IBM WebSphere Commerce Programmer's Guide* provides more information on customizing the back office.

Store Developers may work with database developers and Web designers. Database developers modify and extend the WebSphere Commerce database schema for the purpose of implementing customized store functions, or integrating with existing database information. This member usually has database administrator skills for DB2® or Oracle.

Web designers create the look and feel for the site, and work with Store Developers to create store pages. Web designers should have experience using multimedia tools, HTML and JavaScript skills, and familiarity with JSP technology.

Note: The database developer and Web designer roles are not defined in the WebSphere Commerce Server. If necessary, database developers and Web designers should be assigned Store Developer access.

Once a store archive has been created, Store Developers have the authority to make changes to it manually or by using the Store Profile notebook and Tax and Shipping notebooks, but they do not have the authority to publish the store archive to the WebSphere Commerce Server.

Part 2. Developing your store front

Chapter 3. Developing your store front

This chapter provides an overview of the WebSphere Commerce store front architecture, including how the external portion of your store, the Web assets such as HTML pages, JSP files, style sheets, images, graphics and other multimedia file types, are displayed to your customers.

Store front architecture

WebSphere Commerce uses a system of *commands* and *views* to display the Web assets in a store front to customers.

- *Commands* perform a specific business process, such as adding a product to the shopping cart, processing an order, updating a customer's address book, or displaying a specific product page. When the action is completed, the command returns a view.
- *Views* display the results of commands and user actions, that is, views present your store pages (JSP files) to the customers. In order for the view to invoke a JSP file, the JSP filename must be registered with the view in the view registry (VIEWREG) table. The corresponding JSP file is stored using the JSP filename in the subdirectory (storedir) for the store under the `WCS Stores` webapp doc root.

Both commands and views are invoked using URLs. For example, when a customer clicks **Shopping Cart** in the sample store, the customer invokes the URL `https://hostname/path/OrderItemDisplay?`, which is passed into the WebSphere Commerce Server. The WebSphere Commerce Server calls the OrderItemDisplay command, and the shopping cart page is displayed to the customer.

When a customer clicks **Help** in the sample store, the customer invokes the URL `https://hostname/path/HelpView?`, which is passed into the WebSphere Commerce Server. The WebSphere Commerce Server calls the HelpView, which returns the Help page.

The WebSphere Commerce Server can also map multiple commands to a URL, which allows each store to optionally have its own implementation of that command.

Similarly, the WebSphere Commerce Server also allows you to map multiple JSP files to a single view, where each store can optionally register different JSP filenames for different device types

Note: The product display and category display commands return views as well as JSP filenames. These JSP filenames, which display products and categories are stored in the catalog data. For more information, see "Displaying store catalog assets" on page 74. You can optionally assign different JSP filenames to display products and categories for each member group or language supported by your store.

Default commands and views

WebSphere Commerce provides default commands and views which you can use in your store. These default commands and views are listed in the `wcs.bootstrap.xml` file. The bootstrap files are located in the following directory:

- NT `drive:\WebSphere\CommerceServer\schema\xml`
- 2000 `drive:\Program Files\WebSphere\CommerceServer\schema\xml`
- AIX `/usr/WebSphere/CommerceServer/schema/xml`
- Solaris `/opt/WebSphere/CommerceServer/schema/xml`
- Linux `/opt/WebSphere/CommerceServer/schema/xml`
- 400 `/qibm/proddata/WebCommerce/schema/xml`

If a needed command or view is not provided, you can create your own. For information on creating commands and views, see the *IBM WebSphere Commerce Programmer's Guide*.

Creating your store pages

The largest task in creating your store front is creating the actual store pages. Before beginning development work on the store pages, you should complete the following planning activities:

- Developing a list of store pages needed
- Developing a list of command and view URLs
- Associating JSP filenames with views

Developing a list of store pages

In order to develop a list of the pages needed to create your store, you need to know the business and functional requirements of the store, as well as any business processes that have been defined.

Working from use cases

Many people gather requirements in the form of use cases. Use cases define the business processes in your store, in the form of interactions between the customer and the proposed system. In the case of an online store, use cases may define how a customer registers at the store, browses the catalog, or orders an item.

A set of use cases, detailing the business processes for the sample stores are 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 if you wish to create use cases for your own store.

The following is an example of a Registration use case:

Registration use case: The registration process allows customers to enter personal information in the database.

Actor:

- Customer

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

- E-mail
- Password
- Verify password
- First name
- Last name

- Age (optional)
- Gender (optional)

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 the customer to manage their account following the process in the Manage Personal Account use case.

Alternate flows: 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 is invalid or does not match the verification password, the system issues a warning.

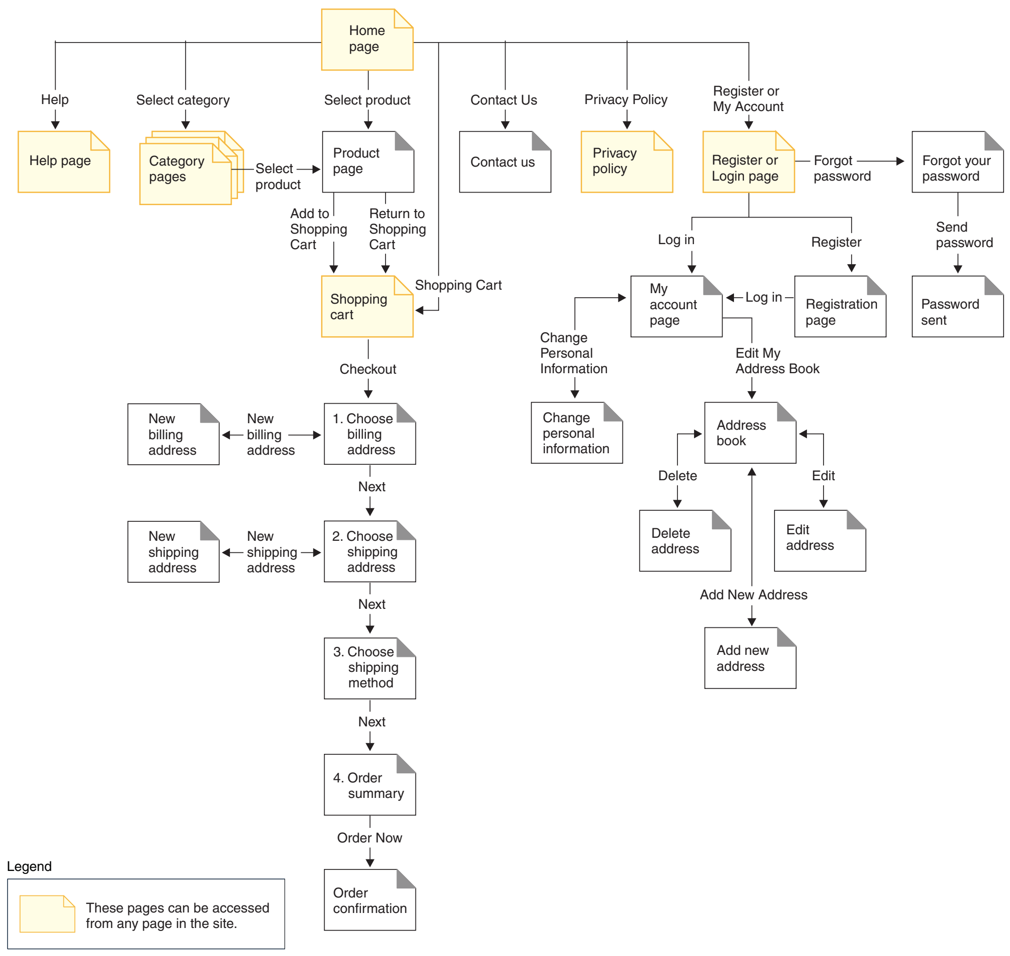
Determine the store shopping flow: Regardless of whether you develop use cases to illustrate your store's business processes, or use another method, once business processes are available, you can create the shopping flow for your store.

Note: Since use cases often contain flow information such as, ″If the customer selects **Submit**, the Order page displays,″ use cases can provide useful information for creating shopping flow diagrams.

The shopping flow reflects the requirements and business processes defined for your store, illustrating how a customer will move through the store. For example, a customer may enter your site through the home page and be asked to register before browsing the catalog, or you may choose to allow customers to view the catalog as guests, without registering. Some shopping flows allow customers to complete a ″quick checkout″, while others require that a customer completes all checkout steps every time they make a purchase. Or, your shopping flow can offer customers the choice of both checkouts.

To verify that the store flow diagram is complete, ensure that all steps in the use cases for your store are illustrated in the store flow diagram.

Mapping out the shopping flow visually, as the following diagram for the InFashion sample store's shopping flow does, allows you to see how customers will travel through your store.



The diagram for the InFashion shopping flow is quite simple. Although it includes the main flow of the a customer's journey through the store, it does not include any error scenarios. For example, what happens when a customer logs in using the wrong password, or enters an invalid credit card number? However, even a simple diagram like this allows you to develop a list of pages needed for the store. To start you will need to create a view for every page listed in the shopping flow diagram.

For example, if you were to create a store with the same shopping flow as in the InFashion diagram, you would have to create the following pages:

Note: The following table lists the view names used in for the InFashion store

InFashion shopping flow diagram pages (as seen by customer)	Corresponding view
Home page	StoreCatalogDisplayView
Help page	HelpView
Contact us	ContactView
Privacy policy	PrivacyView
Register or Login Page	LogonForm
Forgot your password	LogoffView
Password sent	ResetPasswordForm

InFashion shopping flow diagram pages (as seen by customer)	**Corresponding view**
My account page	LogonForm
Change personal information	UserRegistrationForm
Address book	AddressBookForm
Add new address	AddressForm
Delete address	AddressBookForm
Edit address	AddressForm
Registration page	UserRegistrationForm
Shopping cart	OrderItemDisplayViewShiptoAssoc
Choose billing address	OrderItemDisplayViewShiptoAssoc
New billing address	OrderItemDisplayViewShiptoAssoc
Choose shipping address	OrderItemDisplayViewShiptoAssoc
New shipping address	AddressForm
Choose shipping method	OrderItemDisplayViewShiptoDsp
Order summary	OrderDisplayPendingView
Order confirmation	OrderOKView

Note: Many of the views used in InFashion were created specifically for InFashion. These views are listed in the `command.xml` file in the InFashion store archive. For more information, see "Registering commands, views, and URLs in WebSphere Commerce" on page 49.

The above table implies only the basic set of pages you need to create. To determine what other pages you need to create, you can look more closely at the use cases or other methods used to define your business processes.

Error pages: The exception flows in your use cases can also help you determine what error pages you need to create for your store. The registration use case for InFashion specifies the following exceptions flows:

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

As a result, you will need to create an error page or error message for each exception flow.

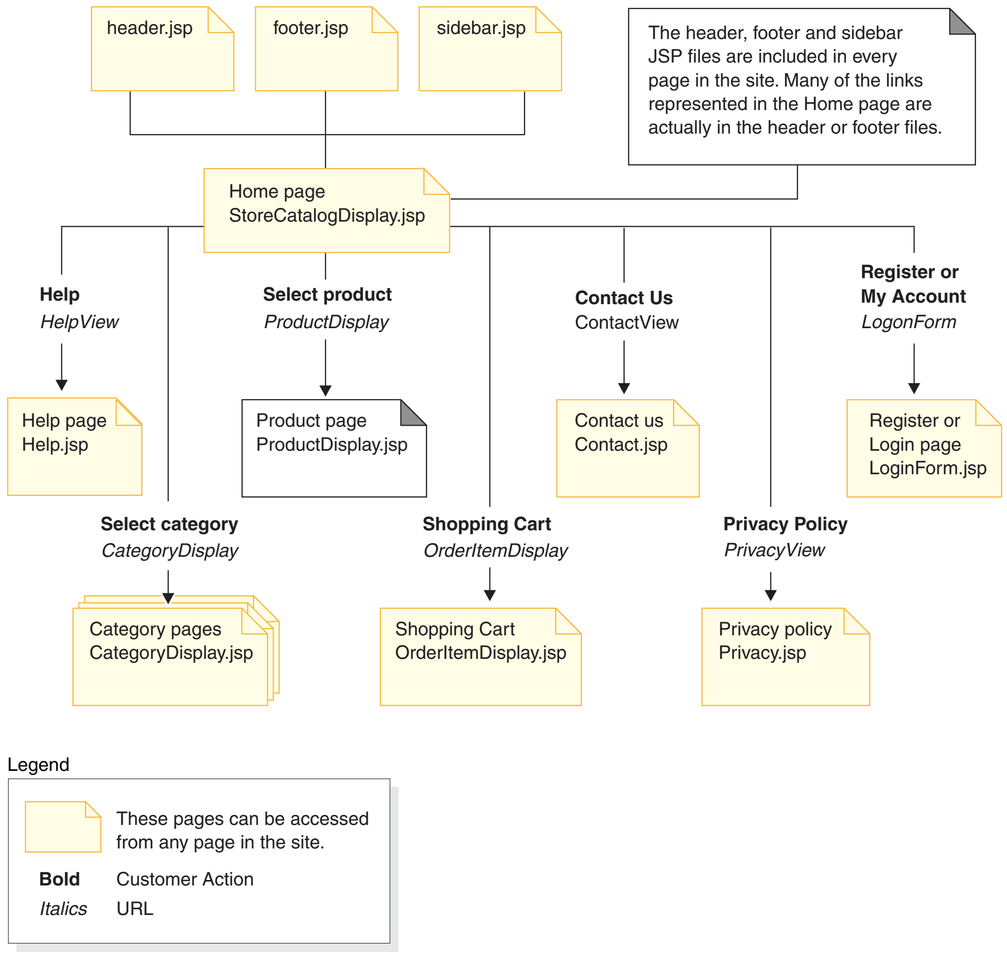
Developing a list of command and view URLs

As demonstrated in the InFashion shopping flow diagram, business processes, such as checkout and register, may require several pages. In order to combine these pages into a working business process or flow, rather than just a collection of pages, you must include commands and views in your pages.

Developing a list of URLs needed

Just as you developed a list of pages necessary to create the store, you also need to develop a list of the command and view URLs necessary to implement the business processes for your store. Using the shopping flow diagram for your store, and the list of default commands and views, identify the URLs necessary to complete each action.

Understanding which command and view URLs are used in the sample stores may also help you determine what URLs you need in your store. The following illustration identifies the URLs for some of the actions in the InFashion shopping flow diagram. For more details, see the information on the samples stores in the WebSphere Commerce online help.



Associating JSP filename to views

The WebSphere Commerce Server uses view commands to compose a view as a response to a request. WebSphere Commerce Server provides the following view commands:

- `HttpForwardViewCommandImpl`: This view command forwards the view request to a JSP file.
- `HttpRedirectViewCommandImpl`: This view command redirects the view request to another URL.

- HttpDirectViewCommandImpl: This type of view command sends the response view directly to the client. It does not call a JSP file. Direct views allow controller commands to produce the output response (rather than the view command).

Use the HttpForwardViewCommandImpl view command to render JSP files directly. For example, in the diagram illustrating the URLs used in InFashion, in order to display the Help page (Help.jsp), the HelpView is registered in the view registry and associated with the Help.jsp and the HttpForwardViewCommandImplcommand. This is demonstrated in the following example:

```
<viewreg
viewname="HelpView"
devicefmt_id="-1"
storeent_id="@storeent_id_1"
interfacename="com.ibm.commerce.command.ForwardViewCommand"
classname="com.ibm.commerce.command.HttpForwardViewCommandImpl"
properties="docname=help.jsp"
internal="0"
https="0"
/>
```

Note that the the fully qualified classname for the interface and the implementation class is used.

Note: In this example, the URL that invokes the view is not restricted. That is, anyone can access the URL directly. If you use this technique, ensure that the JSP file only renders public data.

Use the HttpForwardViewCommandImpl view command to render views returned from a display command. A display command reads data from the database, but does not change it. For example, in the diagram illustrating the URLs used in InFashion, the OrderItemDisplay command returns the OrderItemDisplayViewShiptoAssoc view. When this view was registered in the view registry, the OrderItemDisplay.jsp and the HttpForwardViewCommandImpl were associated with it. This is demonstrated in the following example:

```
<viewreg
viewname="OrderItemDisplayViewShiptoAssoc"
devicefmt_id="-1"
storeent_id="@storeent_id_1"
interfacename="com.ibm.commerce.command.ForwardViewCommand"
classname="com.ibm.commerce.command.HttpForwardViewCommandImpl"
properties="docname=OrderItemDisplay.jsp"
internal="0"
https="0"
/>
```

Note: In this example, the URL that invokes the view is restricted. That is, only those with designated access can invoke the URL directly. Access to this view is controlled by whoever has access to the URL for the command.

You must associate a JSP filename for every view associated with every display command (for example, OrderItemDisplay) you use. For more information about associating JSP filenames with views, see “Registering commands, views, and URLs in WebSphere Commerce” on page 49.

Note: ProductDisplay and CategoryDisplay list the associated JSP filename in the catalog data rather than the view registry.

Use the `HttpRedirectViewCommandImpl` view command to render views returned from a command that changes the database. To use the redirect view, specify the view name using the `&URL=` parameter on the URL. For example, when you add address information in the InFashion sample store AddressForm and click **Submit**, it invokes the AddressAdd command. The URL used to invoke the AddressAdd command specifies AddressBookForm as the view using the `&URL=` parameter. This results in a redirect to the AddressBookForm view. When the AddressBookForm view was registered in the view registry, the `AddressBookForm.jsp` and the `HttpForwardViewCommandImpl` were associated with it.

You must use the `URL=parameter` technique for all non-display commands. Non-display commands are commands that cause changes to the data in the database.

Part 3. Store data overview

Chapter 4. Store data

This chapter provides an overview of the WebSphere Commerce Server store data architecture and the data assets that create a store. The WebSphere Commerce Server information model is also introduced in this chapter.

What is store data?

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).

The store data information model

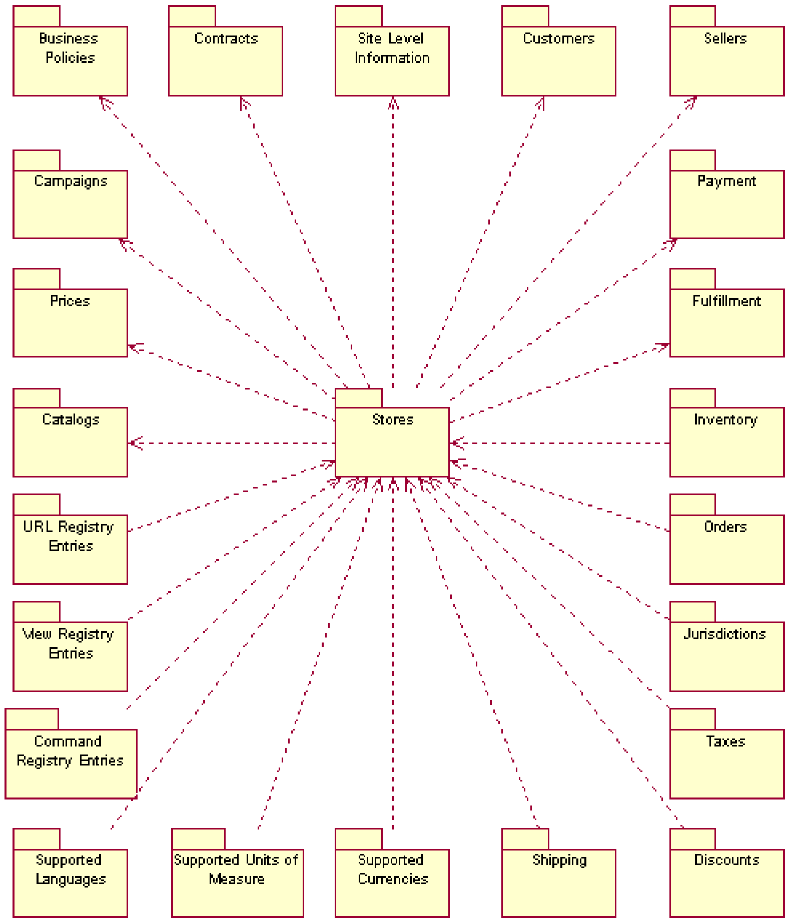
This guide uses an information model to illustrate how store data is structured in the WebSphere Commerce Server. The WebSphere Commerce Server information model is a high-level abstraction of the information contained in the WebSphere Commerce Server data and object models. The information model highlights the most important features of the data and object models, but does not include the lower level details that are specific to the schema and object implementations.

For example, certain tables and objects in the data or object models that contain entity-relationship data (such as foreign key pairs) do not display in the information model as entities. Instead these entity relationships are implied by the relationship lines between entities in the information models. The information model also does not illustrate *detail* extensions (additional data attributes of an entity that are stored in a separate table as a result of implementation concerns: for example, the product description is a separately stored extension of the product entity). For more information on relationship objects and detail extensions, see the object model in the WebSphere Commerce online help.

For more information on the WebSphere Commerce object and data models, see the WebSphere Commerce online help.

Store data assets

The following diagram illustrates the data assets of a WebSphere Commerce store.



This diagram, and all others in the store data section are part of the WebSphere Commerce information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

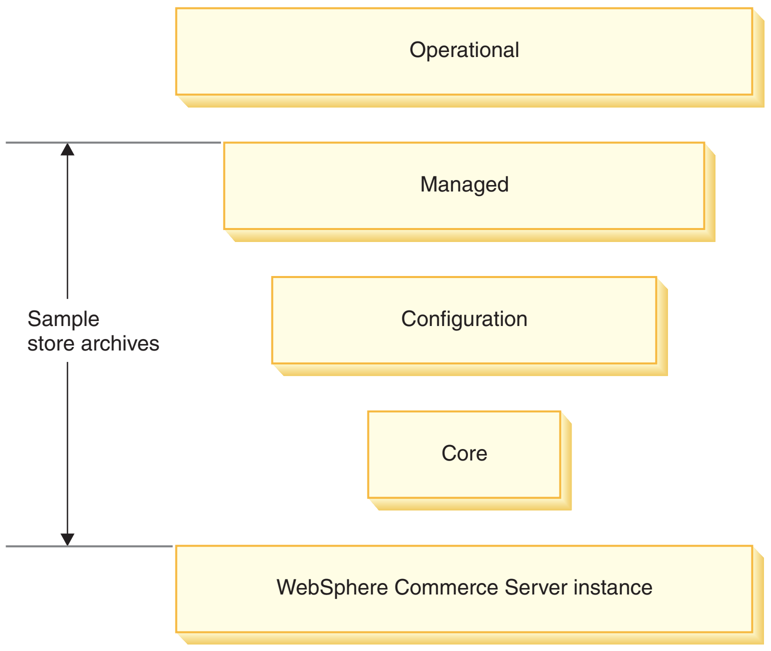
Notice the two directions of arrows in the diagram. In some cases, the arrow points toward the store, for example from Currency to Stores. In this case, the currency assets are exclusive to this particular store and are part of the store. These are the currencies that this store supports. If the store was deleted, the list of currencies supported by a store would also be deleted.

When an arrow points from Stores to an asset, for example Catalog, the asset can be shared by other stores. One catalog can be shared by several stores. However, if the store was deleted, the catalog would still exist.

Each of the data assets illustrated in the above diagram is discussed in more detail in the chapters in Part 4, "Developing your store data" on page 35

Store data architecture

Data in WebSphere Commerce stores conforms to the architecture depicted in the following diagram. Each of the store data assets illustrated in the diagram in "Store data assets" on page 26, can be classified as belonging to one or more of the types of store data illustrated below.



WebSphere Commerce Server instance

The basic level of data is contained in the WebSphere Commerce Server instance. When an instance is created, the bootstrap files, which are loaded in XML format, populate the database with information. The bootstrap files create the following types of data:

- Calculation usage types, device types (browsers, e-mail, I-Mode, and so on), message types and roles
- The default administrative ID, WCSADMIN
- The default commands, views and URLs
- The default business policies
- The languages and currencies supported by the instance
- The default organization, which can be used as the store owner
- The default site organization
- The default store group

This information is available to all stores that exist in that instance, and is identified as the Site Level Information in the diagram in "Store data assets" on page 26.

For more detailed information on the bootstrap files and the database tables they populate, see the WebSphere Commerce online help.

Core data

The next level of store data is the core data. The core data creates the minimum data for a store, including:

- The store identifier in the STOREENT table. This creates a store in the database.

- The default contract.
- The store identifier in the contract database tables.
- The member identifier for the organization that owns the store in the contract database tables.
- The store directory in the STORE table. The store directory is the directory in which the store's Web assets are located.
- The nickname or identifier for the store's address in the STADDRESS table. The nickname is unique for each store.

If you used Store Services to create a store, this information was created for you with the new store archive. Store Services allows you to select the default organization that can act as the store owner, or you can create another organization to act as the owner using the Administration Console. If you did not use Store Services to create a store, you will have to load this information into the database using the Loader package, or edit the database directly.

The Stores data in the diagram in "Store data assets" on page 26 is core data.

Configuration data

Configuration data controls the commerce server run time. The common server run time provides a framework in which the commerce applications are deployed and executed. The framework consists of the programming model, the process model, exception handling, transaction control, data access, and the persistence model. The common server run time leverages the run time services provided by WebSphere Application Server to support WebSphere Commerce Server applications. Configuration data determines which commands, views, and JSP files your store will use to display store pages.

The following data assets identified in the diagram in "Store data assets" on page 26 are classified as Configuration data:

- Command Registry Entries
- View Registry Entries
- URL Registry Entries

Managed data

Managed data is that data which the seller creates, and is read-only for customers of the seller's site. Since the seller is in complete control of the state of this data, managed data may be managed through a content management system.

The following data assets identified in the diagram in "Store data assets" on page 26 that are classified as managed data:

- Campaigns
- Business policies
- Contracts
- Fulfillment centers
- Jurisdictions
- Tax
- Discounts
- Shipping
- Currencies
- Units of measure
- Languages

- Catalogs
- Prices
- Customers
- Sellers
- Payment

Operational data

Operational data is data which is created or changed (directly or indirectly) by customers of the site as a result of their interactions with the site. For example, customer orders are considered operational data, as are inventory levels, which go up and down as your store operates. Customers are also considered operational data. Data created by the seller can also be operational.

Since changes to operational data are not under the complete control of the seller, it doesn't make sense to manage this data using a content management system.

The following data assets identified in the diagram in "Store data assets" on page 26 are classified as operational data:
- Orders
- Inventory
- Fulfillment
- Customers

Note: In some instances the line between operational and managed data may be hard to determine. For example, in one store, customer and contract data may be considered managed data, while in another store, the same type of data may be considered operational. The first store may manage their customer data and related contracts because they have a specific set of customers (that is, customers cannot register online). However, the second store allows customers to register online, and create contract information online.

A second example involves catalog data. In a single seller site, the catalog is considered managed data. In a marketplace site, catalog data may be considered operational.

In some sites, certain records of the same data type may be considered managed while other records are considered operational. For example, the default contract may be managed data, but the specific contracts negotiated online are operational data.

Store data architecture and the sample stores

The sample stores provided with WebSphere Commerce include most of the types of store data in store data architecture. For example, a WebSphere Commerce Server instance must exist before a store can be created using a sample store or a sample store can be published. Then when you create a store based on a sample store using the tools in Store Services, the core data is created. The sample stores include all the necessary configuration, and most of the managed data required for a functional store. When creating stores based on certain sample stores, you may be instructed to complete some set up of data, using the tools in the WebSphere Commerce Accelerator.

Tools for creating data

WebSphere Commerce provides several tools to create and manipulate your store data. These tools are listed below:

WebSphere Commerce Loader package

The Loader package consists primarily of utilities for preparing and loading data into a WebSphere Commerce database. For more information, see Part 7, "Publishing your store" on page 207.

Store Services

Store Services edits pre-published data in the form of a store archive, rather than live data in the database. Store Services also allows you to publish all store data assets to the database. For more information, see Part 7, "Publishing your store" on page 207.

Administration Console

The Administration Console allows you to control your site or store by completing administrative operations and configuration tasks. You can also use the Administration Console to create new organizations and users, as well as assign users to roles (Store Developer, Store Administrator, Site Administrator, and so on). The Administration Console also allows you to identify which notification and messaging types will be available in your store.

WebSphere Commerce Accelerator

The WebSphere Commerce Accelerator is a workbench of online tools that allow you to create and maintain various store assets. A large portion of store data can be created and managed using the tools in the WebSphere Commerce Accelerator. However, in some cases, certain data must already be loaded into the database, using either the publish tools in Store Services or the Loader package before you can create data using the WebSphere Commerce Accelerator. For more information, see the "Tool and store data summary chart" on page 31.

Organizational Administration Console

The Organizational Administration Console allows you to manage the organizations that access your site or store. The Organizational Administration Console also allows the buyer's Administrator to manage buyers within their organization.

Tool and store data summary chart

The following chart lists the tools you can use to create each type of data.

Tools for creating data	**Core data**	**Configuration data**	**Managed data**	**Operational data**
WebSphere Commerce Loader package	Use the Loader package to load core data in the form of an XML file. For more information, see "Creating store data assets in an XML file" on page 42.	Use the Loader package to load configuration data in the form of an XML file. For more information, see "Creating an XML file to register commands, views, and URLs" on page 49.	Use the Loader package to load managed data in the form of an XML file. For more information, see the corresponding chapters on the managed data assets.	In general, operational data cannot be loaded with the Loader Package.
Store Services (for pre-published data in the form of a store archive only)	When you create a new store archive using Store Services, the core data is created for you. For more information on using Store Services, see the WebSphere Commerce online help.	Not applicable.	Store Services allows you to create and edit portions of the following managed assets: • Jurisdictions • Taxes • Shipping • Currency • Languages For more information on which portions of the database assets Store Services allows you to edit or create, see the WebSphere Commerce online help, ″Changing store database assets″.	Not applicable.
Administration Console	Use the Administration Console to create an organization to act as the store owner.	Not applicable.	Not applicable.	Not applicable.

Tools for creating data	Core data	Configuration data	Managed data	Operational data
WebSphere Commerce Accelerator	Not applicable.	Not applicable.	Use the WebSphere Commerce Accelerator to create or edit the following data: • Campaigns • Contracts (a default contract must exist in the database before you can use the Business Relationship Management tools in the WebSphere Commerce Accelerator to create additional contracts or change existing ones. Use the Loader package or Store Services to create a default contract in the database).	Customers create operational data when they register with the store, or make purchases from it. However, in some cases, you can use the WebSphere Commerce Accelerator to place orders for a customer, or to create a return. The WebSphere Commerce Accelerator also allows you to manage your inventory.

Tools for creating data	**Core data**	**Configuration data**	**Managed data**	**Operational data**
WebSphere Commerce Accelerator continued	Not applicable.	Not applicable.	• Fulfillment • Discounts • Catalogs (a master catalog must exist in the database before you can use the Product Management tools in the WebSphere Commerce Accelerator to create a navigational catalog, and add or change product information. Use the Loader package or Store Services to create a master catalog in the database.) • Prices	Not applicable.
Organizational Administration Console	Not applicable.	Not applicable.	Not applicable.	Customers and buyers are created when they enter the store. However, with the Organizational Administration Console, you can approve buyers, or create new ones.

Part 4. Developing your store data

The chapters in this section explain each of the store data assets in more detail. The store data assets in the this section are organized according to the WebSphere Commerce store data architecture structure:

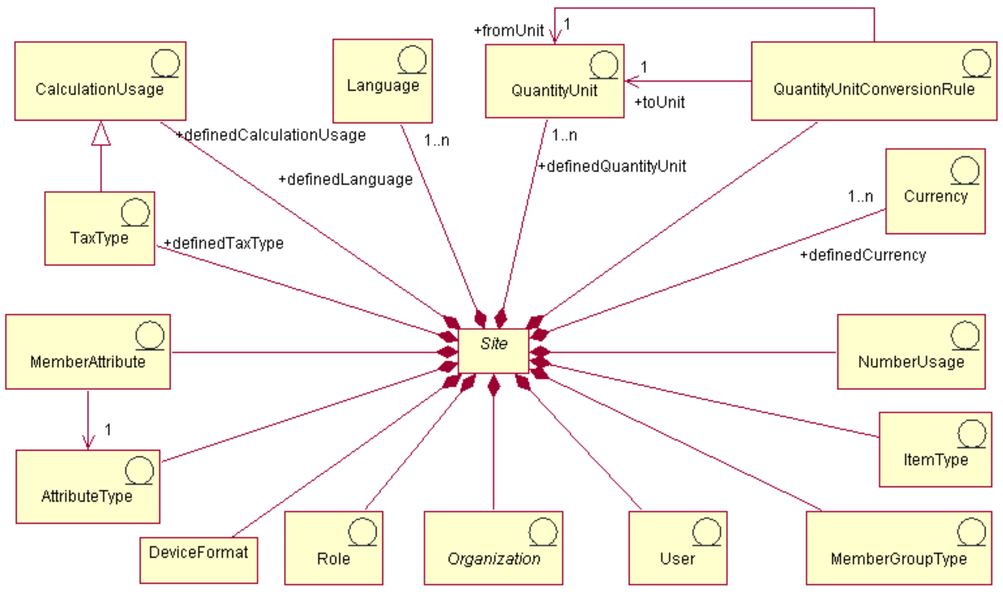
- WebSphere Commerce Server instance
  - Site
- Core data
  - Store
- Configuration data
  - Command Registry
  - View Registry
  - URL Registry
- Managed data
  - Shared assets
    - Catalog
    - Prices
    - Contracts (including Business Policies)
    - Fulfillment
    - Campaigns
    - Payment
  - Exclusive to the store assets
    - Supported languages
    - Supported currencies
    - Supported units of measure
    - Jurisdictions
    - Shipping
    - Taxation
    - Discounts
- Operational data
  - Inventory
  - Orders
  - Customers

Chapter 5. Site assets

Each WebSphere Commerce Server instance has its own database of relational information. An instance is created by the bootstrap files, which populate the database tables with information, after the schema has been created. Once the data has been loaded, you can see the pre-loaded information in the appropriate database tables. Many database tables contain store or store group level information that is particular to a store or group of stores. This information is usually managed by Store Administrators. However, some tables contain information that represents WebSphere Commerce site level capabilities available for use by all stores in the instance, and which are managed by the WebSphere Commerce Site Administrator. These capabilities are discussed in this chapter. For more information on the bootstrap files, see the WebSphere Commerce online help. For more information on store-specific asset information see Chapter 6, "Store assets" on page 41.

Understanding site assets in WebSphere Commerce

The following diagram illustrates the types of data the site contains and their relationships to the site.



For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303. This diagram, and all others in the store data section are part of the WebSphere Commerce information model. For more information on the information model, see "The store data information model" on page 25.

Language

A site can define many *languages* in the LANGUAGE table, and describe them in the LANGUAGEDS table. Each store generally supports a subset of these languages by adding rows to the STORELANG table. The ten pre-defined languages are: German, Traditional and Simplified Chinese, Japanese, Korean, Italian, French, Spanish, Brazilian Portuguese, and English.

Member attributes

Member attributes are stored in the MBRATTR table and represent the set of defined attribute names for which values can be stored for organizations or users. Examples of such attribute names include JobFunction, ProcurementCard, SpendingLimit, ReferredBy, and CountryOfOperation. Attribute values for particular organizations or users can be stored in the MBRATTRVAL table, and these values can be different for different stores or store groups.

Attribute types

Attribute types are stored in the ATTRTYPE table and represent the defined data types that can be used to represent attribute values. Examples of data types include INTEGER, STRING, and FLOAT.

Member group types

Member group types are stored in the MBRGRPTYPE table and represent the set of defined member group usages. Member groups are assigned usages by adding rows to the MBRGRPUSG table. Examples of member group usages include AccessGroup (for use with access control policies) and UserGroup (for general purposes, such as customer groups).

User

User represents authenticated user identities. Users generally represent customers placing or approving orders on behalf of buying organizations, selling agents processing orders for selling organizations or maintaining store level assets, or Site Administrators maintaining the WebSphere Commerce Server instance. Each user is associated with one site and is defined in the USERS table.

Organization

Organization represents organizations and organizational units within organizations. Organizations generally represent business entities responsible for buying or selling. Orders placed by customers in a business-to-business buying organization are recorded as being placed on behalf of the buying organization. Stores, catalogs, and fulfillment centers are owned by organizations responsible for certain aspects of selling. Organizations are defined in the ORGENTITY table.

Role

Role represents the set of defined roles that users can be assigned within organizations. For example, a user may be assigned the role of Customer Service Representative within a selling organization, or may be assigned the role of Buyer Approver within a buying organization. The names and descriptions of the default roles are populated in the ROLE table. For more information on specific roles, see the WebSphere Commerce online help.

Quantity unit conversion

Each site has *quantity conversions*. These represent multiplication or division operations that are used to convert between different units of measure. These are populated in the QTYCONVERT table.

Quantity units

Quantity units represent the set of units of measure for the site. They are defined in the QTYUNIT table and described in the QTYUNITDSC table. Each store can specify how amounts in each unit of measure are rounded and formatted for display, depending on their intended usage, by adding rows to the QTYFORMAT table.

Tax types

Tax types represent the calculation usages that calculate taxes. Sales tax and shipping tax are two different calculation usages that calculate taxes. Tax types are defined in the TAXTYPE table.

Calculation usage

Calculation usage represents the different kinds of calculations that can be performed by the OrderPrepare command. Calculation usages are defined for discounts, shipping, sales tax, shipping tax, and e-coupons. Calculation usages are defined in the CALUSAGE table.

Currency

Each site defines a number of *currencies* in the SETCURR table and describes them in the SETCURRDSC table. Each store supports a subset of these currencies by adding rows to the CURLIST table, one row for each currency supported.

Number usage

Number usage represents the intended usage for numbers. Stores can specify different rounding and formatting rules for the numbers they display according to how they are used. For example, a store may round unit prices to four decimal places by specifying the ″unit price″ usage, but other monetary amounts to two decimal places by specifying the ″default″ usage. Number usage is defined in the NUMBRUSG table, and described in the NUMBRUSGDS table.

Item types

Item types represent the different kinds of base items. The two types of base items in WebSphere Commerce are dynamic kit and normal item. Item types are pre-defined in the ITEMTYPE table. For more information on base items, see Chapter 21, “Inventory assets” on page 167.

Device formats

Device formats are stored in DEVICEFMT table and represent the many device formats a site uses such as browsers, I_MODE, e-mail, MQXML, and MQNC. All these device types allow users to interact with the site through various media.

Note: For some of the site assets, such as Language, Currency, Quantity unit, and Quantity unit conversion rule, the Site Administrator can extend the site level capabilities by adding rows to the appropriate tables. For the others, related customizations may be also be required to extend the site level capabilities they represent. For example, if a Site Administrator added a new number usage to display subtotals with a customized currency symbol, then the program that displays subtotals would have to be customized to specify the new subtotal number usage when formatting subtotal amounts for display.

Creating site assets in WebSphere Commerce

Site assets are created when you create an instance in the WebSphere Commerce Server. For more information on creating an instance in the WebSphere Commerce Server, refer to the *IBM WebSphere Commerce Installation Guide*, Chapter 5 ″Creating or Modifying an Instance.″

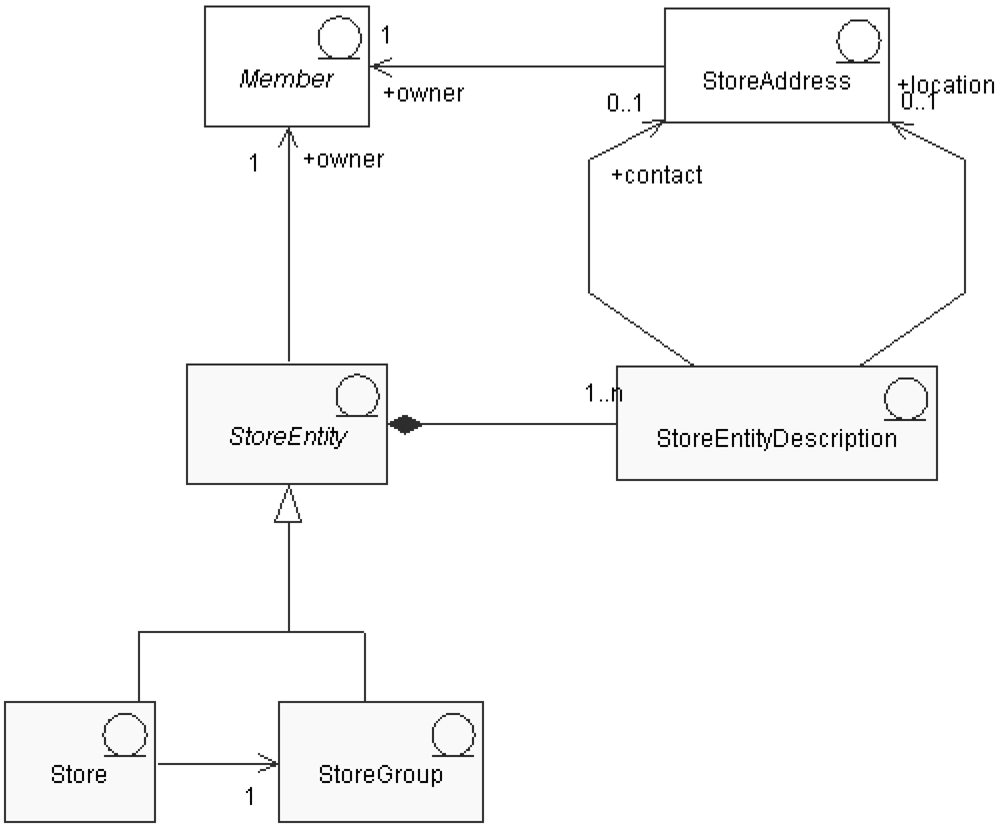
Chapter 6. Store assets

In order to create a store in WebSphere Commerce you must first create the following in the database:

- The store
- The group to which it belongs
- The abstract store entity object that dually represents a store or store group

Understanding store assets in WebSphere Commerce

The following diagram illustrates the store assets in the WebSphere Commerce Server.



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

Store entity

A *store entity* is an abstract superclass that can represent either a store or a store group.

A store entity has one owner (a member). For more information on members, see Chapter 23, “Customer and Seller assets” on page 175.

Store entity description

The *store entity description* describes the store entity. A store entity may include a description. If your store supports multiple languages, the store entity description may be in multiple languages. The description may include a contact address for the store entity, as well as a location address for the store entity.

Store

A *store* is a store entity. A store must belong to a store group.

Store group

A *store group* is a collection of stores. A store group is a store entity. The store group acts as a container for common information, which can be stored at a store group level and shared by all the stores in the store group. For example, stores in the same store group can share information such as tax categories, supported languages, supported currencies, calculation codes, and shipping jurisdictions.

Currently, only one store group can exist and be maintained at the site administration level within a WebSphere Commerce Server.

For more detailed information on the structure of store assets in WebSphere Commerce Server, see the store object and data models in the WebSphere Commerce online help.

Creating store assets in WebSphere Commerce

The Store Services tools in WebSphere Commerce allow you to create or edit the following store assets:

- The store identifier and member identifier in the contact assets
- The store identifier in the STOREENT table
- The store directory in the STORE table
- The address nickname in the STADDRESS table
- The store description
- The store address

Note: The Store Services tools work with pre-populated XML files in the form of a store archive.

As a result, you have two options for creating store assets:

- Edit the existing store assets from one of the sample stores provided with WebSphere Commerce, or an existing store archive.
- Create store assets in the form of an XML file that can be published as part of a store archive, or loaded using the Loader package.

For information on editing the store assets in an existing store archive, see the WebSphere Commerce online help. For information on creating store assets in the form of an XML file, see “Creating store data assets in an XML file”.

Creating store data assets in an XML file

Create your store assets in the format of XML files that can be loaded into the database using the Loader package. If you are creating a multicultural store, you may want to create separate XML files for each locale your store supports. The locale-specific file should specify all description information, so it can be easily translated. For more information on the Loader package, see Part 7, “Publishing your store” on page 207.

The sample stores, from which many of the examples in these tasks are taken, use one `store.xml` file for all information that does not need to be translated, and another `store.xml` file for each locale the store supports, for the information that needs to be translated. The locale-specific files contain all the description information.

To create store assets, do the following:

1. Review the XML files used to create store assets for the sample stores. All files for the sample stores are located in the corresponding store archive file.

   The store archive files are located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
   - AIX `/usr/WebSphere/CommerceServer/samplestores`
   - Solaris `/opt/WebSphere/CommerceServer/samplestores`
   - Linux `/opt/WebSphere/CommerceServer/samplestores`
   - 400 `/qibm/proddata/WebCommerce/samplestores`

   Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

   Each sample store includes two `store.xml` files, which include the store information. To view the `store.xml` files in the store archive, decompress the store archive using a ZIP program. The `store.xml` files are located in the data directory. The language-specific `store.xml` is in a locale-specific subdirectory of the data directory.
2. Review the information in Appendix B, “Creating your data” on page 305.
3. Create a `store.xml` file, either by copying one of the `store.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `store.xml`. The DTD files 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`
   - Linux `/opt/WebSphere/CommerceServer/xml/sar`
   - 400 `/qibm/proddata/WebCommerce/xml/sar`
4. Create a store entity.

   a. Using the following example as your guide, define a store entity in your XML file for the STOREENT table.

   ```
   <storeent
    storeent_id="@storeent_id_1"
    member_id="&MEMBER_ID"
   ```

```
 type="S"
 identifier="ToolTech"
 setccurr="USD"
/>
```

where

- `storeent_id` is a generated unique key.
- `member_id` is the owner of the store entity.
- `type` is the kind of store entity: G = StoreGroup, S = Store.
- `identifier` is a string that, along with the owner, uniquely identifies the store entity.
- `setccurr` is the default currency for a store entity, in other words, the currency that will be used by a customer that does not already have a preferred currency. If it is NULL for a Store, the default currency is obtained from its store group.

5. Create a store address.
   a. Using the following example as your guide, create the store address or addresses in your XML file for the STADDRESS table. If you are creating a multicultural catalog, you should include this information in a locale-specific XML file.

```
<staddress
 staddress_id="@staddress_id_en_US_1"
 member_id="&MEMBER_ID"
 nickname="storeaddress_English"
 address1="12xx Martindale Avenue"
 address2="Suite 9xx"
 businesstitle="ToolTech"
 city="Toolsville"
 state="Ontario"
 zipcode="Lxx 1xx"
 country="Canada"
 phone1="1-800-555-1234"
 fax1="1-800-555-4321"
 email1="info@tooltech.xxx"
 />
```

   where

   - `staddress_id` is a generated unique key.
   - `member_id` is the owner of the store entity.

6. Create a description for the store entity.
   a. Using the following example as your guide, create the description of the store entity in your XML file for the STOREENTDS table. If you are creating a multicultural catalog, you should include this information in a locale-specific XML file.

```
<storeentds
 description="Commerce Models Store entity"
 language_id="&en_US"
 displayname="ToolTech"
 storeent_id="@storeent_id_1"
 staddress_id_cont="@staddress_id_en_US_1"
 staddress_id_loc="@staddress_id_en_US_1"
```

   where

   - `description` is a longer description of the store entity, suitable for display to customers.

- `language_id` is the default language for information displayed to customers shopping in the store.
- `displayname` is a brief description of the store entity, suitable for display to customers.
- `storeent_id` is the store entity.
- `staddress_id_cont` is the contact address of the StoreEntity.
- `staddress_id_loc` is the physical location of the StoreEntity.

7. Create a store in the database.
   a. Using the following example as your guide, define a store in your XML file in the STORE table.

```
<store
store_id="@storeent_id_1"
directory="ToolTech"
ffmcenter_id="@ffmcenter_id_1"
language_id="&en_US"
storegrp_id="-1"
allocationgoodfor="43200"
bopmpadfactor="0"
defaultbooffset="2592000"
ffmcselectionflags="0"
maxbooffset="7776000"
rejectedordexpiry="259200"
rtnffmctr_id="@ffmcenter_id_1"
pricerefflags="0"
storetype="B2B"
/>
```

where

- `store_id` is a generated unique key.
- `directory` is the directory in which store-specific Web assets are found. The actual location of these assets in the file system is based on the value of this column, plus several configuration parameters in the WebSphere Commerce configuration file: StoresDocRoot, StoresWebPath, and StoresPropertiesPath. For example, if the StoresDocRoot is `D:\WebSphere\wcs\stores`, StoresWebPath is `web`, StorePropertiesPath is `properties`, and the value of this column is `mystore`, then the JSP files will be located in the directory `D:\WebSphere\wcs\stores\web\mystore` and the property files will be located in `D:\WebSphere\wcs\stores\properties\mystore`.
- `ffmcenter_id` is the default fulfillment center for the store.
- `language_id` is the default language for information displayed to customers shopping in the store.
- `storegrp_id` is the store group the store is associated with. This number is generated in the STOREGRP table.
- `allocationgoodfor` means that the ReleaseExpiredAllocations scheduler job can be used to reverse ATP inventory allocations when this many seconds have passed since the allocations were made.
- `bopmpadfactor` means if this store calculates order amounts (such as tax or shipping charges) differently for different fulfillment centers, the order amount for a previously submitted order can change when fulfillment centers are finally allocated to backordered items. This padding factor represents a percentage by which the order amount presented to Payment Manager can be increased, if necessary. For example, specify 5 to allow an increase of up to 5 percent.

- `defaultbooffset` is after an estimated availability time cannot be determined for a backordered OrderItem, it will be set to this many seconds in the future.
- `maxbooffset` means if the estimated availability time for a backordered OrderItem would normally exceed this many seconds in the future, it will be set to this many seconds in the future.
- `rejectedordexpiry` are orders with payment in Declined state longer than this number of seconds and are candidates for cancellation.
- `rtnffmctr_id` is the default fulfillment center for returning merchandise to the store.
- `pricerefflags` contains bit flags that control which TradingAgreements and Offers are searched when prices are refreshed by the default implementation of the GetContractUnitPrices task command:
  - 1 = usePreviousOnly - Use the ones referenced by the OrderItems. Fail if they can no longer be used.
  - 2 = usePreviousOrSearchAgain - Same as usePreviousOnly, but instead of failing when they can no longer be used, search the ones saved in the ORDIOFFER and ORDITRD tables
  - 4 = alwaysSearchAgain - Always search the ones saved in the ORDIOFFER and ORDITRD tables.
- `storetype` indicates one of the following store types, for use by a user interface that provides appropriate functions depending on the StoreType: B2B = Business-to-Business. B2C = Business-to-Consumer.

8. Define a supported language for the store.
   a. Using the following example as your guide, define a supported language for your store in your XML file to add information to the STORELANG table. If your store supports multiple languages, you should include this information in a locale-specific XML file (one for each language your store supports).

```
<storelang
 language_id="&en_US"
 storeent_id="@storeent_id_1"
 />
```

   where
   - `language_id` is the language supported by the store entity.
   - `storeent_id` is the store entity.

   b. Using the following example as your guide, add information about the language to the STORELANGDS table. If your store supports multiple languages, you should include this information in a locale-specific XML file (one for each language your store supports).

```
<storlangds
  description="United States"
  language_id="&en_US"
  storeent_id="@storeent_id_1"
  language_id_desc="&en_US"
 />
```

   where
   - `description` is a brief description of the language, suitable for display to customers in a selection list.
   - `language_id` is the language of the description.
   - `storeent_id` is the store entity that supports the language.

- `language_id_desc` is the language being described.

For more information about the use of **@** and **&** see Appendix B, “Creating your data” on page 305.

Chapter 7. Commands, views, and URL registry data

When you create a WebSphere Commerce Server instance, the default commands, views, and URLs provided with WebSphere Commerce are registered in the WebSphere Commerce Server database in the corresponding tables: CMDREG, VIEWREG, and URLREG. These commands, views, and URLs are available for use in all stores residing in the instance.

WebSphere Commerce also provides default JSP files to display the default views. These JSP files are associated with the views in the VIEWREG table.

If you create new commands, views, or URLs, or customize existing ones, you must register them in the corresponding database tables (CMDREG, VIEWREG, and URLREG) before they are available for use in your store. If you create new JSP files for use in your store, you must associate them with the corresponding view in the VIEWREG table.

Note: If you create a new JSP file, but give it the same name as the default JSP file associated with the view, you do not need to register the new JSP file in the VIEWREG table.

For more information on creating or customizing command, views, or URLs, see the *IBM WebSphere Commerce Programmer's Guide*. The *Programmer's Guide* also contains information on how and when to register commands, views, URLs, and JSP files.

For more detailed information on the structure of command and view assets in the WebSphere Commerce Server, see the command and view data models in the WebSphere Commerce online help.

Registering commands, views, and URLs in WebSphere Commerce

If you create or customize multiple new commands, views, URLs, or JSP files for your store, you may want to register them using an XML file, which you can then load into the database using the Loader package, or as part of a store archive that can be published using Store Services. For more information on the Loader package, see Part 7, "Publishing your store" on page 207.

Note: Before creating an XML file to load new or customized commands, refer to the *Programmer's Guide*for more detail on how commands work.

Creating an XML file to register commands, views, and URLs

To create an XML file to register the new commands, views, and JSP files for your store, do the following:

1. Review the XML files used to register commands, views, JSP files for the sample stores. Each sample store includes a `command.xml` file, which includes the registration information. The store archive files are located in the following directory:
   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`

  - AIX `/usr/WebSphere/CommerceServer/samplestores`
  - Solaris `/opt/WebSphere/CommerceServer/samplestores`
  - Linux `/opt/WebSphere/CommerceServer/samplestores`
  - 400 `/qibm/proddata/WebCommerce/samplestores`

  Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

  To view the contents of the store archive, use a decompression program. The `command.xml` file is located in the `data` directory.

2. Review the information in Appendix B, "Creating your data" on page 305.
3. Create a `command.xml` file, either by copying one of the `command.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `command.xml`. The DTD files 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`
  - Linux `/opt/WebSphere/CommerceServer/xml/sar`
  - 400 `/qibm/proddata/WebCommerce/xml/sar`
4. Controller commands must be registered in the URLREG table and the CMDREG table. To register a new or customized controller command in the URLREG table, create an entry in the XML file for each new customized controller command, using the following example as your guide:

```
<urlreg
url="MyProductDisplay"
storeent_id="@storeent_id_1”
interfacename="com.mystore.commerce.catalog.commands.ProductDisplayCmd"
https="0"
description="Product display command for my store"
authenticated="0"
internal="0" />
```

  where
  - `urlreg` is the name of the database table (URLREG) that this information will populate.
  - `url` is the URI name
  - `storeent_id` is the store entity identifier and the use of the @ symbol is known as internal-alias resolution. When using internal-alias resolution, an alias is substituted in place of the primary key (identifier) in the XML document. This alias is then used elsewhere in the XML file to refer to that element. This eliminates the need to know the unique indexes necessary to build the XML file. During publish, the ID Resolver replaces the @ symbol with a unique value. For more information, see Appendix B, "Creating your data" on page 305.
  - `interfacename` is the controller command interface name

- `https` is the secure HTTP required for this URL request. Use 1 when secure HTTP is required and 0 when it is not.
- `authenticated` is whether or not user log on is required for this URL request. Use 1 when authentication is required and 0 when it is not.
- `internal` indicates whether the command is internal to WebSphere Commerce. URLs that are internal are used by WebSphere Commerce tools. Use 1 when it is internal and 0 when it is external. URLs you create should be external.

5. To register a new controller command, or a new task command, in the CMDREG table, create an entry in the XML file for each new or customized controller or task commands, using the following example of a task command (from the ToolTech sample store `command.xml` file) as your guide:

   ```
   < cmdreg
   storeent_id="@storeent_id_1"
   interfacename="com.ibm.commerce.payment.commands.DoPaymentCmd"
   classname="com.ibm.commerce.payment.commands.DoPaymentMPFCmdImpl"/>
   ```

   where
   - `cmdreg` is the name of the database table (CMDREG) that this information will populate.
   - `storeent_id` is the store entity identifier and the use of the @ symbol is known as internal alias resolution. When using internal-alias resolution, an alias is substituted in place of the primary key (identifier) in the XML document. This alias is then used elsewhere in the XML file to refer to that element. This eliminates the need to know the unique indexes necessary to build the XML file. During publish, the ID Resolver replaces the @ symbol with a unique value. For more information, see Appendix B, “Creating your data” on page 305.
   - `interfacename` is the command interface name
   - `classname` is the command implementation class name. Typically, this name is the interface name with Impl appended at the end.

6. To register new views, or to associate new JSP files with a view, create an entry in the VIEWREG table, using the following example (from the ToolTech sample store `command.xml` file) as your guide:

   ```
   <viewreg
   viewname="OrderOptionsView"
   devicefmt_id="-1"
   storeent_id="@storeent_id_1"
   interfacename="com.ibm.commerce.command.ForwardViewCommand"
   classname="com.ibm.commerce.command.HttpForwardViewCommandImpl"
   properties="docname=Shipping.jsp"
   internal="0"
   https="0"/>
   ```

   where
   - `viewreg` is the name of the database table (VIEWREG) that this information will populate.
   - `viewname` is the name of the view.
   - `devicefmt_id` is the type of device on which this view will be used, for example, a browser.

- `storeent_id` is the store entity identifier and the use of the @ symbol is known as internal-alias resolution. When using internal-alias resolution, an alias is substituted in place of the primary key (identifier) in the XML document. This alias is then used elsewhere in the XML file to refer to that element. This eliminates the need to know the unique indexes necessary to build the XML file. During publish, the ID Resolver replaces the @ symbol with a unique value. For more information, see Appendix B, "Creating your data" on page 305.
- `interfacename` is the view command interface name. Default options are ForwardView, DirectView, and RedirectView.
- `classname` is the view implementation class name. Typically, this name is the interface name with Impl appended at the end.
- `properties` is the default name-value pairs set as input properties to the command. If the same page is always displayed set the JSP file name in this property, for example, docname=Shipping.jsp.
- `internal` indicates whether the view is internal to WebSphere Commerce. Internal views are used by WebSphere Commerce tools. Use 1 when it is internal and 0 when it is external. Views you create should be external.
- `https` is the secure HTTP required for this URL request. Use 1 when secure HTTP is required and 0 when it is not.

For more information about the use of @ and & see Appendix B, "Creating your data" on page 305.

Chapter 8. Catalog assets

Like a traditional catalog, your online catalog consists of the goods and services you offer for sale. Although the size and structure of online catalogs can differ greatly from store to store, depending on the type and amount of merchandise available for purchase, catalogs require the following:

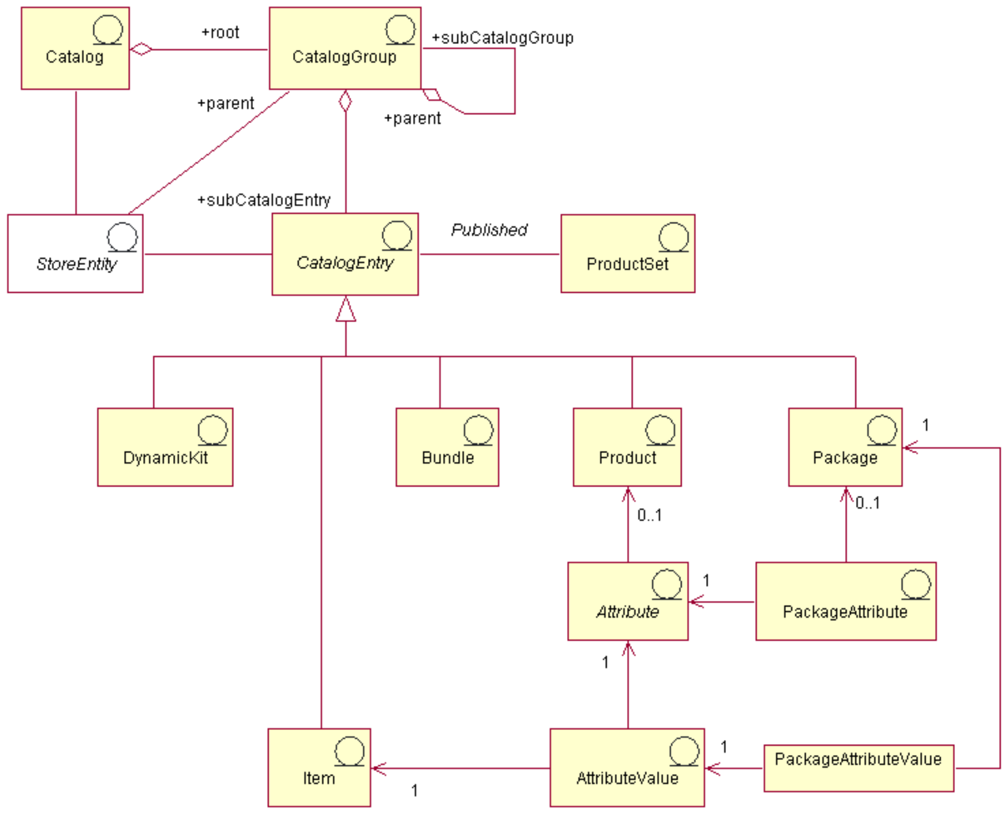
- What you are selling, including
  - Prices, which are almost always included in an online catalog.
  - Product data, such as descriptions and images of your merchandise.
  - Categories, as most, but not all catalogs divide merchandise into categories, to facilitate navigation for customers.
- A display method for what you are selling. Catalog display pages outline how a page looks to your customers and provide a consistent look and feel between various catalog pages. How you structure your catalog depends on your merchandise.

Understanding catalogs in WebSphere Commerce

WebSphere Commerce places several requirements on your store's online catalog. Every store in the WebSphere Commerce system must have a master catalog. The master catalog is the central location to manage your store's merchandise. It is the single catalog containing all products, items, relationships, and standard prices for everything that is for sale in your store.

You can share the master catalog across stores and define as many stores as needed. In addition to creating a master catalog for catalog management, you may also choose to create one or more navigational catalogs for display purposes. A navigational catalog may contain the same entries as the master catalog, but will have a much more flexible structure for customer display purposes. You can have as many navigational catalogs as you want. However, since it is the master catalog that is used to manage your online merchandise, we recommend that you also use the master catalog as your navigational catalog to minimize maintenance overhead.

If you are creating a new master catalog for use with a WebSphere Commerce store, or if you are using an existing master catalog available from a WebSphere Commerce sample store, such as ToolTech, you will have to modify your catalog to meet these requirements. The following diagram outlines the basic structure of a catalog in WebSphere Commerce.



This diagram, and all others in the store data section are part of the WebSphere Commerce information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

Catalogs

The *catalog* is the starting point for the information model. The catalog contains all hierarchical and navigational information for the online catalog and is a collection of catalog groups and catalog entries that are available for display and purchase at an online store.

In WebSphere Commerce, a catalog is represented in the database by a *catalog entity*. A catalog entity consists of a unique catalog ID and a description of the catalog, for example, the catalog name. Since each catalog is a separate, unique entity, it can easily be associated with one or more stores. Every store in the WebSphere Commerce system must be related to at least one catalog entity. The master catalog is a special catalog that contains all of the items that are for sale in your online store.

Catalog groups

Catalog groups are generic groupings of your catalog entries, created for navigational and catalog partitioning purposes. A catalog group belongs to a catalog and may contain more than one catalog group or catalog entries. You can associate catalog groups to more than one catalog. A catalog group is also known as a *category*.

A flat catalog is a catalog that does not group its products in categories; instead, it displays a list of products. Although it is possible to create a flat catalog in WebSphere Commerce, it is recommended that you create catalog groups for structural and navigational purposes.

When creating catalog groups, you must first arrange your catalog in a hierarchy, or inverted tree. The tree begins at a general catalog group (called the root category), and branches out into increasingly specific subcategories until it cannot be further divided. Each lowest level catalog group, which contains only products, is a leaf. A catalog group is the parent to the categories immediately below it, and a child of the one above. For example, Men's Fashion is a grouping of the men's apparel categories, while the catalog groups Pants and Shirts are groupings of products.

Catalog entries

Each catalog group contains *catalog entries*. Catalog entries represent orderable merchandise in an online catalog. The entries typically have a name or part number, a description, one or more prices, images, and other details. A catalog entry can be a product, item, package, bundle, or dynamic kit. If necessary, you can create new catalog entry types that do not fit into one of the five existing models. More information on each type of catalog entry is available below.

Products

A *product* is a type of catalog entry. A product acts as a template for a group of items or SKUs that exhibit the same attributes. For example, a shirt is a product in your catalog. After adding attributes and attribute values to the shirt, each variation becomes an item, such as a small black shirt.

Items

An *item* is a tangible unit of merchandise that has a specific name, part number, and price. For example, a small black shirt is an item while a shirt is a product. All items related to a particular product exhibit the same set of attributes and are distinguished by their attribute values.

Note: For WebSphere Commerce Accelerator users, items and *SKUs* are considered synonymous. When using the Product Management tools in the WebSphere Commerce Accelerator, the orderable item is called a SKU. In the WebSphere Commerce database schema, this particular type of catalog entry is called an item.

Packages

A *package* is an atomic collection of catalog entries. For example, a computer package might contain a specific central processing unit, monitor, and hard drive that cannot be sold separately. Similar to a product, a package has defining attributes and is a container for fully resolved packages. A fully resolved package is comparable to a SKU. A package has its own price and is an actual orderable SKU that can be added to a shopping cart. You cannot decompose or modify a package either during navigation or after the package has been placed in the shopping cart.

Bundles

A *bundle* is a collection of catalog entries to allow customers to buy multiple items at once. For example, a bundle for a computer might be composed of a central processing unit, a monitor, a hard drive, and a CD-ROM drive. A bundle is a grouping of items, or a combination of products, items, and fully resolved packages. If you select a bundle which only contains items, the bundle is decomposed into separate orderable SKUs that are added individually to the shopping cart. However, if you select a bundle which contains products, these products need to be resolved into items through SKU resolution before they can be added to a shopping cart. In either case, once a bundle is decomposed and its component items are added to a shopping cart, you can modify or remove each item.

Dynamic kits

A *dynamic kit* is a type of catalog entry which can be dynamically configured by the customer. This configuration (or grouping) of products is based on the customer's requirements and is sold as a single unit. The components of a dynamic kit are composed by an external product configurator through a set of predefined rules and user interaction. Adding a dynamic kit to an order is similar to adding a package. Like a package, the individual components of a dynamic kit cannot be modified and the entire configuration must be fulfilled as a whole. However, you may change the dynamic kit components by reconfiguring it using an external product configurator.

Product sets

Product sets are associated with published CatalogEntry objects. A product set provides a mechanism to partition your catalog into logical subsets. This partitioning allows you to show different parts of your catalog to different users. You can create a contract and specify that the participants of the contract are only entitled to purchase products that fall into a predefined product set. WebSphere Commerce provides tools to create product sets from the master catalog and to make use of them in contracts for entitlement filtering.

Attributes

Attributes are properties of products in an online store, such as color or size. An attribute belongs to a product. Each possible combination of attributes and attribute values defines an item.

Attribute values

Attribute values are properties of an attribute such as a specific color (blue or yellow) or size (small, medium, or large). You must predefine attribute values before assigning them to items. Each possible combination of attributes and attribute values defines an item.

Package attributes

Package attributes are inherited from the attributes of the products that are contained within packages. A package can have zero or more package attributes. A package attribute refers to one attribute.

Package attribute values

Package attribute values are the values assigned to package attributes. Package attribute values are inherited from the attribute values of the products that are contained within packages. A package attribute value refers to one attribute value.

For more detailed information on the structure of catalog assets in WebSphere Commerce, see the catalog data models in the WebSphere Commerce online help.

Creating catalog assets in WebSphere Commerce

To create the catalog assets for your store, you need to create a master catalog by adding information to several WebSphere Commerce database tables. You can create your catalog using XML files that are loaded into the database by the Loader package. If you are creating a multicultural catalog, you will need separate XML files for each locale your store supports. Each locale specific XML file adds the translatable information, such as descriptions, for your catalog, catalog groups, and catalog entries.

The following is an overview of the catalog creation process:

1. In WebSphere Commerce, a catalog is created using XML files. Creating a catalog begins with a catalog entity, your database's equivalent of a paper catalog.
2. Create the catalog structure and navigation by adding catalog groups to determine the categories and layout of your merchandise.
3. Create inventory information as a base for the catalog entries.
4. Add your merchandise in the form of catalog entries, which represent products, SKUs, packages, bundles, and dynamic kits.
5. Attributes and attribute values are added to your catalog's products to distinguish the different SKUs from one another.
6. You can create packages and bundles to group certain catalog entries together for promotional purposes.
7. The relationships between the catalog groups and catalog entries are created next. This determines which entries belong to a catalog group.
8. You can create merchandising associations for your catalog entries as product recommendation strategies.
9. Associate your catalog, catalog groups, and catalog entries to your WebSphere Commerce store.
10. In the final steps, you need to create:
    a. taxes for your merchandise,
    b. shipping methods,
    c. a fulfillment center to act as an inventory warehouse and a shipping and receiving center. A store can have more than one fulfillment center defined,
    d. prices for your merchandise.

Creating a master catalog

To create a master catalog that contains multiple levels of categories, complete the following tasks:

Part 1: Preparing for catalog creation

1. Review the catalog information and its corresponding object and data models within WebSphere Commerce. The catalog information is a component of the WebSphere Commerce Server that provides online catalog navigation, partitioning, categorization, and associations for orderable merchandise.
2. Review the WebSphere Commerce Loader package information. The Loader package consists primarily of utilities for preparing and loading data into a WebSphere Commerce database. You can use the Loader package to load large amounts of data and to update data in your database. For more information on the Loader package, see Part 7, "Publishing your store" on page 207.

3. Review the information in Appendix B, “Creating your data” on page 305.
4. Create an organization through the Administration Console to act as the catalog owner. For more information, see the WebSphere Commerce online help topic ″Creating an organization″.
5. Create a new XML file for your master catalog by using the existing XML entries and `catalog.xml` files from the ToolTech sample store as your guide. If you are creating a multicultural catalog, create a separate `catalog.xml` file for each locale your store supports. The locale-specific file should specify all description information, so it can be easily translated. In this example, one `catalog.xml` file will be used for all information that does not need to be translated, and a second `catalog.xml` will be used for each locale the store supports and will include the information that needs to be translated. Or, if you prefer, you can use the existing XML file from the ToolTech sample store and change the information as needed. The `catalog.xml` files from the ToolTech sample store are located in its store archive file. To view the `catalog.xml` files, decompress the store archive using a ZIP program. The `catalog.xml` files are located in the following data directories:
   - NT `*drive*:\WebSphere\CommerceServer\samplestores`
   - 2000 `*drive*:\Program Files\WebSphere\CommerceServer\samplestores`
   - AIX `/usr/WebSphere/CommerceServer/samplestores`
   - Solaris Linux `/opt/WebSphere/CommerceServer/samplestores`
   - 400 `/qibm/proddata/WebCommerce/samplestores`

   Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

   The `catalog.dtd` file is 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 Linux `/opt/WebSphere/CommerceServer/xml/sar`
   - 400 `/qibm/proddata/WebCommerce/xml/sar`

Part 2: Creating a catalog entity

1. Using the following example from the ToolTech sample store as your guide, create a catalog entity by adding information to the CATALOG and CATALOGDSC tables. A catalog entity represents a catalog in the database.
   ```
   <catalog
   catalog_id="@catalog_id_1"
   member_id="&MEMBER_ID;"
   identifier="ToolTech"
   description="ToolTech Catalog"
   tpclevel="0"
   />
   ```

   where
   - `catalog_id` is the internal reference number.
   - `member_id` is the internal reference number that identifies the owner of the catalog.

- `identifer` is an external name for the catalog.
- `description` is a description of the catalog.

2. Using the following example from the ToolTech sample store as your guide, add the catalog's description in the locale-specific XML file for translation purposes:

```
<catalogdsc
catalog_id="@catalog_id_1"
language_id="&en_US;"
name="Store master catalog"
/>
```

where

- `catalog_id` is the internal reference number relating this language specific information to a catalog.
- `language_id` is the identifier of the language.
- `name` is the language-dependent name of the catalog.

Part 3: Creating catalog groups

1. Using the following example from the ToolTech sample store as your guide, create catalog groups by adding information to the CATGROUP and CATGRPDESC tables. Catalog groups, also known as categories, are groupings of other catalog groups or products. Complete this task for each catalog group in your catalog:

```
<catgroup
catgroup_id="@catgroup_id_1"
member_id="&MEMBER_ID;"
identifier="Woodworking"
markfordelete="0"
/>
```

where

- `catgroup_id` is the internal reference number of the catalog group
- `member_id` is the internal reference number that identifies the owner of the catalog.
- `identifer` is an external name for the catalog.
- `markfordelete` indicates whether the catalog group has been marked for deletion:
  - 0 = No.
  - 1 = Yes.

2. Using the following example from the ToolTech sample store as your guide, add the catalog group's description in the locale-specific XML file for translation purposes. Complete this task for each catalog group in your catalog:

```
<catgrpdesc
language_id="&en_US;"
catgroup_id="@catgroup_id_1"
name="Woodworking"
shortdescription="Woodworking"
longdescription="Woodworking"
published="1"
/>
```

where

- `language_id` is the identifier of the language.
- `catgroup_id` is the internal reference number of the catalog group.

- name is language-dependent name of the catalog.
- shortdescription is a brief description of the catalog group.
- longdescription is a detailed description of the catalog group.
- published indicates whether this catalog group should be displayed for the language indicated by language_id:
  - 0 = No.
  - 1 = Yes.

Note: Each time you create a catalog group and its description, the catgroup_id changes to represent a new catalog group. For example, catgroup_id="@catgroup_id_2" , catgroup_id="@catgroup_id_3" , and catgroup_id="@catgroup_id_4", and so on.

3. After creating your catalog groups, assign a top-level catalog group to the catalog by adding information to the CATTOGRP table. This catalog group is the parent to the catalog groups immediately below it. Complete this task for each top-level catalog group in your catalog. Use the following example from the ToolTech sample store as your guide:

```
<cattogrp
catalog_id="@catalog_id_1"
catgroup_id="@catgroup_id_1"
/>
```

   where
   - catalog_id is the reference number of the catalog.
   - catgroup_id is the reference number of the catalog group.

   Note: Each time you assign top-level catalog groups to the catalog, the catgroup_id is modified to represent a new catalog group association. For example, catgroup_id="@catgroup_id_2" , catgroup_id="@catgroup_id_3" , and catgroup_id="@catgroup_id_4", and so on.

4. Once the parent and child structure has been determined for your catalog groups, create relationships between the catalog groups by adding information to the CATGRPREL table. Complete this task for each parent and child catalog group structure in your catalog. Use the following example from the ToolTech sample store as your guide:

```
<catgrprel
catgroup_id_parent="@catgroup_id_1"
catgroup_id_child="@catgroup_id_11"
catalog_id="@catalog_id_1"
sequence="0"
/>
```

   where
   - catgroup_id_parent is the source catalog group of this relationship.
   - catgroup_id_child is the target catalog group of this relationship.
   - catalog_id is the reference number of the catalog.
   - sequence is the number that determines the display order of the contents of the catalog group.

   Note: With each catalog group relationship, the catgroup_id_child and the sequence is modified to represent a new relationship. For example, subsequent relationships would be displayed as catgroup_id_child="@catgroup_id_12" and sequence="1", and catgroup_id_child="@catgroup_id_13" and sequence="2", and so on. If you are not using a navigational structure in your catalog, then you can remove the CATGRPREL relationship.

Part 4: Creating inventory information

1. Using the following example from the ToolTech sample store as your guide, create inventory information by adding information to the BASEITEM, BASEITEMDSC, ITEMSPC, ITEMVERSN, VERSIONSPC, DISTARRANG, and STOREITEM tables. Begin by creating base items by adding information to the BASEITEM table. Base items represent a general family of products with a common name and description. Complete this task for each group of inventory items in your catalog:

   ```
   <baseitem
   baseitem_id="@baseitem_id_102"
   member_id="&MEMBER_ID;"
   markfordelete="0"
   partnumber="tooltech_sku_102"
   itemtype_id="ITEM"
   quantitymeasure="C62"
   quantitymultiple="1.0"
   />
   ```

   where
   - `baseitem_id` is the generated unique key.
   - `member_id` is the owner of the base item.
   - `markfordelete` indicates whether the base item is marked for deletion:
     - 0 = No.
     - 1 = Yes.
   - `partnumber` uniquely identifies the base item for the owner.
   - `itemtype_id` is the type of base item:
     - ITEM = items, packages, or bundles
     - DNKT = dynamic kits.
   - `quantitymeasure` is the unit of measure for the quantity multiple.
   - `quantitymultiple` is the amount of the base item that is measured in integral units. Along with `quantitymeasure`, this indicates how much each integral unit represents.

   Note: You must create a base item for every product that you create in your catalog. Each time you create a base item, the `baseitem_id` and `partnumber` numbers change to create a new base item. For example, a new base item would contain `baseitem_id="@baseitem_id_147"` and `partnumber="tooltech_sku_147"` as entries, while another base item would contain `baseitem_id="@baseitem_id_192"` and `partnumber="tooltech_sku_192"` as entries, and so on.

2. Using the following example from the ToolTech sample store as your guide, add information about specified items to the database. A specified item is an item with values for all its attributes, and represents an item, package, bundle, or dynamic kit in the catalog. Complete this task for each specified item in your catalog:

   ```
   <itemspc
   itemspc_id="@itemspc_id_106"
   baseitem_id="@baseitem_id_102"
   markfordelete="0"
   ```

```
partnumber="T0000106"
member_id="&MEMBER_ID;"
discontinued="N"
/>
```

where

- `itemspc_id` is the generated unique key.
- `baseitem_id` is the product base item.
- `markfordelete` indicates whether the specified item is marked for deletion:
  - 0 = No.
  - 1 = Yes.
- `partnumber` uniquely identifies the specified item for the owner.
- `member_id` is the owner of the specified item.
- `discontinued` indicates whether the specified item has been discontinued:
  - Y = Discontinued and can be ordered if there is sufficient inventory but it cannot be backordered.
  - N = Active and may be backordered if out of stock.

Note: You must create a specified item for each item that you create in your catalog. Each time you define a specified item, the `itemspc_id="@itemspc_id_107"`, `baseitem_id="@baseitem_id_102"`, `partnumber="T0000107"` numbers change to create a new specified item. For example, a new specified item would contain `itemspc_id="@itemspc_id_108"`, `baseitem_id="@baseitem_id_102"`, and `partnumber="T0000108"` as entries, while another specified item would contain `itemspc_id`, `baseitem_id`, and `partnumber` as entries, and so on.

3. Using the following example from the ToolTech sample store as your guide, add the following information for a relationship between an item version and a base item to the database. Complete this task for each such relationship in your catalog:

```
<itemversn
itemversn_id="@itemversn_id_102"
baseitem_id="@baseitem_id_102"
expirationdate="2010-01-01 00:00:00.000000"
versionname="version"
/>
```

where

- `itemversn_id` is a generated reference number which identifies the item version.
- `baseitem_id` is the base item.
- `expirationdate` is the time the item version expires.
- `versionname` uniquely identifies the item version for its base item.

Note: Each time you create a relationship between an item version and a base item, the `itemversn_id` and `baseitem_id` numbers change to create a new relationship. `baseitem_id` matches an existing base item. For example, a new relationship would contain `itemversn_id="@itemversn_id_107"` and `baseitem_id="@baseitem_id_107"` as entries, while another relationship would contain `itemversn_id="@itemversn_id_108"` and `baseitem_id="@baseitem_id_108"` as entries, and so on.

4. Using the following example from the ToolTech sample store as your guide, add the following information for a relationship between a product version and a specified item to the database. Complete this task for each such relationship in your catalog:

   ```
   <versionspc
   versionspc_id="@versionspc_id_106"
   itemspc_id="@itemspc_id_106"
   itemversn_id="@itemversn_id_102"
   />
   ```

   where
   - `versionspc_id` is the generated unique identifier.
   - `itemspc_id` is the specified item that the catalog entry relates to.
   - `itemversn_id` identifies the item version.

   Note: Each time you create a relationship between a product version and a specified item, the `versionspc_id` and `itemspc_id` numbers change to create a new relationship. `itemspc_id` matches an existing specified item. For example, a new relationship would contain `versionspc_id="@versionspc_id_107"` and `itemspc_id="@itemspc_id_107"` as entries, while another relationship would contain `versionspc_id="@versionspc_id_108"` and `itemspc_id="@itemspc_id_108"` as entries, and so on.
5. Using the following example from the ToolTech sample store as your guide, add the distribution arrangements to the database. A distribution arrangement enables a store to sell its own inventory. Complete this task for each distribution arrangement in your catalog:

   ```
   <distarrang
   distarrang_id="@distarrang_id_102"
   wholesalestore_id="@storeent_id_1"
   merchantstore_id="@storeent_id_1"
   baseitem_id="@baseitem_id_102"
   pickingmethod="F"
   startdate="2000-12-25 00:00:00.000000"
   enddate="2010-01-01 00:00:00.000000"
   />
   ```

   where
   - `distarrang_id` is the reference number of the distribution arrangement.
   - `wholesalestore_id` is the wholesale store that owns the inventory that can be sold by the merchant store. This wholesale store must be the same as `merchantstore_id`.
   - `merchantstore_id` is the merchant store that can sell from the inventory of the wholesale store. This merchant store must be the same as `wholesalestore_id`.
   - `baseitem_id` is the product covered by the distribution arrangement.
   - `pickingmethod` determines the sequence in which inventory is picked from the RECEIPT table under this arrangement:
     - F = FIFO (First In First Out) - the least recently received inventory.
     - L = LIFO (Last in First Out) - the most recently received inventory.
   - `startdate` is the time the distribution arrangement starts being effective.
   - `enddate` is the time the distribution arrangement stops being effective.

Note: Each time you create a distribution arrangement, the `distarrang_id` and the `baseitem_id` numbers change to create a new distribution arrangement. For example, a second distribution arrangement might contain the values `distarrang_id="@distarrang_id_147"` and `baseitem_id="@baseitem_id_147"`, while a third might contain `distarrang_id="@distarrang_id_192"` and `baseitem_id="@baseitem_id_192"`, and so on.

6. Using the following example from the ToolTech sample store as your guide, add the attributes that affect how a particular store allocates inventory for the specified items of a particular base item to the database. Complete this task for each base item in your catalog:

```
<storeitem
baseitem_id="@baseitem_id_102"
storeent_id="@storeent_id_1"
trackinventory="Y"
forcebackorder="N"
releaseseparately="N"
returnnotdesired="N"
backorderable="Y"
creditable="Y"
minqtyforsplit="0"
/>
```

where

- `baseitem_id` is the base item.
- `storeent_id` is the store or the store group.
- `trackinventory` controls whether or not inventory is tracked in the RECEIPT table:
  - N = Inventory is not tracked and there are no entries in the RECEIPT table.
  - Y = Inventory is tracked in the RECEIPT table.
- `forcebackorder` temporarily suspends allocation of specified items for the base item:
  - N = Inventory can be allocated (normal behavior).
  - Y = Inventory cannot be allocated, even if there is enough inventory.
- `releaseseparately` controls how specified order items for the base item are released:
  - N = order items may be released along with other order items.
  - Y = order items must be released separately (in their own boxes).
- `returnnotdesired` indicates that an item return is not wanted (for example, perishable food items), even if customer is willing or able to return it:
  - N = Request for credit evaluated based on the customer’s intention to return the item, but the return is not expected.
  - Y = Request for credit evaluated as if return is expected.
- `backorderable` indicates that specified items for the base item cannot be backordered:
  - N = Items may not be backordered.
  - Y = Items may be backordered.
- `creditable` indicates whether the merchant will, without an override, issue a credit for this item:
  - N = Sold as-is.
  - Y = Creditable.

- `minqtyforsplit` indicates that order items will not be automatically split during inventory allocation if the remaining unallocated quantity in the new order item would be less than the specified minimum quantity.

Note: Each time you define the inventory allocation rules for a store item, the `baseitem_id` number changes to represent a new base item. For example, a new allocation might contain `baseitem_id="@baseitem_id_147"` while a third might contain `baseitem_id="@baseitem_id_192"`, and so on.

7. Using the following example from the ToolTech sample store as your guide, add the base item description to the locale-specific XML file for translation purposes. Complete this task for each base item description in your catalog:

```
<baseitmdsc
baseitem_id="@baseitem_id_102"
language_id="&en_US;"
shortdescription="Circular Saw"
longdescription="Light on weight but not in quality. The Circular Saw
weighs a maximum of 10.9lbs., with a choice of a 12 or 14 amp motor,
and speeds of up to 600 rpms! Low friction 220V aluminum alloy shoe
will ensure the job gets done on time."
/>
```

where

- `baseitem_id` is the generated unique key.
- `language_id` is the language of this information.
- `shortdescription` is a brief description of the base item.
- `longdescription` is a detailed description of the base item.

Part 5: Creating catalog entries

1. Using the following example from the ToolTech sample store as your guide, create catalog entries by adding information to the CATENTRY and CATENTDESC tables. Each type of catalog entry — products, items, packages, bundles, and dynamic kits — represents the orderable pieces of merchandise for sale in your catalog. You need to define a base item for each product catalog entry. Complete this task for each product catalog entry in your catalog:

```
<catentry
catentry_id="@product_id_102"
baseitem_id="@baseitem_id_102"
member_id="&MEMBER_ID"
catenttype_id="ProductBean"
partnumber="T0000102"
mfpartnumber="Sprain-Tools-102"
mfname="Sprain Tools"
markfordelete="0"
buyable="1"
/>
```

where

- `catentry_id` is the internal reference number of the product catalog entry.
- `baseitem_id` is the base item that the catalog entry relates to.
- `member_id` is the reference number that identifies the catalog entry.
- `catenttype_id` identifies the type of catalog entry:
  - ItemBean = identifies an item.
  - ProductBean = identifies a product.
  - PackageBean = identifies a package.
  - BundleBean = identifies a bundle.

- DynamicKitBean = identifies a dynamic kit.

- `partnumber` is the reference number that identifies the part number of the catalog entry.
- `mfpartnumber` is the part number used by the manufacturer to identify the catalog entry.
- `mfname` is the name of the manufacturer of the catalog entry.
- `markfordelete` indicates whether the catalog entry is marked for deletion:
  - 0 = No.
  - 1 = Yes.
- `buyable` indicates whether you can purchase the catalog entry individually:
  - 0 = No.
  - 1 = Yes.

  Note: Each time you add a base item to a product catalog entry, the `catentry_id` and the `baseitem_id` sequence changes to represent a new catalog entry. The `catenttype_id` changes depending on the type of catalog entry.
- Using the following example from the ToolTech sample store as your guide, define a specified item for each catalog entry. Complete this task for each catalog entry in your catalog:

```
<catentry
catentry_id="@catentry_id_106"
itemspc_id="@itemspc_id_106"
member_id="&MEMBER_ID"
catenttype_id="ItemBean"
partnumber="T0000106"
mfpartnumber="Sprain-Tools-106"
mfname="Sprain Tools"
markfordelete="0"
buyable="1"
/>
```

  where
  - `catentry_id` is the internal reference number of the catalog entry.
  - `itemspc_id` is the specified item that the catalog entry belongs to.
  - `member_id` is the reference number that identifies the catalog entry.
  - `cattentype_id` identifies the type of catalog entry:
    - ItemBean = identifies an item.
    - ProductBean = identifies a product.
    - PackageBean = identifies a package.
    - BundleBean = identifies a bundle.
    - DynamicKitBean = identifies a dynamic kit.
  - `partnumber` is the reference number that identifies the part number of the catalog entry.
  - `mfpartnumber` is the part number used by the manufacturer to identify the catalog entry.
  - `mfname` is the name of manufacturer of the catalog entry.
  - `markfordelete` indicates whether the catalog entry is marked for deletion:
    - 0 = No.
    - 1 = Yes.

– `buyable` indicates whether you can purchase the catalog entry individually:
  - 0 = No.
  - 1 = Yes.

Note: Each time you add a specified item to a catalog entry, the `catentry_id` and the `itemspc_id` sequence changes to represent a new catalog entry. The `catenttype_id` changes depending on the type of catalog entry. Under the master catalog structural restriction, a catalog entry cannot belong to more than one category. To place a catalog entry in more than one category, you must use a navigational catalog.

- Using the following example from the ToolTech sample store as your guide, add the description to the locale-specific XML file. Complete this task for each catalog entry description in your catalog:

```
<catentdesc
catentry_id="@product_id_102"
language_id="&en_US"
name="Circular"
shortdescription="Circular Saw"
longdescription="Light on weight but not in quality. The Circular Saw
weighs a maximum of 10.9lbs., with a choice of a 12 or 14 amp motor,
and speeds of up to 600 rpms! Low friction 220V aluminum alloy shoe
will ensure the job gets done on time."
thumbnail="images/circular_saw_sm.gif"
fullimage="images/circular_saw.gif"
available="1"
published="1"
/>
```

where

– `catentry_id` is the internal reference number that indicates the catalog entry that this language-specific information relates to.
– `language_id` is the identifier of the language.
– `name` is the language-dependent name of the catalog entry.
– `shortdescription` is a brief description of the catalog entry.
– `longdescription` is a detailed description of the catalog entry.
– `thumbnail` is the path for the thumbnail image.
– `fullimage` is the path for the full image.
– `available` indicates the length of time to availability of the catalog entry.
– `published` indicates whether this catalog entry should be displayed for the language indicated by `language_id`
  - 0 = Display.
  - 1 = Not display.

Part 6: Creating attributes and attribute values

1. Using the following example from the ToolTech sample store as your guide, create attributes and attribute values for your products by adding information to the ATTRIBUTE and ATTRVALUE tables in the locale-specific XML file for translation purposes. Each product in your catalog has a specific set of attributes, such as size and color for a shirt or a pair of pants. Items are defined by the attribute values. For example, while a shirt is a product, a medium, black shirt is an item. Complete this task for each attribute in your catalog:

```
<attribute
attribute_id="@attribute_id_103"
language_id="&en_US"
```

```
attrtype_id="STRING"
name="Amps"
sequence="0"
description="Amps"
catentry_id="@product_id_102"
description2="Amps"
/>
```

where
- `attribute_id` is the internal reference number of the attribute.
- `language_id` is the language that this attribute pertains to.
- `attrtype_id` is the type of the corresponding attribute value.
- `name` is the name of the attribute.
- `sequence` is a sequence number that determines the display order of attributes for a given product.
- `description` is the description of the attribute.
- `catentry_id` is the reference number of the product to which this attribute belongs.
- `description2` is an additional description of the attribute.

Note: Each time you add an attribute to a product defined by `catentry_id`, the `attribute_id` sequence changes to represent a new attribute.

2. Using the following example from the ToolTech sample store as your guide, add the attribute values. Complete this task for each attribute value in your catalog:

```
<attrvalue
attrvalue_id="@attrvalue_id_114"
language_id="&en_US"
attribute_id="@attribute_id_103"
name="12.0amps"
attrtype_id="STRING"
stringvalue="12.0amps"
sequence="0"
catentry_id="@catentry_id_106"
/>
```

where
- `attrvalue_id` is the internal reference number of attribute value
- `language_id` is the language that this attribute value pertains to
- `attribute_id` is the internal reference number of the attribute associated with the value
- `name` is the name of the attribute value
- `attrtype_id` is the type of attribute value
- `stringvalue` is the attribute value
- `sequence` is a sequence number that determines the display order of attribute values for a given attribute
- `catentry_id` is the item ID that this attribute value describes

Note: Each time you add an attribute value to an attribute, the `attrvalue_id` sequences changes to represent different values. The `attribute_id` sequence changes to represent a different attribute. The `sequence` increases with each new attribute values. For example, subsequent attribute values would be `sequence="1"`, `sequence="2"`, and `sequence="3"`, and so on.

Part 7: Creating relationships between products and items

1. After creating products and items for your catalog, define the relationships between products and items by adding information to the CATENTREL table. Use the following example from the ToolTech sample store as your guide. Complete this task for each product and item relationship value in your catalog:

   ```
   <catentrel
   catentry_id_parent="@product_id_147"
   catreltype_id="PRODUCT_ITEM"
   catentry_id_child="@catentry_id_152"
   sequence="2"
   quantity="1"
   />
   ```

   where

   - `catentry_id_parent` is the reference number of the source catalog entry in this relationship, that is, the product.
   - `catreltype_id` is the type of relationship: PRODUCT_ITEM
   - `catentry_id_child` is the reference number of the target catalog entry in this relationship, that is, the item.
   - `sequence` is the sequence number used to determine the display order.
   - `quantity` is a quantity that can be associated with the relationship.

   Note: Each time you add a relationship between a product and item, the `catentry_id_parent` and the `catentry_id_child` numbers change to create different relationships, based on the `catreltype_id`. With each new relationship, the `sequence` number is different. For example, if you have `sequence="2"`, the next relationship will have `sequence="3"`, followed by `sequence="4"`, and so on.

Part 8: Creating packages and bundles

1. Once you have created your products and items, create packages and bundles by adding information to the CATENTRY, CATENTDESC, and CATENTREL tables. Using the following example from the ToolTech sample store as your guide, begin by creating a package or bundle by adding information to the CATENTRY table. Complete this task for each package and bundle in your catalog:

   ```
   <catentry
   catentry_id="@package_id_102"
   member_id="&MEMBER_ID"
   catenttype_id="PackageBean"
   partnumber="sku-@package_id_102"
   mfpartnumber="sku-@package_id_102"
   mfname="ToolTech"
   markfordelete="0"
   buyable="1"
   />
   ```

   where

   - `catentry_id` is the reference number of the catalog entry.
   - `member_id` is the reference number that identifies the owner of the catalog entry.
   - `catenttype_id` identifies the type of catalog entry:
     – PackageBean = identifies a package.
     – BundleBean = identifies a bundle.

- `partnumber` is the reference number that identifies the part number of the catalog entry.
- `mfpartnumber` is the part number used by the manufacturer to identify the catalog entry.
- `mfname` is the name of the manufacturer of the catalog entry.
- `markfordelete` indicates if the catalog entry is marked for deletion:
  - 0 = No.
  - 1 = Yes.
- `buyable` indicates whether the catalog entry can be purchased individually:
  - 0 = No.
  - 1 = Yes.

  Note: Each time you create a package or a bundle, the `catentry_id`, `partnumber`, and `mfpartnumber` numbers change to create different package or bundle. For example, to create a new package, you could use `catentry_id="@package_id_103"`, `partnumber="sku-@package_id_103"`, and `mfpartnumber="sku-@package_id_103"`, including `catenttype_id="PackageBean"` to identify the entry as a package. To create a new bundle, you could use `catentry_id="@package_id_110"`, `partnumber="sku-@package_id_110"`, and `mfpartnumber="sku-@package_id_110"`, including `catenttype_id="BundleBean"` to identify the entry as a bundle, and so on.
- Using the following example from the ToolTech sample store as your guide, add the package or bundle description by adding information to the CATENTDESC table in the locale-specific XML file for translation purposes. Complete this task for each package and bundle description in your catalog:

```
<catentdesc
catentry_id="@catentry_id_102"
language_id="-1"
name="computer"
shortdescription="Computer"
longdescription="A combination of a central processing unit, monitor,
 hard drive, and color printer. An ideal starter system."
thumbnail="images/package_system_sm.gif"
fullimage="images/package_system.gif"
available="1"
published="1"
/>
```

  where
  - `catentry_id` is the internal reference number that indicates the catalog entry that this language specific information relates to.
  - `language_id` is the identifier of the language.
  - `name` is the language-dependent name of the catalog entry.
  - `shortdescription` is a brief description of the catalog entry.
  - `longdescription` is a detailed description of the catalog entry.
  - `thumbnail` is the thumbnail image path of the catalog entry.
  - `fullimage` is the full image path of the catalog entry.
  - `available` indicates the length of time to availability of the catalog entry.
  - `published` indicates whether the catalog entry should be displayed for the language indicated by `language_id`:
    - 0 = Do not display catalog entry.

  - 1 = Display catalog entry.
- Using the following example from the ToolTech sample store as your guide, create relationships between packages or bundles and their components by adding information to the CATENTREL table. Complete this task for each package or bundle component relationship in your catalog:

  ```
  <catentrel
  catentry_id_parent="@catentry_id_102"
  catreltype_id="PACKAGE_COMPONENT"
  catentry_id_child="@catentry_id_97"
  sequence="1.0"
  quantity="1.0"
  />
  ```

  where
  – `catentry_id_parent` is the reference number of the source catalog entry in this relationship, that is, the package or bundle.
  – `catreltype_id` is the type of this relationship:
    - PACKAGE_COMPONENT represents a relationship between a package and its components.
    - BUNDLE_COMPONENT represents a relationship between a bundle and its components.
  – `catentry_id_child` is the reference number of the target catalog entry in this relationship, that is, the component.
  – `sequence` is the sequence number used to determine the display order.
  – `quantity` is a quantity that can be associated with the relationship.

  Note: Each time you create a relationship between a package and bundle, the `catentry_id_parent` and `catentry_id_child` number changes to match existing catalog entries. With each new relationship, the `sequence` number is different. For example, if you begin with `sequence="1.0"`, the next relationship will have `sequence="2.0"`, followed by `sequence="3.0"`, and so on.

Part 9: Creating relationships between catalog groups and catalog entries

1. After creating catalog groups and catalog entries in your catalog, define the relationships between catalog groups and catalog entries by adding information to the CATGPENREL table. Under the master catalog structural restriction, a catalog entry cannot belong to more than one category. To place a catalog entry in more than one category, you must use a navigational catalog. Use the following example from the ToolTech sample store as your guide. Complete this task for each catalog group and catalog entry relationship in your catalog:

   ```
   <catgpenrel
   catgroup_id="@catgroup_id_11"
   catalog_id="@catalog_id_1"
   catentry_id="@product_id_102"
   sequence="0"
   />
   ```

   where
   - `catgroup_id` is the source catalog group of this relationship.
   - `catalog_id` is the catalog inside of which this relationship is found.
   - `catentry_id` is the target catalog entry of this relationship.

- `sequence` is the sequence number that determines the display order of the contents of the catalog group.

Note: Each time you create a relationship between catalog groups and catalog entries, the `catgroup_id` and `catentry_id` numbers change to form new relationships with different catalog groups and catalog entries. With each new relationship, the `sequence` number is different. For example, if you begin with `sequence="0"`, the next relationship will have `sequence="1"`, followed by `sequence="2"`, and so on.

Part 10: Creating merchandising associations

1. Using the following example from the ToolTech sample store as your guide, create merchandising associations between catalog entries by adding information to the MASSOCECE table. Complete this task for each merchandising association in your catalog:

```
<massoccece
massoccece_id="@relationship_id_100"
massoctype_id="X-SELL"
catentry_id_from="@product_id_1"
catentry_id_to="@product_id_15"
massoc_id="REQUIRES"
quantity="2.0"
rank="1.00000"
/>
```

   where

   - `massoccece_id` is the reference number of this entry.
   - `massoctype_id` is the identifier of the association type:
     - X-SELL = cross-sell.
     - UPSELL = up-sell.
     - ACCESSORY = accessory.
   - `catentry_id_from` is the catalog entry that is the source of the association.
   - `catentry_id_to` is the catalog entry that is the target of the association.
   - `massoc_id` is the identifier of the semantic specifier:
     - REQUIRED
     - OPTIONAL
     - COMES WITH
   - `quantity` is the quantity related to this association.
   - `rank` is the sequence number used for display order.

   Note: Each time you add a merchandising association, the `massoccece_id` number changes to represent a new relationship. The `catentry_id_from` and the `catentry_id_to` numbers vary to create new merchandise content for the association.

Part 11: Associating your catalog to a store

1. Associate your catalog to a store by assigning the catalog, its catalog groups, and catalog entries to a store in the database by using the existing `storecatalog.xml` from the ToolTech sample store as your guide. You should also assign display pages to the catalog groups and catalog entries. Add this information to the STORECAT, STORECENT, STORECGRP, DISPCGPREL, and DISPENTREL tables. If you are creating a multicultural catalog, create a separate store-catalog relationship XML file for each locale your store supports:

```
<storecat
catalog_id="@catalog_id_1"
storeent_id="@storeent_id_1"
mastercatalog="1"
/>
```

where
- `catalog_id` is the reference number of the catalog.
- `storeent_id` is the reference number of the store entity in the database.
- `mastercatalog` specifies a master catalog for the store. A value of 1 indicates that this catalog is designated as a master catalog.

2. Using the following example from the ToolTech sample store as your guide, add catalog entries to the store-catalog relationship. Complete this task for each catalog entry in your catalog:

   ```
   <storecent
   storeent_id="@storeent_id_1"
   catentry_id="@product_id_102"
   />
   ```

   where
   - `storeent_id` is the reference number of the store entity in the database.
   - `catentry_id` is the reference number of the catalog entry.

   Note: Each time you add a `catentry_id` to the store entity, the reference number changes to match an existing catalog entry.
3. Using the following example from the ToolTech sample store as your guide, add catalog groups to the store entity. Complete this task for each catalog group in your catalog:

   ```
   <storecgrp
   storeent_id="@storeent_id_1"
   catgroup_id="@catgroup_id_1"
   />
   ```

   where
   - `storeent_id` is the reference number of the store entity in the database.
   - `catgroup_id` is the reference number of the catalog group.

   Note: Each time you add a `catgroup_id` to the store entity, the reference number changes to match an existing catalog group.

   .

Part 12: Associating taxes to your catalog

1. Associate taxes to the products and services in your catalog for a specific store. You must associate a tax calculation code with the catalog entries by adding this information to the to the CATENCALCD table. For more information, see "Creating tax assets in WebSphere Commerce" on page 150.

Part 13: Associating shipping methods to your catalog

1. To associate shipping methods to the products and services in your catalog, you must associate a shipping calculation code with the catalog entries. Add this information to the CATENCALCD table. For more information, see "Creating shipping assets in WebSphere Commerce" on page 133.

Part 14: Associating a fulfillment center to your catalog

1. Associate your catalog with a fulfillment center to ship products to customers. A fulfillment center manages product inventory and shipping for a store. Add this information to the FFMCENTER table. For more information, see "Creating fulfillment assets in WebSphere Commerce" on page 108.

Part 15: Creating prices for your catalog entries

1. Create the pricing for your catalog entries. Pricing represents the price range for a catalog entry and any criteria that must be satisfied in order to use that price. To create a functional catalog, you need to add offering information to the database. Add this information to the TRADEPOSCN, TDPSCNCNTR, MGPTRDPSCN, OFFER, and OFFERPRICE tables. For more information, see "Creating pricing assets in WebSphere Commerce" on page 88. Or you can create or update the pricing for a catalog entry using the Product Management tools in the WebSphere Commerce Accelerator.

Part 16: Loading the XML file

1. After you have created your data, load the XML file into the database by either using the Loader package or through the Publish function in Store Services. For more information on the Loader package, see Part 7, "Publishing your store" on page 207.

   Note: You can also use the Product Management tools from the WebSphere Commerce Accelerator to create catalog assets for your master catalog. For more detailed information on the Product Management tools, see the WebSphere Commerce online help.

Displaying store catalog assets

After associating a catalog, catalog groups, and catalog entries to a store, assign JSP templates to display your catalog entries and catalog groups by creating these relationships in the database. Create these relationships in the format of XML files that can be loaded into the database using the Loader package.

The `storecatalog.xml` file from the ToolTech sample is located in its store archive file. To view the `storecatalog.xml` file, decompress the store archive using a ZIP program. The `storecatalog.xml` file is located in the following data directories:

- NT `*drive*:\WebSphere\CommerceServer\samplestores`
- 2000 `*drive*:\Program Files\WebSphere\CommerceServer\samplestores`
- AIX `/usr/WebSphere/CommerceServer/samplestores`
- Solaris Linux `/opt/WebSphere/CommerceServer/samplestores`
- 400 `/qibm/proddata/WebCommerce/samplestores`

Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

The `store-catalog.dtd` file is 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 Linux `/opt/WebSphere/CommerceServer/xml/sar`

- 400 /qibm/proddata/WebCommerce/xml/sar

Before you can create store-catalog relationships, ensure that you have created the store data assets. Complete the following tasks, each of which creates entries in the `storecatalog.xml` file:

1. In order to display your catalog groups (categories) in your store, you must assign JSP templates to your catalog groups. You can assign a particular display page template to a catalog group or a default template to display all catalog groups. Using the following example from the ToolTech sample store as your guide, assign catalog group templates by adding information to the DISPCGPREL table. Complete this task for each template you want to assign to your catalog groups:

   ```
   <dispcgprel
   catgroup_id="@catgroup_id_1"
   devicefmt_id="-1"
   dispcgprel_id="@dispcgprel_id_1"
   mbrgrp_id="0"
   pagename="/ToolTech/CategoryDisplay.jsp"
   storeent_id="@storeent_id_1"
   rank="0"/>
   ```

   where

   - `catgroup_id` is the reference number of the catalog group for which this page name will be displayed. A value of 0 indicates that this page name will be used for all catalog groups.
   - `devicefmt_id` is the reference number of the device type that the page will be displayed on. A value of –1 indicates that this template page will be used by an HTTP browser.
   - `dispcgprel_id` is the reference number of this entry.
   - `mbrgrp_id` is the reference number of the member group for which this template page will be displayed. A value of 0 indicates that this template page will be used for all member groups.
   - `pagename` is the name and path of the display template page.
   - `rank` is a sequence number used to break ties when more than one page satisfies the selection criteria.

   Note: Each time you assign a JSP template to a catalog group, the `catentry_id` changes sequence to match an existing catalog entry.

2. To display your catalog entries (products, items, packages, bundles, and dynamic kits) in your store, you must assign JSP templates to your catalog entries. You can assign a default template to display all catalog entries, or a default to display each type of catalog entry, for example, a template for products and another template for items, or a specific template for a specific catalog entry. Using the following example from the ToolTech sample store as your guide, assign templates by adding information to the DISPENTREL table. Complete this task for each template you want to assign to your catalog entries:

   ```
   <dispentrel
   auctionstate="0"
   catentry_id="0"
   catenttype_id="ProductBean"
   devicefmt_id="-1"
   dispentrel_id="@dispentrel_id_1"
   mbrgrp="0"
   pagename="/ToolTech/ProductDisplay.jsp"
   storeent_id="@storeent_id_1"
   rank="0"/>
   ```

where

- `auctionstate` indicates that this template page displays a catalog entry that is on auction:
  - 0 = Not an auction template.
  - 1 = Auction template.
- `catentry_id` is the reference number of the catalog entry for which this page name will be displayed. A value of 0 indicates that this page name will be used for all catalog entries.
- `catenttype_id` is the type of catalog entry that this page will be used to display:
  - ProductBean = Displays a product.
  - ItemBean = Displays an item.
  - PackageBean = Displays a package.
  - BundleBean = Displays a bundle.
  - DynamicKitBean = Displays a dynamic kit.
- `devicefmt_id` is the reference number of the device type that the page will be displayed on. A value of −1 indicates that this template page will be used by an HTTP browser.
- `dispentrel_id` is the reference number of the catalog entry.
- `mbrgrp` is the reference number of the member group for which this template page will be displayed. A value of 0 indicates that this template page will be used for all member groups.
- `pagename` is the name and path of the display template page.
- `storeent_id` is the reference number of the store for which this page will be displayed.
- `rank` is a sequence number used to break ties when more than one page satisfies the selection criteria.

Note: Each time you assign a JSP template to a catalog entry, the `catentry_id` changes sequence to match an existing catalog entry.

Creating a navigational catalog

A WebSphere Commerce store allows two types of catalogs: master and navigational. Navigational catalogs do not need to meet the structural restrictions that are placed on master catalogs. These catalog are meant to provide a flexible display structure to allow you to create navigational catalogs that suit your store's requirements.

In particular, navigational catalogs do not need to satisfy the following restrictions that are imposed on master catalogs:

- A master catalog must be a proper tree, which means that there are no cycles and cannot use the following structure: The parent category A has a subcategory B. It is important that B and any of B's subcategories are not the parent category of A.
- A product cannot belong to more than one category.

The following tasks create a navigational catalog by modifying the NewFashion sample store catalog. The resulting catalog can no longer be classified as a master catalog since it the following steps introduce category cycles and the categorization of some products into multiple categories. A classic navigational catalog is created by adding information to the category relationship tables: CATGRPREL, which holds the subcategory relationships, and CATGPENREL, which holds the category-product relationships. Although these examples involve NewFashion, you can follow these basic steps with your own master catalog, making the appropriate adjustments to match your catalog information, structure, and designs.

Creating category cycles

The NewFashion catalog contains four top categories: **Men's Fashions**, **Women's Fashions**, **New Arrivals**, and **Homepage promotions**. This example shows you how to copy the **New Arrivals** category into **Men's Fashions** and copy **Homepage promotions** into **Women's Fashions**, and then change name to **New Arrivals**.

To change the NewFashion sample store master catalog to a navigational catalog using category cycles, do the following:

1. Publish the NewFashion store archive to create the NewFashion sample store. NewFashion is available in US English and one of the nine national languages shipped with WebSphere Commerce. Choose one of the `NewFashion_en_US_`*`locale`*`.sar` files for publication.
2. Open the `catalog.xml` file in an editor. The file is located in the following WebSphere Commerce directory:
   - NT *`drive`*`:\WebSphere\CommerceServer\samplestores\NewFashion \`*`locale`*`\data`
   - 2000 *`drive`*`:\Program Files\WebSphere\CommerceServer\samplestores\NewFashion \`*`locale`*`\data`
   - AIX `/usr/WebSphere/CommerceServer/samplestores/NewFashion /`*`locale`*`/data`
   - Solaris Linux `/opt/WebSphere/Commerce/samplestores/NewFashion /`*`locale`*`/data`
   - 400 `/qibm/proddata/WebCommerce/samplestores/NewFashion /`*`locale`*`/data`
3. Locate the CATGRPREL data section in the `catalog.xml` file. Create a new top category relationship between **Men's Fashion** and **New Arrivals**. Currently, both categories are at the top level. For a navigational relationship, create a new section which places **New Arrivals** as a subcategory of **Men's Fashion** while preserving the original structure. Under the CATGRPREL section, add the following extract:

```
<catgrprel
catgroup_id_parent="@catgroup_id_11"
catgroup_id_child="@catgroup_id_21"
catalog_id="@catalog_id_1"
sequence="7"
/>
```

   where
   - `catgroup_id_parent` is the catalog group internal reference number of the parent category as defined by the NewFashion sample store. In this example, `@catgroup_id_11` is the **Men's Fashion** category.
   - `catgroup_id_child` is the catalog group internal reference number of the child category as defined by the NewFashion sample store. In this example, `@catgroup_id_21` is the **New Arrivals** category.

- `catalog_id` is the internal reference number of the catalog as defined by the NewFashion sample store.
- `sequence` is the number that determines the display order of the contents of the catalog group as defined by the NewFashion sample store. In this example, the **New Arrivals** category will be displayed last, after the first six **Men's Fashions** subcategories.

4. Repeat the above step, this time creating a new top category relationship between **Women's Fashions** and **Homepage promotions**. Currently, both categories are at the top level. For a navigational relationship, create a new section which places **Homepage promotions** as a subcategory of **Women's Fashion** while preserving the original structure. Under the CATGRPREL section, add the following extract:

```
<catgrprel
catgroup_id_parent="@catgroup_id_20"
catgroup_id_child="@catgroup_id_22"
catalog_id="@catalog_id_1"
sequence="9"
/>
```

   where
   - `catgroup_id_parent` is the catalog group internal reference number of the parent category as defined by the NewFashion sample store. In this example, `@catgroup_id_20` is the **Women's Fashion** category.
   - `catgroup_id_child` is the catalog group internal reference number of the child category as defined by the NewFashion sample store. In this example, `@catgroup_id_22` is the **Homepage promotions** category.
   - `catalog_id` is the internal reference number of the catalog as defined by the NewFashion sample store.
   - `sequence` is the number that determines the display order of the contents of the catalog group as defined by the NewFashion sample store. In this example, the **Homepage promotions** category will be displayed last.
5. Now that **Homepage promotions** is a subcategory to **Women's Fashions**, the category name is erroneous. Rename the category name to **New Arrivals**, to match the new subcategory in **Men's Fashions**.
6. Save the `catalog.xml` file.
7. To view your changes, do one of the following: publish the modified NewFashion store archive through Store Services or load the `catalog.xml` file with the Loader package as instructed in "Loading database asset groups" on page 267.

Adding a product to a second category

This example shows you how to copy products from one category to another while preserving the original structure. The **Homepage promotions** category contains the **Summer Nightgown** product, which could also belong under the **Sleepwear** subcategory for the **Women's Fashions** top category. These instructions will show you how to copy the **Summer Nightgown** product and its SKUs to the **Sleepwear** category.

To change the NewFashion sample store master catalog to a navigational catalog adding a product to a second category, do the following:

1. Publish the NewFashion store archive to create the NewFashion sample store. NewFashion is available in US English and one of the nine national languages shipped with WebSphere Commerce. Choose one of the `NewFashion_en_US_`*`locale`*`.sar` files for publication.

2. Open the `catalog.xml` file in an editor. The file is located in the following WebSphere Commerce directory:
   - NT *drive*:\WebSphere\CommerceServer\samplestores\NewFashion *locale*\data
   - 2000 *drive*:\Program Files\WebSphere\CommerceServer\samplestores\NewFashion *locale*\data
   - AIX /usr/WebSphere/CommerceServer/samplestores/NewFashion /*locale*/data
   - Solaris Linux /opt/WebSphere/CommerceServer/samplestores/NewFashion /*locale*/data
   - 400 /qibm/proddata/WebCommerce/samplestores/NewFashion /*locale*/data
3. Locate the CATGPENREL data section in the `catalog.xml` file. Create a new product entry for **Summer Nightgown**, originally a product under the **Homepage promotions** category. Under the CATGPENREL section, add the following extract to include the product:

   ```
   <catgpenrel
   catgroup_id="@catgroup_id_18"
   catalog_id="@catalog_id_1"
   catentry_id="@product_id_2692"
   sequence="2"
   />
   ```

   where
   - `catgroup_id` is the catalog group internal reference number as defined by the NewFashion sample store. In this example, `@catgroup_id_18` is the **Sleepwear** category.
   - `catalog_id` is the internal reference number of the catalog as defined by the NewFashion sample store.
   - `catentry_id` is the catalog entry internal reference number as defined by the NewFashion sample store. In this example, `@catentry_id_2692` is the **Summer Nightgown** product.
   - `sequence` is the number that determines the display order of the contents of the catalog group as defined by the NewFashion sample store. In this example, the **Summer Nightgown** product will be displayed last.
4. After adding the **Summer Nightgown** product entry, add the SKU entries for the product under the CATGPENREL section, as defined in the NewFashion sample store. Currently, the **Summer Nightgown** product contains ten defined SKUs. Under the CATGPENREL section, add the following extracts to include the SKUs:

   ```
   <catgpenrel
   catgroup_id="@catgroup_id_18"
   catalog_id="@catalog_id_1"
   catentry_id="@catentry_id_2695"
   sequence="2"
   />

   <catgpenrel
   catgroup_id="@catgroup_id_18"
   catalog_id="@catalog_id_1"
   catentry_id="@catentry_id_2696"
   sequence="2"
   />
   ```

```
<catgpenrel
catgroup_id="@catgroup_id_18"
catalog_id="@catalog_id_1"
catentry_id="@catentry_id_2697"
sequence="2"
/>

<catgpenrel
catgroup_id="@catgroup_id_18"
catalog_id="@catalog_id_1"
catentry_id="@catentry_id_2698"
sequence="2"
/>

<catgpenrel
catgroup_id="@catgroup_id_18"
catalog_id="@catalog_id_1"
catentry_id="@catentry_id_2699"
sequence="2"
/>

<catgpenrel
catgroup_id="@catgroup_id_18"
catalog_id="@catalog_id_1"
catentry_id="@catentry_id_2700"
sequence="2"
/>

<catgpenrel
catgroup_id="@catgroup_id_18"
catalog_id="@catalog_id_1"
catentry_id="@catentry_id_2701"
sequence="2"
/>

<catgpenrel
catgroup_id="@catgroup_id_18"
catalog_id="@catalog_id_1"
catentry_id="@catentry_id_2702"
sequence="2"
/>

<catgpenrel
catgroup_id="@catgroup_id_18"
catalog_id="@catalog_id_1"
catentry_id="@catentry_id_2703"
sequence="2"
/>

<catgpenrel
catgroup_id="@catgroup_id_18"
catalog_id="@catalog_id_1"
catentry_id="@catentry_id_2704"
sequence="2"
/>
```

where

- `catgroup_id` is the catalog group internal reference number as defined by the NewFashion sample store. In this example, `@catgroup_id_18` is the **Sleepwear** category.
- `catalog_id` is the internal reference number of the catalog as defined by the NewFashion sample store.

- `catentry_id` is the catalog entry internal reference number as defined by the NewFashion sample store. In this example, `@catentry_id_2695` through `@catentry_id_2704` represent the ten SKUs that have been defined for the **Summer Nightgown** product.
- `sequence` is the number that determines the display order of the contents of the catalog group as defined by the NewFashion sample store. In this example, the **Summer Nightgown** SKUs will be displayed last.

5. Save the `catalog.xml` file.
6. To view your changes, do one of the following: publish the modified NewFashion store archive through Store Services or load the `catalog.xml` file with the Loader package as instructed in "Loading database asset groups" on page 267.

Managing catalog assets in WebSphere Commerce

Over time, you will need to update the database asset information from the master catalog. Maintaining your catalog is an ongoing process, as you will need to continually add and remove merchandise, create and associate categories or catalog groups, and update product information, such as descriptions and price.

You can change your catalog assets by editing the WebSphere Commerce XML data using the existing database entries and `catalog.xml` files from your store. Use the WebSphere Commerce sample store XML files as a reference, located in the following data directory:

- NT `*drive*:\WebSphere\CommerceServer\samplestores`
- 2000 `*drive*:\Program Files\WebSphere\CommerceServer\samplestores`
- AIX `/usr/WebSphere/CommerceServer/samplestores`
- Solaris Linux `/opt/WebSphere/CommerceServer/samplestores`
- 400 `/qibm/proddata/WebCommerce/samplestores`

Note: These examples originate from the NewFashion sample store and identify which XML elements must be modified to change the catalog asset information.

Catalog groups

Catalog groups are created in a WebSphere Commerce catalog using the CATGROUP and CATGRPDESC database tables. From the `catalog.xml` file, a typical catalog group looks like the following extract:

```
<catgroup
catgroup_id="@catgroup_id_1"
member_id="&MEMBER_ID"
identifier="Accessories"
markfordelete="0"
/>
```

The `catgroup_id` is the internal reference number of the catalog group. Each catalog group is assigned an internal reference number in WebSphere Commerce, which identifies the group when adding catalog entries. The `identifer` is an external name for the catalog group. Both elements are unique within the database assets and cannot be duplicated.

Names and descriptions belong to the locale specific `catalog.xml` file, one of which is required for each locale your store supports. A typical catalog group containing translatable information looks like the following extract:

```
<catgrpdesc
language_id="&en_US"
catgroup_id="@catgroup_id_1"
name="Accessories"
shortdescription="Accessories"
longdescription="Accessories"
published="1"
/>
```

The `language_id` identifies the language of your catalog information. This identifier must change to match each language your store supports. The `name` is displayed to the customer, as are the `shortdescription` and `longdescription` elements, which may contain a brief and detailed description of the catalog group.

When creating a new catalog group, follow the above structure for the information.

Notes:

1. While the `identifer` and `name` elements are identical in the above example, the content can vary. For instance, you might choose to rename your catalog group to **Complementary Additions**. In such a case, you do not need to change the information in `identifer`, only `name`.
2. When deleting catalog groups, ensure that `catgroup_id` occurrences are updated accordingly. For instance, if you also want to delete the catalog entries under the catalog group, then you would remove the entire XML entries. However, if you plan to keep the catalog entries, then you need to change the `catgroup_id` to the correct group.

Catalog entries

Catalog entries are created in a WebSphere Commerce catalog using the information from the CATENTRY and CATENTDESC database tables. A catalog entry can be a product, item, package, bundle, or dynamic kit. From the `catalog.xml` file, a typical catalog entry looks like the following extract:

```
<catentry
catentry_id="@product_id_102"
baseitem_id="@baseitem_id_102"
member_id="&MEMBER_ID"
catenttype_id="ProductBean"
partnumber="product-sku-nf-102"
mfpartnumber="product-sku-nf-102"
mfname="NewFashion"
markfordelete="0"
buyable="1"
/>
```

The `catentry_id` is the internal reference number of the product catalog entry. The `baseitem_id` is base item that the catalog entry relates to, for inventory purposes. The `partnumber` is the reference number that identifies the part number of the catalog entry. The `mfpartnumber` is the part number used by the manufacturer to identify the catalog entry. These elements are unique within the database assets and cannot be duplicated.

The `catenttype_id` identifies the type of catalog entry: ItemBean, ProductBean, PackageBean, BundleBean, or DynamicKitBean.

Names and descriptions belong to the locale specific `catalog.xml` file, one of which is required for each locale your store supports. Merchandise images are also included in this file. A typical catalog group containing translatable information looks like the following extract:

```
<catentdesc
catentry_id="@product_id_102"
language_id="&en_US"
name="Belt"
shortdescription="Classic belt"
longdescription="This classic belt looks great with your favorite jeans,
or takes you to work in style. 1 1/2 inches wide in full-grain leather
with a solid nickel buckle."
thumbnail="images/mens_accessories_belt_sm.gif"
fullimage="images/mens_accessories_belt.gif"
available="1"
published="1"
/>
```

The `language_id` identifies the language of your catalog information. This identifier must change to match each language your store supports. The `name` is displayed to the customer, as are the `shortdescription` and `longdescription` elements, which may contain a brief and detailed description of the catalog entry.

When creating a new catalog entry, follow the above structure for the information.

Notes:

1. When deleting catalog entries, ensure that each occurrence of the unique elements are updated accordingly. For instance, if you also want to delete the catalog entries under the catalog group, then you would remove the entire XML entries. However, if you plan to keep the catalog entries, then you need to change the `catgroup_id` to the correct group.
2. Products must be created before other types of catalog entries.

If you do not want to manually change the XML files, choose one of the following methods: the Product Management tools or the Catalog Manager utilities.

Product Management tools

The Product Management tools in the WebSphere Commerce Accelerator allow you to manage the products in your store's master catalog using various wizards and notebooks. You can update your catalog's content or create new catalog data:

- Create, update, and delete products and product details using the wizard or notebook. Products act as templates for SKUs, the individual items which are ultimately sold to a customer. Product details include the product code (which uniquely identifies the product), the product name and description, any merchandising options (such as displaying a product to customers or indicating if that product is part of a special promotion), the product images, tax and shipping specifications, discounts assigned to the products, and manufacturer information.
- Generate, update, and delete SKUs (or items) for purchase. SKUs represent each orderable item of merchandise for sale. All SKUs related to a particular product exhibit the same set of attributes and are distinguished by their attribute values. Additions or changes made to SKUs include the same information as products, except on an orderable basis.
- Create, update, and delete categories (or catalog groups), which are a group of objects that have similar properties which are used to organize products or services offered by the store. You can manage the category hierarchy of your

master catalog by creating, changing and deleting categories and details about the categories, such as the category code, the name, and description, including parent category and images.
- Associate products and SKUs with categories by choosing the parent category or moving products and SKUs from one category to another.
- Create attributes and attribute values for products. Each possible combination of attributes and attribute values equals a new SKU. You must predefine attribute values before assigning them to SKUs. After creating attributes and their values, you can create or update information such as name, description, (text, whole numbers, or decimal numbers), and sequence in which the attributes and attribute values will appear.
- Create, update, delete, and associate catalog pricing with products. You can define a price for a product or SKU, in one or more currencies, along with a set of conditions such as setting a price for single or bulk quantities, which must be satisfied in order to use the price.

You can refer to the Product Management section in the online help for detailed instructions on each task.

Notes:
1. The Product Management tools are recommended for minor changes only. For large catalog updates, such as adding or removing seasonal merchandise or preparing for a clearance sale, use the Loader package.
2. Any changes to the catalog data cannot be displayed in the store unless you disable caching or remove the currently cached JSP pages. For more information, refer to the CacheDelete command in the WebSphere Commerce online help. The CacheDelete command initiates remote cleanup of the dynamic page cache and allows you to manage the cache without requiring direct access to the file system. Before using this command ensure that Auto Page Invalidation is enabled. Note that you must be logged in as either a Site Administrator or Store Developer to use this command.

Catalog Manager

You can also maintain your catalog using the Loader package or the Web Editor, two utilities available from the Catalog Manager. The Loader package is ideal for importing large amounts of existing product information into the database. In WebSphere Commerce, this is the primary tool to create and manage catalog information. This package consists primarily of command utilities for preparing and loading data into a WebSphere Commerce database. The Loader package also allows you to extract data from a database as an XML document, transform XML data into alternate XML formats, and transform data between a character-delimited variable format and an XML data format.

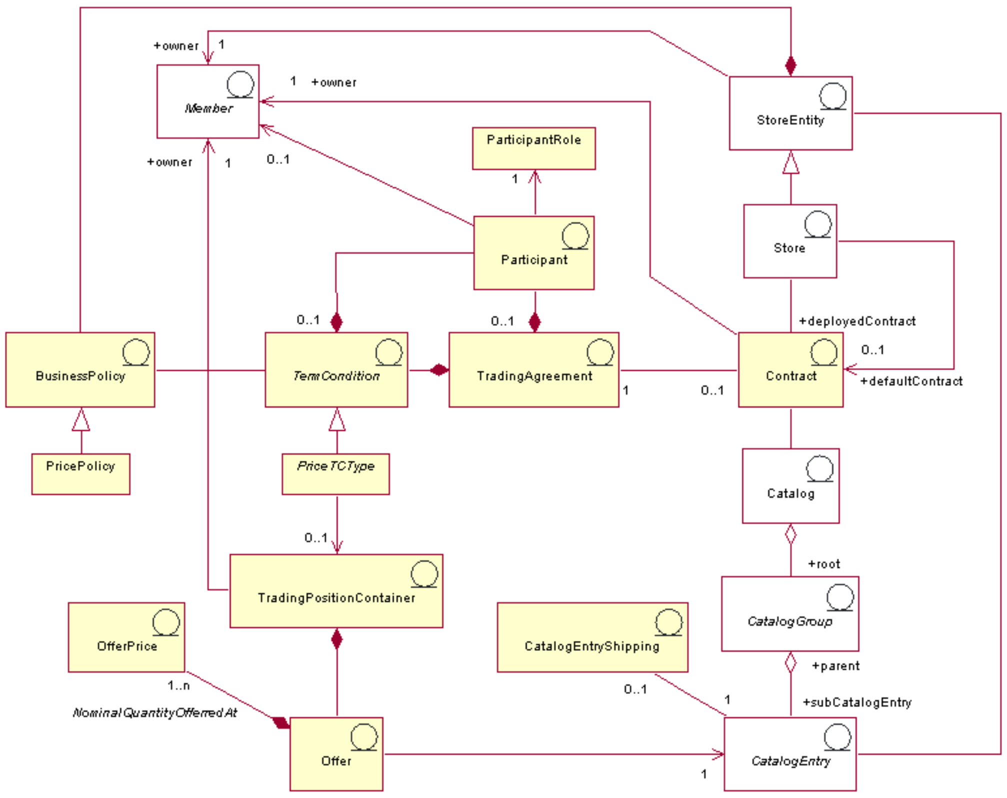
The Web Editor uses a Web browser interface, in which you can create, edit, or delete catalog data. Data-entry forms for viewing and updating information are central to the Web Editor. In the simplest case, the forms correspond to tables in the WebSphere Commerce database. The administrator can choose to use the default forms provided or to customize the available forms. Refer to the *IBM WebSphere Commerce Version 5.4 Catalog Manager User's Guide* for more information.

Chapter 9. Pricing assets

Pricing represents the price for a catalog entry and any criteria that must be satisfied in order to use that price. In order to create a functional catalog, you need to add pricing information to the database. You can create pricing information in the format of XML files that can be loaded into the database using the Loader package. Or you can use the Product Management tools from the WebSphere Commerce Accelerator for small amounts of pricing data.

Understanding pricing in WebSphere Commerce

The following diagram illustrates the pricing assets in the WebSphere Commerce Server.



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

Offer

Offers, or pricing, are different prices for the same product or item to different customers or organizations. An offer represents the price of a catalog entry and criteria, such as the quantity to be purchased, that the customer must satisfy in order to pay that price. For example, merchandise or services are often priced differently for children, students, adults, and seniors. In WebSphere Commerce, an offer is also known as a trading position and is part of a trading position container.

Offer price

The *offer price* is a price at which catalog entries are offered by a store by means of trading agreements or contracts. An offer can have one or more than one offer prices defined in multiple currencies.

Trading position container

An offer is part of a *trading position container*, which is owned by a member. A trading position container contains trading positions. It can be made available to all customers, or to only customers in certain groups through the trading agreements or contracts, and the terms and conditions in the contracts. Under a contract, a trading position container is a price business object that can be referenced by multiple price business policies and can be shared by a store or all stores in a store group. A trading position container is also referred to as a *price list*.

Terms and conditions

Terms and conditions define the behavior and properties of a trading agreement. Many terms and conditions reference business policies because several aspects of a store's operation are defined by business policies.

Types of pricing terms and conditions

Pricing terms and conditions define what products are available under a contract and what prices the customer will pay for the products. At least one of the following pricing terms is required in a contract:

- A *customized price list* specifies that both the list of products for sale and their prices are customized for sale in a contract and their price is customized. Items are not limited to a section of the store catalog they can be from anywhere in the store catalog.
- An *entire catalog with adjustment term* offers all of the products available in a store catalog for sale with a percentage adjustment (mark-up or discount) from the base price as defined in the store catalog. If no adjustment is specified, items are sold at the base price.
- A *price list with adjustment term* offers all of the products available in a price list for sale with a percentage adjustment (mark-up or discount) from the base price as defined in the store catalog. If no adjustment is specified, items are sold at the base price.
- A *price list with selective adjustment term* is similar to price list with adjustment except the adjustment is not applied to the entire price list. The adjustment is made on a subset of the price list. The subset of the price list may either be a product set business policy or a customized product set.

Trading agreement

A *trading agreement* can be a contract, an RFQ, a business account, or an auction. A trading agreement is an agreement negotiated between a seller and a buyer upon which the buyer is enabled to purchase certain items with the specified terms and conditions and the business policies stipulated in the contract. For example, it allows the customer to purchase products from a store at the specified price for a specified period of time, under the pricing terms and conditions In WebSphere Commerce, all customers must shop in a store under a contract, A store may deploy one or more contracts and one of them can be designated to be the default contract. A default contract contains a set of terms and conditions that are associated with a set of store default policies. A trading agreement may contain zero or more participants of different roles.

Participant

A *participant* can be part of either a trading agreement or terms and conditions. A participant is a member which can be a member group, an organization, and so on. If a participant of a buyer role is specified for a contract, a buyer must be a member of the buyer participant in order to shop under the contract. The terms and conditions in the contract can also contain zero or multiple participants.

Participant role

A participant can have one of the following *participant roles*:

- Creator
- Seller
- Buyer
- Supplier
- Approver
- Account holder
- Buyer contact
- Seller contact
- Attorney
- Administrator.

Contract

A *contract* contains the offer price for the product. In WebSphere Commerce, all customers must shop under a contract. A contract allows the customer to purchase products from a store at the specified price for a specified period of time, under the terms and conditions, and business policies, stipulated in the contract. A store owns zero or more contracts, and owns at least one default contract.

Business policy

Business policies are sets of rules followed by a store or store group that define business processes, industry practices, and the scope and characteristics of a store or store groups offerings. Business policies are enforced with a combination of a combination of one or more business policy commands that implement the rules of the business policy, a reference to a business object that the rules act on, and a set of properties to configure the operation of the business policy commands.

Price policy

A *price policy* contains a reference to a price list and can be associated with multiple business policy commands that define how the business policies will be implemented on the price lists. The policy may be defined for a store or a store group. If the policy is registered for a store group, then the policy may be used by all stores in that group.

Catalog entry shipping

Catalog entry shipping information includes information about how the product is packaged for shipping. Each catalog entry can have different types of shipping information defined. For example, the height, weight, and length of the product when packaged.

Other pricing assets

The following assets are associated with pricings:

- A *member* who owns the trading position container. A trading position container only has one owner.
- A *store entity* represents a store in the WebSphere Commerce Server database.
- A *catalog* contains catalog entries that will be referenced in a contract. The catalog contains all hierarchical and navigational information for the online catalog and is a collection of catalog groups and catalog entries that are available for display and purchase at an online store.
- A *catalog group*, or category, are generic groupings of catalog entries, created for navigational and catalog partitioning purposes. A catalog group belongs to a catalog and may contain more than one catalog group or catalog entries. You can associate catalog groups to more than one catalog.
- A *catalog entry* represents orderable merchandise in an online catalog. Catalog entries belong to catalog groups. An offer is always associated with one catalog entry.

For more detailed information on the structure of pricing assets in the WebSphere Commerce Server, see the pricing object and data models in the WebSphere Commerce online help.

Creating pricing assets in WebSphere Commerce

You have two options for creating your pricing assets:

- Create prices using the Product Management tools in the WebSphere Commerce Accelerator. Using the tools in the WebSphere Commerce Accelerator is most suited to creating prices for a very small catalog.
- Create prices in an XML file, which can be loaded by the WebSphere Commerce Loader package, or as a part of a store archive, which can be published through Store Services. This method is more suitable for creating large amounts of data.

For more information on creating prices using the Product Management tools in the WebSphere Commerce Accelerator, see the WebSphere Commerce online help. For more information on creating prices in an XML file, see “Creating pricing assets in an XML file”.

Creating pricing assets in an XML file

Create your pricing assets in the format of XML files that can be loaded into the database using the Loader package. For more information on the Loader package, see Part 7, “Publishing your store” on page 207.

1. Review the XML files used to create pricing assets for the sample stores. All files for the sample stores are located in the corresponding store archive file.

   The store archive files are located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\samplestores`

- 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
- AIX `/usr/WebSphere/CommerceServer/samplestores`
- Solaris `/opt/WebSphere/CommerceServer/samplestores`
- Linux `/opt/WebSphere/CommerceServer/samplestores`
- 400 `/qibm/proddata/WebCommerce/samplestores`

Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

Each sample store includes two `offering.xml` files, which include the pricing information. To view the `offering.xml` files in the store archive, decompress it using a ZIP program. The `offering.xml` files are located in the data directory. The language-specific `offering.xml` is in a locale-specific subdirectory of the data directory.

2. Review the information in Appendix B, "Creating your data" on page 305.
3. Create an `offering.xml` file, either by copying one of the `offering.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `offering.xml`. The DTD files 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`
   - Linux `/opt/WebSphere/CommerceServer/xml/sar`
   - 400 `/qibm/proddata/WebCommerce/xml/sar`
4. Create a trading position container. In order to offer prices for the goods in your store, you must first create a trading position container. To create a trading position container, add information to the TRADEPOSCN table.
   a. Using the following example as your guide, create a trading position container in your XML file in the TRADEPOSCN table:

```
<tradeposcn
tradeposcn_id="@tradeposcn_id_101"
member_id="&MEMBER_ID"
markfordelete="0"
name="ToolTech"
precedence="0"

/>
```

   where
   - `tradeposcn_id` is a generated unique key
   - `member_id` is the owner of the trading position container.
   - `markfordelete` is as follows:
     – 0 = the TradingPositionContainer can be used
     – 1 = the TradingPositionContainer has been marked for deletion (refer to the DBClean utility) and should not be used
   - `name`is a mnemonic name for the trading position container, unique for a particular owner.

- `precedence` is when more than one trading position containers is qualified at a particular time, the one with the highest PRECEDENCE is used.

5. Associate the master catalog with a trading position container by adding information to the CATGRPTPC table. When you associate the master catalog with a trading position container, every catalog entry in the master catalog must have a standard price. For more information on creating master catalogs, see "Displaying store catalog assets" on page 74.
   a. Using the following example as your guide, associate the master catalog to the trading position container by adding information to the CATGRPTPC table:
      ```
      <catgrptpc
      catalog_id="@catalog_id_1"
      tradeposcn_id="@tradeposcn_id_101"
      />
      ```
      where
      - `catalog_id` is the master catalog.
      - `tradeposcn_id` is the trading position container.
6. Create offers and offer price for catalog entries by adding information to the OFFER and OFFERPRICE tables
   a. Using the following example as your guide, create an offer for a catalog entry by adding information to the OFFER table. Note that you must have created catalog entries before you can create prices. For more information on creating catalog entries, see "Displaying store catalog assets" on page 74.
      ```
      offer
      offer_id="@offer_id_138"
      startdate="2000-06-19 00:00:00.000000"
      catentry_id="@product_id_102"
      precedence="0"
      published="1"
      identifier="1"
      flags="1"
      tradeposcn_id="@tradeposcn_id_101"
      />
      ```
      where
      - `offer_id` is a generated unique key.
      - `startdate` is the start of the time range during which this offer is effective.
      - `catentry_id` is the catalog entry offered for sale.
      - `precedence` is when more than one offer is effective at a particular time, the one with the highest PRECEDENCE is used.
      - `published` is
        – 0 = not published (temporarily disabled)
        – 1 = published
        – 2 = marked for deletion (and not published).
      - `identifier` is a number that uniquely identifies this offer along with its specified catalog entry and trading position container.
      - `flags` are
        – 1 = shiptoAddressRequired - if 1, OrderPrepare will return an error if an OrderItem references this Offer but does not have a shipping address.
      - `tradeposcn_id` is the trading position container this offer is part of.

b. Using the following example as your guide, create an offerprice for a catalog entry by adding information to the OFFERPRICE table. The offer price is the actual price at which a catalog entry is offered for sale. Note that you must have created catalog entries before you can create prices. For more information on creating catalog entries, see "Displaying store catalog assets" on page 74.

```
<offerprice
offer_id="@offer_id_138"
currency="USD"
price="590.00"
/>
```

where

- `offer_id` is offer associated with this price.
- `currency` is the currency which the price is offered in.
- `price` is the price for the nominal quantity (see CATENTSHIP.NOMINALQUANTITY) of the product referred to by the offer.

Note: To display multiple currencies in your store, create a separate XML entry in the OFFERPRICE table for each currency. For example, to display the currency in Canadian dollars, use `currency="CAD"` in a new XML entry. The `price` value would change to reflect the price in Canadian dollars. Or you can use a conversion, allowing the customer to display different rates based on the currency they select. For more information, see "Creating currency assets using an XML file" on page 121.

c. Repeat steps a and b for all catalog entries in your catalog.

For more information about the use of @ and **&** see Appendix B, "Creating your data" on page 305.

Chapter 10. Contract assets

In WebSphere Commerce, all store customers must shop under a contract. A contract allows customers to purchase products from a store at a specified price for a specified period of time under specific conditions. When browsing a store's catalog, customers will only see products covered by the contracts they are entitled to within the store.

If you want customers who do not have any contract with your store (for example, guest shoppers) to be able to shop in the store, or if you want customers to be able to purchase products not covered by their contracts, your store will require a *default contract*.

Important

WebSphere Commerce Professional Edition supports only the store default contract.

Contracts other than the store default contract are supported only by WebSphere Commerce Business Edition.

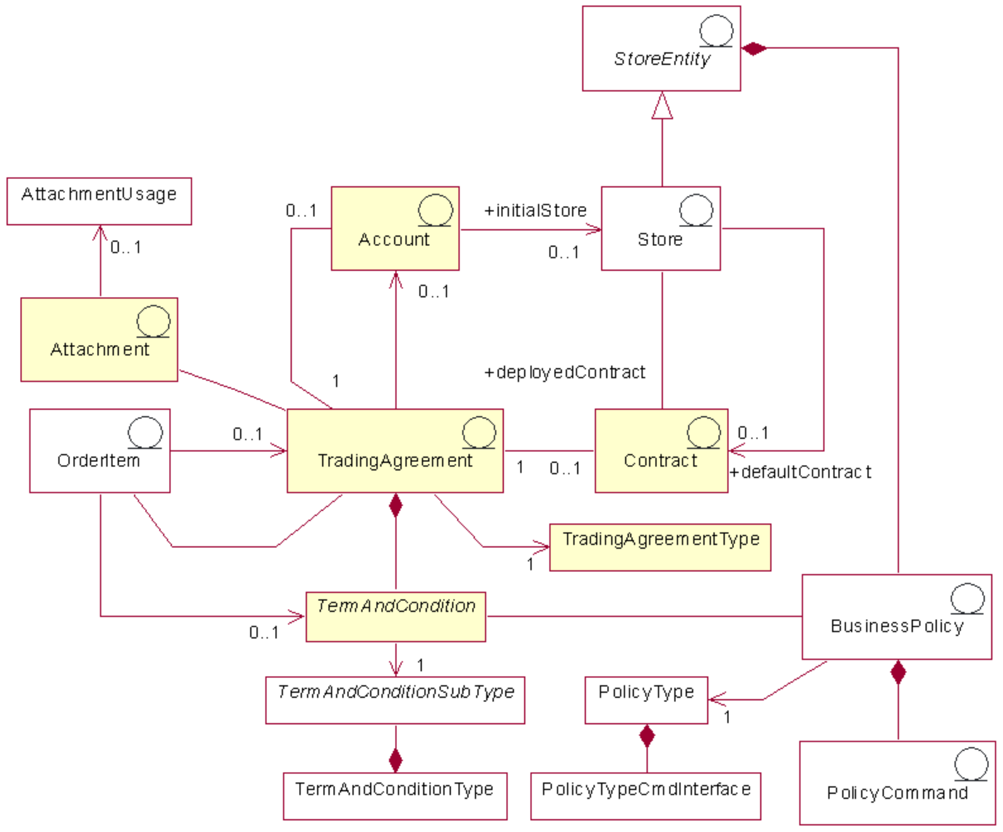
In order to allow all customers to shop at a store, a store created with WebSphere Commerce must include the following:

- Business policies
- Default contract

The business policies are referenced by the default contract, thus allowing all customers to shop at a store.

Understanding contracts in WebSphere Commerce

The following diagram illustrates the structure of contracts in WebSphere Commerce:



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

Account (business account)

A business account represents the relationship between a buyer organization and a seller organization. A business account can be used to organize various trading agreements and to specify terms and conditions related to the relationship between buyer and seller such as: invoice customization, purchase order verification, or maintaining a buyer's line of credit with the seller.

Contracts are associated with business accounts since they represent an agreement between a buyer and a seller. The exception to this is the store default contract which cannot be associated with a business account. A business account can have many contracts associated with the account.

A business account is a type of trading agreement. For a description of trading agreements, see "Trading agreement" on page 95.

Important: Business accounts are only supported by WebSphere Commerce Business Edition.

Contract

There are two types of active contracts associated with stores: deployed contracts and default contracts. Deployed contracts entitle specific buyer organizations or individual buyers and can be created using the WebSphere Commerce Accelerator after you have created your store. A deployed contracts is associated with one business account. A default contract defines the default behavior of your store for buyers who do not have any other contracts with your store. A default contract can only be created using XML files and only one default contract may be defined for a store. For more information on contracts, refer to the online information. For information on creating a default contract asset, see "Creating a default contract asset in WebSphere Commerce" on page 97.

A contract is a type of trading agreement. For a description of trading agreements, see "Trading agreement".

Trading agreement

WebSphere Commerce provides a number of trading mechanisms governing the interactions between buyers and sellers. The following trading mechanisms are supported by different editions of WebSphere Commerce:

- Auctions (supported by both Business and Professional editions)
- Business accounts (only supported by Business edition)
- Contracts (see restrictions discussed previously in this chapter)
- Request for quotes (RFQs) (only supported by Business edition)

All of these trading mechanisms have common properties. For example, all trading mechanisms have participants and they all have rules governing the behavior of the trading mechanism. The rules governing the behavior of trading mechanisms are known as *terms and conditions* in WebSphere Commerce.

A trading agreement represents an instance of a trading mechanism and records the properties of that instance of a trading mechanism. Each contract, business account, and RFQ in WebSphere Commerce is represented by a trading agreement. There is a single trading agreement that governs all auctions in WebSphere Commerce.

A trading agreement consists of a profile stored in the TRADING table; participants stored in the PARTICIPNT table; terms and conditions stored in the TERMCOND table; and optional attachments stored as Universal Resource Identifiers (URIs) in the ATTACHMENT table. Because a trading agreement can have multiple attachments, attachments are related to the trading agreement through the TRDATTACH table. Note that attachments are not supported for RFQs.

In addition to the general trading agreement, each type of trading agreement stores additional information specific to the type of trading agreement in its own table: CONTRACT stores contract-specific information; RFQ stores RFQ-specific information; and ACCOUNT stores business account-specific information.

Terms and conditions

Terms and conditions define the behavior and properties of a trading agreement.

For contracts, the terms and conditions define how a contract is implemented for a buyer organization. They define what is being sold under the contract; the price of the items being sold; how the items are shipped; how orders are paid for; how item returns are handled; how orders are approved; and where orders are shipped from.

Some terms and conditions reference business policies because many aspects of a store's operation are defined by business policies. Terms and conditions provide parameters for the business polices they reference. Providing parameters to the business policies allows you to modify the behavior of business policies for each contract.

Business policies

Business policies are sets of rules followed by a store or group of stores. Business policies define business processes, industry practices, and the scope and characteristics of a store's or group of stores' offerings. They are the central source and reference template for all allowed and supported practices within a store or group of stores.

In WebSphere Commerce, business policies are enforced with a combination of one or more business policy commands that implement the rules of the business policy, a reference to a business object that the rules act on, and a set of properties to configure the operation of the business policy commands. Terms and conditions may provide parameters for the business polices they reference. This allows the behavior of the business policy to be modified depending on the term and condition referencing the business policy.

Attachment

An attachment provides addition information about a trading agreement that is not covered by other elements of the trading agreement. An example is a file that provides additional information about RFQ requirements and any general remarks about the RFQ. A trading agreement can have multiple attachments. Attachments are stored outside of WebSphere Commerce and the trading agreement stores Universal Resource Identifiers (URIs) to the attachments. Examples of URIs include the following:

- http://www.mycompany.com/information/document1.txt
- file:///home/joeuser/mydocs/document1
- ftp://ftp.mycompany.com/information/attachment.txt

All attachments can be assigned an attachment usage that indicates what the attachment is for. The attachment usage is an optional property of an attachment.

Order item

An order item is a product that is included with an order. Different order items in a single order may be purchased under different contract trading agreements. Buyers can select the contract trading agreement they shop under at either the start of the shopping flow or when they add an item to their order, depending on the store design. When purchasing items under different contract trading agreements the following rules apply:

- Contract trading agreements for all items in an order must share at least one payment method. If the contract for an item does not share a payment method, the buyer can not add that item to the order. Only the payment methods shared by all items in an order can be used to pay for the order.
- All items in an order must come from contract trading agreements belonging to the same business account or the store default contract.

For more detailed information on the structure of contract assets in WebSphere Commerce, see the contract data model in the WebSphere Commerce online help.

Creating a default contract asset in WebSphere Commerce

The default contract defines the default behavior of a store. As with all contracts, you can set the available products, prices, payment methods, shipping methods, and other store behavior.

The store default contracts provided with the WebSphere Commerce sample stores contain terms and conditions that specify the following:

- Customers can purchase all products available in the master catalog for the store at standard prices set in the master catalog (no discounts or mark–ups).
- Any shipping charges are paid to the seller (store).
- Customers can return purchases without penalty charges within a certain number of days.
- Customers can receive refunds using the same payment method used for the original purchase.

Also, the most general version of a store's default contract omits terms and conditions that restrict the payment and shipping methods that buyers can use. Omitting these terms allows buyers to pay for purchases using any of the default payment methods supported by the store and use any shipping method available in the store.

The default contract's properties are defined in its terms and conditions. Some of the terms and conditions reference business policies. For more information on business policies and terms and conditions, refer to the online information.

To create a default contract asset, do the following:

1. Review the online information on terms and conditions, contracts, default contracts, and business policies.
2. Review the business policies defined in the `wcs.bootstrap.xml` file. For information on the `wcs.bootstrap.xml` file, refer to the online information.
3. Review the files used to create default contract assets for the sample stores. All sample stores files are located in the corresponding store archive file. Each sample store includes a `businesspolicy.xml` and `contract.xml`, which includes additional business policy information and default contract information. The store archive files are located in the following directory:
   - AIX `/usr/WebSphere/CommerceServer/samplestores`
   - 400 `/qibm/proddata/WebCommerce/samplestores`
   - Linux `/opt/WebSphere/CommerceServer/samplestores`
   - Solaris `/opt/WebSphere/CommerceServer/samplestores`
   - 2000 *drive*`:\Program Files\WebSphere\CommerceServer\samplestores`
   - NT *drive*`:\WebSphere\CommerceServer\samplestores`

   Notes:

   a. The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

   b. To view the businesspolicy.xml and contract.xml files in the store archive, decompress them using a ZIP program. The files are located in the data directory.
   c. The contract asset files for the ToolTech sample store that is provided with WebSphere Commerce Business Edition includes information for contracts other than the store default contract.
4. Review the information in Appendix B, "Creating your data" on page 305.
5. Create a businesspolicy.xml file by copying one of the businesspolicy.xml files in the sample store archives, or by creating a new file. Instructions on creating a new file are in "Creating business policy XML files" on page 99. If you want to create different business policies from the ones discussed, see the DTD file that corresponds to businesspolicy.xml. The DTD files are located in the following directory:
   - AIX /usr/WebSphere/CommerceServer/xml/sar
   - 400 /qibm/proddata/WebCommerce/xml/sar
   - Linux /opt/WebSphere/CommercServer/xml/sar
   - Solaris /opt/WebSphere/CommercServer/xml/sar
   - 2000 *drive*:\Program Files\WebSphere\CommerceServer\xml\sar
   - NT *drive*:\WebSphere\CommerceServer\xml\sar
6. Load the businesspolicy.xml file using the Loader package. For more information on the Loader package, see Part 7, "Publishing your store" on page 207. If you are creating a multicultural store, you may want to create separate XML files for each locale your store supports. The locale-specific file should specify all description information, so it can be easily translated.
7. Create a contract.xml file by copying one of the contract.xml files in the sample store archives, or by creating a new file. Instructions for creating a new file are in "Creating a default contract XML File" on page 100. If you want to create a more complex default contract, review the B2BTrading.dtd file which defines the structure of a contract.xml file. The B2BTrading.dtd is located in the following directory:
   - AIX /usr/WebSphere/CommerceServer/xml/trading
   - 400 /qibm/proddata/WebCommerce/xml/trading
   - Linux /opt/WebSphere/CommerceServer/xml/trading
   - Solaris /opt/WebSphere/CommerceServer/xml/trading
   - 2000 *drive*:\Program Files\WebSphere\CommerceServer\xml\trading
   - NT *drive*:\WebSphere\CommerceServer\xml\trading
8. Publish the contract using the ContractImportApprovedVersion command. For more information, see Chapter 29, "Publishing business accounts and contracts" on page 273. Information on the ContractImportApprovedVersion command is also available in the online information.

WebSphere Commerce Business Edition users can define contracts for specific customers using the WebSphere Commerce Accelerator. For more information on creating contracts for specific customers, refer to the online information.

Creating business policy XML files

While WebSphere Commerce provides a number of business policies that the terms and conditions in your store's default contract can reference, some business policies must still be defined by you. You must define any return charge, return approval, and pricing business policies that the store default contract terms reference. Commands for these business policies are provided and can be used without modification. If you want to create your own business policies, refer to *IBM WebSphere Commerce Programmer's Guide*.

In order to create business policies for your store, you must create the business policy and associate one or more commands with the business policy. To create a business policy, add information to the POLICY table. To associate a command with a business policy, add information to the POLICYCMD table.

To create a business policy and associate commands with the policy, do the following:

1. Create a business policy in your business policies XML file by adding information to the POLICY table. Use the following example as a guide:

   ```
   <policy
   policy_id="@policy_id_10"
   policyname="MasterCatalogPriceList"
   policytype_id="Price"
   storeent_id="@storeent_id_1"
   properties="name=InFashion&amp;member_id=&MEMBER_ID"
   />
   ```

   where

   - `policy_id` is the unique, numeric identifier for the business policy.
   - `policyname` is a unique name for this business policy.
   - `policytype_id` is the type of policy being defined. Valid `policytype_ids` are:
     - InvoiceFormat
     - Payment
     - Price
     - ProductSet
     - ReturnApproval
     - ReturnCharge
     - ReturnPayment
     - ShippingCharge
     - ShippingPayment
   - `storeent_id` is the store or store group.
   - `properties` is a list of name–value pairs that is sent to the business policy command.
2. Associate a command with the business policy in your business policies XML file by adding information to the POLICYCMD table. Use the following example as a guide:

   ```
   <policycmd
   policy_id="@policy_id_10"
   businesscmdclass=
     "com.ibm.com.commerce.price.commands.CalculateContractPricesCmdImpl"
   />
   ```

   where

- `policy_id` is the numeric identifier of the business policy with which the command is being associated.
- `businesscmdclass` is the name of Java class implementing the business policy.

The line breaks in the `businesscmdclass` attribute are for display purposes only.

For more information about the use of @ and & see Appendix B, "Creating your data" on page 305.

Creating a default contract XML File

In order to create a default contract, you must define the contract, the contract owner, the contract description, the contract participants, and the terms and conditions of the contracts. Contract information is stored in four tables: CONTRACT, PARTICIPNT, TRADING, and TERMCOND.

The default contract is associated with a store using the STOREDEF database table. For WebSphere Commerce Business Edition users, contracts other than the default contract are associated with a store using the STORECNTR database table.

To create a default contract, do the following:

1. Define the default contract in your XML file. The default contract is defined at the beginning of the XML file as follows:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE Trading SYSTEM "B2BTrading.dtd">
   <Trading>
   <Contract state="Active" origin="Manual"
     name="&STORE_IDENTIFIER; Default Contract" majorVersionNumber="1"
     minorVersionNumber="0" contractUsage="Default">
   ```

   Note that line breaks in the `Contract` element are for display purposes only.
2. Define the contract owner. Use the following example as a guide:

   ```
   <ContractOwner>
     <Member>
       <Organization distinguishName="&MEMBER_IDENTIFIER;" />
     </Member>
   </ContractOwner>
   ```

   where `distinguishName` is the name of the user owning the contract in LDAP distinguished name format. For example, `uid=erickoeck,ou=People,dc=ibm,dc=com`.
3. Define the contract description in your contract XML file. Use the following example as a guide:

   ```
   <ContractDescription title="This is a store default contract." languageId="-1">
   </ContractDescription>
   ```

   where
   - `title` is a text description of the contract.
   - `languageId` is the language the title is in. The following values are predefined for `languageId`:
     - -1 (English – US)
     - -2 (French)
     - -3 (German)
     - -4 (Italian)

   – -5 (Spanish)
   – -6 (Brazilian Portuguese)
   – -7 (Simplified Chinese)
   – -8 (Traditional Chinese)
   – -9 (Korean)
   – -10 (Japanese)

   Additional values can be defined for `languageId` by updating the language assets for your store. For more information on language assets, see Chapter 14, "Language assets" on page 117.

4. Define the contract participants in your contract XML file. Use the following example as a guide:

```
<Participant role="Buyer">
</Participant>
<Participant role="Seller">
  <Member>
    <Organization distinguishName="&MEMBER_IDENTIFIER;"/>
  </Member>
</Participant>
```

   where `distinguishName` is the name of the user that is the seller for this contract in LDAP distinguished name format. For example, `uid=erickoeck,ou=People,dc=ibm,dc=com`. In many cases, this will be the same as the contract owner.

   Note: No members are specified in the buyer participant role because the contract is available to all users with a buyer role.

5. Define the terms and conditions in your contract XML file. The XML elements and attributes are different for the various types of terms and conditions. Use the `B2BTrading.dtd` file to learn the XML elements and attributes to use for each type of term. When defining terms and conditions the following attributes are commonly used:

   policyName
      The name of the business policy that the term and condition references. This name is stored in POLICY.POLICYNAME.

   policyType
      The type of business policy that the term and condition references. Valid values are:
      - Price
      - ProductSet
      - InvoiceFormat
      - Payment
      - ReturnApproval
      - ReturnCharge
      - ReturnPayment
      - ShippingCharge
      - ShippingMode

   storeIdentity
      The store or store group for the term and condition.

distinguishName

The name of the user that owns the store or store group. The name must be in LDAP distinguished name format. For example, `uid=wcsadmin,o=Root Organization`.

The following sample terms and conditions are preceded by a description of what they define:

- All buyers can purchase all items in the store's master catalog at the prices set in the master catalog:

```
<TermCondition>
  <PriceTC>
    <PriceTCMasterCatalogWithOptionalAdjustment>
    </PriceTCMasterCatalogWithOptionalAdjustment>
  </PriceTC>
</TermCondition>
```

- Buyers pay any shipping charges to the seller:

```
<TermCondition>
  <ShippingTC>
    <ShippingTCShippingCharge>
      <PolicyReference policyName="StandardShippingChargeBySeller"
        policyType="ShippingCharge" storeIdentity="&STORE_IDENTIFIER;">
        <Member>
          <User distinguishName="&MEMBER_IDENTIFIER;">
        </Member>
      </PolicyReference>
    </ShippingTCShippingCharge>
  </ShippingTC>
</TermCondition>
```

  Line breaks in the `PolicyReference` element are for display purposes only.

- Buyers can return products without any return charges. The products must be returned within the number of days defined in the `ApprovalByDays` business policy:

```
<TermCondition>
  <ReturnTC>
    <ReturnTCReturnCharge>
      <ReturnChargePolicyReference>
        <PolicyReference policyName="NoCharges"
         policyType="ReturnCharge"
         storeIdentity="&STORE_IDENTIFIER;">
          <Member>
            <Organization distinguishName="&MEMBER_IDENTIFIER;">
          </Member>
        </PolicyReference>
      </ReturnChargePolicyReference>
      <ReturnApprovalPolicyReference>
        <PolicyReference policyName="ApprovalByDays"
         policyType="ReturnApproval"
         storeIdentity="&STORE_IDENTIFIER;">
          <Member>
            <Organization distinguishName="&MEMBER_IDENTIFIER;">
          </Member>
        </PolicyReference>
      </ReturnApprovalPolicyReference>
    </ReturnTCReturnCharge>
  </ReturnTC>
</TermCondition>
```

  Line breaks in the `PolicyReference` elements are for display purposes only.

Note for WebSphere Commerce Business Edition users:
Omitting these terms and conditions from the store default contract indicates that, by default, the store does not accept returns. Other contracts, however, may allow buyers to do returns, by defining the returns term and condition.

Note for WebSphere Commerce Professional Edition users:
Omitting these terms and conditions from the store default contract indicates that the store does not accept returns.

- Refunds are paid using the same payment method the buyer used when completing the order:

```
<TermCondition>
  <ReturnTC>
    <ReturnTCRefundPaymentMethod>
      <PolicyReference policyName="UseOriginalPayment"
       policyType="ReturnPayment" storeIdentity="&STORE_IDENTIFIER;">
        <Member>
          <User distinguishName="&MEMBER_IDENTIFIER;">
        </Member>
      </PolicyReference>
    </ReturnTCRefundPaymentMethod>
  </ReturnTC>
</TermCondition>
```

  Note that line breaks in the `PolicyReference` element are for display purposes only.

For more information about the use of @ and & see Appendix B, "Creating your data" on page 305.

Chapter 11. Fulfillment assets

Fulfillment centers are used by stores as both inventory warehouses and shipping and receiving centers. One store may have one or many fulfillment centers associated with it. The fulfillment center manages the product inventory and shipping for a store. Fulfillment includes picking, packing, and shipping. Picking is the selection of products in one or more releases from a fulfillment center, packing is putting these products into shipping containers, and shipping is sending them to customers.

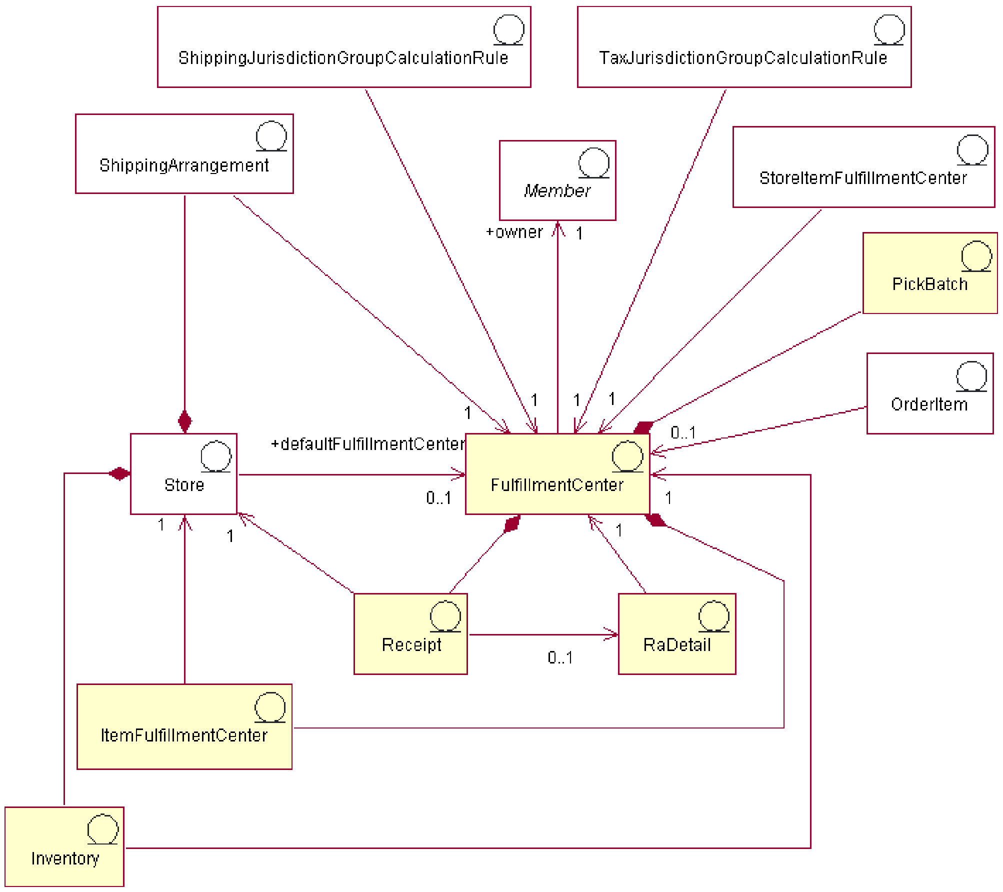
Products are configured for fulfillment with the Product wizard and the Product notebook. Product configuration provides options to track inventory, allow backorders, force backorders, release the product separately, and specify that the product should not be returned.

Typically, there are a number of people working in a fulfillment center at one time, each with a different task or tasks to perform. The WebSphere Commerce Accelerator divides the most common tasks into roles, and these roles are assigned to users. In the WebSphere Commerce Accelerator, you must select one fulfillment center at logon time, if you have been assigned one or more roles pertaining to fulfillment.

Note: For more information on fulfillment, fulfillment centers, and roles, refer to the WebSphere Commerce online help.

Understanding fulfillment assets in WebSphere Commerce

In order to understand fulfillment assets, it is necessary to understand the relationships between fulfillment and the store. This can be explained by the use of an information model. The following sections describe the relationships that inventory has to a store and other assets.



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

Fulfillment center

In the preceding diagram, the *fulfillment center* is at the center of the fulfillment process. A fulfillment center has an owner, defined in the MEMBER table. Each store is associated with a fulfillment center and has a default fulfillment center. A fulfillment center can have several stores associated with it. There are several interactions between the store and the fulfillment center, as indicated in the diagram. For more information on store assets, see "Understanding store assets in WebSphere Commerce" on page 41.

Receipts

Fulfillment centers receive inventory for items on a daily, weekly, or monthly basis. When inventory is received for an item, a *receipt* is created in the RECEIPT table which records information about the quantity received, as well as the store which owns the inventory. As orders are processed, the RECEIPT table is updated to reflect the current available inventory levels. For information on creating receipts, see "Creating inventory assets in WebSphere Commerce" on page 170.

RaDetail

RaDetail is the detailed information about items on an expected inventory record. This information can be used to estimate when inventory may be expected to be received at a fulfillment center and provide customers with expected shipping dates for backordered items.

Inventory

A store has *inventory* which is associated with the fulfillment center. Inventory includes everything that can be physically accounted for in a fulfillment center. Inventory is associated with one store and one fulfillment center. Information about the inventory that a store owns at the fulfillment center is also recorded such as reserved quantities, amounts on backorder, and amounts allocated to backorders. This information is stored in the ITEMFFMCTR table. For more information on inventory and inventory assets, see Chapter 21, "Inventory assets" on page 167.

Shipping arrangements

The last relationship between the store and the fulfillment center involves *shipping arrangements*. Shipping arrangements indicate that a fulfillment center can ship products on behalf of a store using a shipping mode. Each store has a shipping arrangement with a fulfillment center and vice versa. Shipping arrangements are set up in the SHPARRANGE table. For information on creating shipping arrangements, see "Creating shipping fulfillment assets" on page 142.

Other fulfillment assets

There are other relationships to a fulfillment center that are not directly related to a store. A pick batch is one that is associated with one fulfillment center. A pick batch groups together order releases for their processing as a unit at a fulfillment center, and creates pick slips and pack slips. Once items have been picked and packed, an order release can then be shipped, and the shipment can be confirmed. Pick batch information is stored in the PICKBATCH table. An order item is also associated with one fulfillment center. An item is a specific instance of a product, defined by attributes. Information about each item in an order is stored in the ORDERITEM table. For more information on order assets, see Chapter 22, "Order assets" on page 171.

Like other entities, a fulfillment center has rules which govern some of it's actions. Each fulfillment centre has rules for tax and shipping charges. These are defined in the TAXJCRULE and SHPJCRULE tables respectively. For more information on tax and shipping assets, see Chapter 18, "Shipping assets" on page 131, and "Understanding tax assets in WebSphere Commerce" on page 147.

For more detailed information on the structure of fulfillment assets in WebSphere Commerce Server, see the fulfillment data models in the WebSphere Commerce online help.

Creating fulfillment assets in WebSphere Commerce

Before your store can ship goods to a customer, you must define the fulfillment center, or centers, that will supply these goods. Create this information in the format of XML files that can be loaded into the database using the Loader package. For more information on the Loader package, see Part 7, "Publishing your store" on page 207.

To create fulfillment assets for your store using an XML file, do the following:

1. Review the XML files used to create fulfillment assets for the sample stores. All files for the sample stores are located in the corresponding store archive file.

   The store archive files are located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
   - AIX `/usr/WebSphere/CommerceServer/samplestores`
   - Solaris `/opt/WebSphere/CommerceServer/samplestores`
   - Linux `/opt/WebSphere/CommerceServer/samplestores`
   - 400 `/qibm/proddata/WebCommerce/samplestores`

   Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

   Each sample store includes a `fulfillment.xml` file, which includes the fulfillment information. To view the `fulfillment.xml` file in the store archive, decompress it using a ZIP program. The `fulfillment.xml` file is located in the data directory.
2. Review the information in Appendix B, "Creating your data" on page 305.
3. Create a `fulfillment.xml` file, either by copying one of the `fulfillment.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `fulfillment.xml`. The DTD files 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`
   - Linux `/opt/WebSphere/CommerceServer/xml/sar`
   - 400 `/qibm/proddata/WebCommerce/xml/sar`
4. Define the fulfillment center, or centers that your store supports:

   a. Using the following example as your guide, define a fulfillment center in the XML file in the FFMCENTER table:

   ```
   <ffmcenter
      ffmcenter_id="@ffmcenter_id_1"
      member_id="&MEMBER_ID"
      name="ToolTech Home"
      defaultshipoffset="0"
      markfordelete="0"
      />
   ```

where

- `ffmcenter_id` is a generated unique key
- `member_id` is the owner of the fulfillment center
- `name` is a string that, along with the owner, uniquely identifies this fulfillment center.
- `defaultshipoffset` is an estimate of the number of seconds it takes for an item to be shipped from this fulfillment center. This value can be overridden in the STORITMFFC table.
- `markfordelete` indicates whether the fulfillment center should be deleted as follows: 0 = do not delete. 1 = delete if no longer in use. For more details, see the information on the Database Cleanup utility in the WebSphere Commerce online help.

b. Using the following example as your guide, describe the fulfillment center in the XML file in the FFMCENTDS table. If you are creating a multicultural store, you should include this information in a locale-specific XML file.

```
<ffmcentds
   ffmcenter_id="@ffmcenter_id_1"
   description="The fulfillment center that supplies products to ToolTech."
   language_id="&en_US"
   displayname="ToolTech Fulfillment"
   staddress_id="@staddress_id_en_US_1"
   />
```

where

- `ffmcenter_id` is a generated unique key
- `description` is a description of the fulfillment center, suitable for display to a customer.
- `language_id` is the language in which this information will display.
- `displayname` is the name of the fulfillment center, suitable for display to a customer.
- `staddress_id` is the physical location of the fulfillment center.

c. Repeat steps a and b for all fulfillment centers that your store supports.

For more information about the use of @ and **&** see Appendix B, "Creating your data" on page 305.

Creating store fulfillment assets

After you have defined the fulfillment center or centers that will supply goods for your store, you must associate a fulfillment center to each product. That is, you must identify which fulfillment center will supply which of your products. To create this relationship, add information to the INVENTORY table. Create this information in the format of XML files that can be loaded into the database using the Loader package. For more information on the Loader package, see Part 7, "Publishing your store" on page 207.

Note:

1. You must create store assets before you can associate a store with a fulfillment center. For more information on creating store assets, see "Creating store data assets in an XML file" on page 42. You must also create the catalog assets before you can create the store fulfillment assets. For more information, see "Displaying store catalog assets" on page 74.

2. Create store fulfillment assets only if you implement non-ATP fulfillment. The INVENTORY table is not used by a store that includes the ATP functions.

To create the store fulfillment relationship using an XML file, do the following:

1. Review the XML files used to create store fulfillment assets for the sample stores. All files for the sample stores are located in the corresponding store archive file.

   The store archive files are located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
   - AIX `/usr/WebSphere/CommerceServer/samplestores`
   - Solaris `/opt/WebSphere/CommerceServer/samplestores`
   - Linux `/opt/WebSphere/CommerceServer/samplestores`
   - 400 `/qibm/proddata/WebCommerce/samplestores`

   Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

   Each sample store includes a `storefulfill.xml` file, which includes the store fulfillment information. To view the `storefulfill.xml` file in the store archive, decompress it using a ZIP program. The `storefulfill.xml` file is located in the data directory.
2. Review the information in Appendix B, "Creating your data" on page 305.
3. Create a `storefulfill.xml` file, either by copying one of the `storefulfill.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `storefulfill.xml`. The DTD files are located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\xml\sar`
   - 2000 `drive:\Program Files\WebSphere\CommercServer\xml\sar`
   - AIX `/usr/WebSphere/CommerceServer/xml/sar`
   - Solaris `/opt/WebSphere/CommerceServer/xml/sar`
   - Linux `/opt/WebSphere/CommerceServer/xml/sar`
   - 400 `/qibm/proddata/WebCommerce/xml/sar`
4. Using the following example as your guide, create a store-fullfillment center relationship in the XML file, by adding information to the INVENTORY table.

   ```
   <inventory
   catentry_id="@catentry_id_1470"
   quantity="100"
   ffmcenter_id="@ffmcenter_id_1"
   store_id="@storeent_id_1"
   quantitymeasure="C62"
   inventoryflags="0"
   />
   ```

   where

   - `catentry_id` is the catalog entry that this fulfillment center will supply.

- `quantity` is the quantity amount, in units indicated by QUANTITYMEASURE, available from this fulfillment center.
- `ffmcenter_id` is the fulfillment center that will be supplying the inventory.
- `store_id` is the store for which the inventory is being supplied.
- `quantitymeasure` is the unit of measurement for QUANTITY.
- `inventoryflags` are bit flags that indicate how QUANTITY is used:
  - 1 = noUpdate. The default UpdateInventory task command does not update QUANTITY.
  - 2 = noCheck. The default CheckInventory and UpdateInventory task commands do not check QUANTITY.

5. Repeat step 3 for each catalog entry in your store.

For more information about the use of @ and & see Appendix B, "Creating your data" on page 305.

Chapter 12. Campaign assets

Campaigns serve to organize your marketing efforts. Campaigns are typically created by either a Marketing Manager, or by a Merchandising Manager. They are often associated with a certain set of objectives. For instance, a ″Back to School″ campaign may have an objective of increasing sales of children's clothes during the campaign.

Understanding campaigns in WebSphere Commerce

Within WebSphere Commerce, campaigns contain any number of campaign initiatives, which define a condition. The campaign initiatives generate targeted content for the customers, when the defined condition is evaluated to be true. The result is that a campaign is the high-level marketing element that organizes the initiatives.

Campaign initiatives are associated with a campaign that contains a collection of initiatives. As an example of this relationship, if an office supply store had a ″Back to School″ campaign, the initiatives would be responsible for lower-level actions, such as advertising a discount on pens, or suggesting lined paper to any customer who has registered and listed her occupation as a student.

Campaign initiatives are capable of displaying three types of dynamic content:
- Suggestive selling initiative
- Collaborative filtering-based recommendation
- Awareness advertisement

Suggestive selling content is designed to provide rule-based category and product recommendations, targeted at a specific customer audience, based on a customer's profile, and other customers' behaviors. Initiatives displaying this type of content are intended to be used to create cross-sell and up-sell opportunities.

Collaborative filtering-based recommendations are also intended to create product recommendations, but they use a different recommendation algorithm, which targets items based on customers' overall behavior, rather than predefined rules.

Awareness advertisements are designed to provide advertising content targeted at a specific customer audience, based on the same criteria as those used for suggestive selling, but they are intended to be used to increase a customer's awareness about activities at the online store, highlight special offers, and to increase brand awareness.

Initiatives can be incorporated into any page on the site. When the site is designed, special placeholders, called e-Marketing Spots, are placed on the site. When displayed to a customer, these placeholders are replaced by the specific targeted content. Target locations are assigned by scheduling initiatives to display in e-Marketing Spots in the desired locations. For more information on adding e-Marketing Spots to your store, see Chapter 32, "Adding e-Marketing Spots to your store" on page 295.

Campaign initiatives contain a condition that determines when and to whom they are displayed. This condition is defined when the initiative is created and can be changed during the lifetime of the initiative to adjust the initiative's visibility and the displayed content.

Campaign initiatives generate statistics about their use. These statistics can be viewed using the WebSphere Commerce Accelerator by Merchants, Marketing Managers, and Merchandising Managers. The statistics illustrate an initiative's clickthrough rate for each e-Marketing Spot where it is implemented. These statistics provide feedback on the effectiveness of the initiative, as well as comparative success rates among the various locations in which it displays.

Creating campaign assets in WebSphere Commerce

Campaigns and campaign initiatives are typically created by either a Marketing Manager, or by a Merchandising Manager using the Campaign and Campaign Initiative wizards in the WebSphere Commerce Accelerator. For more information, see the WebSphere Commerce online help.

For more information on adding e-Marketing Spots to your store, see Chapter 32, "Adding e-Marketing Spots to your store" on page 295.

Chapter 13. Payment assets

WebSphere Commerce supports the IBM Payment Manager. In order to create payment assets for your store, specify whether the store uses the Payment Manager, and if so, what type of payment cassette and brands the store accepts.

Specify this information by doing the following:

- Create payment data in the form of an XML file (`paymentinfo.xml`) that is loaded during store publish using Store Services. This configures Payment Manager with the merchant and the brand types specified for the store being published. For more information, see "Create payment assets using an XML file".

  Note: `paymentinfo.xml` does not populate tables in WebSphere Commerce Server database. It configures the Payment Manager. `paymentinfo.xml` is only applicable if you are using offline credit card as the payment method. To configure other methods of payment, see the next bullet.
- Complete the set up of Payment Manager 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. For more information, see the WebSphere Commerce online help topic *"Setting up Payment Manager for your store."*

Create payment assets using an XML file

To create payment assets for your store using an XML file, do the following:

1. Review the XML files used to create payment assets for the sample stores. All files for the sample stores are located in the corresponding store archive file.

   The store archive files are located in the following directory:

   - ▶ NT `drive:\WebSphere\CommerceServer\samplestores`
   - ▶ 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
   - ▶ AIX `/usr/WebSphere/CommerceServer/samplestores`
   - ▶ Solaris `/opt/WebSphere/CommerceServer/samplestores`
   - ▶ Linux `/opt/WebSphere/CommerceServer/samplestores`
   - ▶ 400 `/qibm/proddata/WebCommerce/samplestores`

   Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

   Each sample store includes a `paymentinfo.xml` file, which include the payment information. To view the `paymentinfo.xml` file in the store archive, decompress it using a ZIP program. The `paymentinfo.xml` files are located in the data directory.
2. Create a `paymentinfo.xml` file, either by copying one of the `paymentinfo.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `paymentinfo.xml`. The DTD files 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`
- Linux `/opt/WebSphere/CommerceServer/xml/sar`
- 400 `/qibm/proddata/WebCommerce/xml/sar`

3. Enable or disable Payment Manager.
   a. Using the following example as your guide, in your XML file enable or disable Payment Manager and specify what types of payment cassette, currencies and brands your store accepts

```
<paymentinfo>
  <PaymentManager enable="yes"/>
  <Cassette type="OfflineCard">
   <Account currency="USD">
    <Brand type="MasterCard"/>
    <Brand type="VISA"/>
    <Brand type="American Express"/>
   <Account/>
   <Account currency="EUR">
    <Brand type="MasterCard"/>
    <Brand type="VISA"/>
    <Brand type="American Express"/>
   </Account>
  </Cassette>
</paymentinfo>
```

   where:
   - `enable` is whether Payment Manager is enabled or disabled.
   - `Cassette type` is the type of cassette supported.
   - `Account currency` is the currency your store supports. Account currency is required if you are using the OfflineCard cassette type. The currency must be identified in a three letter code conforming to the ISO 4217 standard. For example, ″USD″ for U.S. dollars.
   - `Brand type` is the type of credit card supported by the account and the currency.

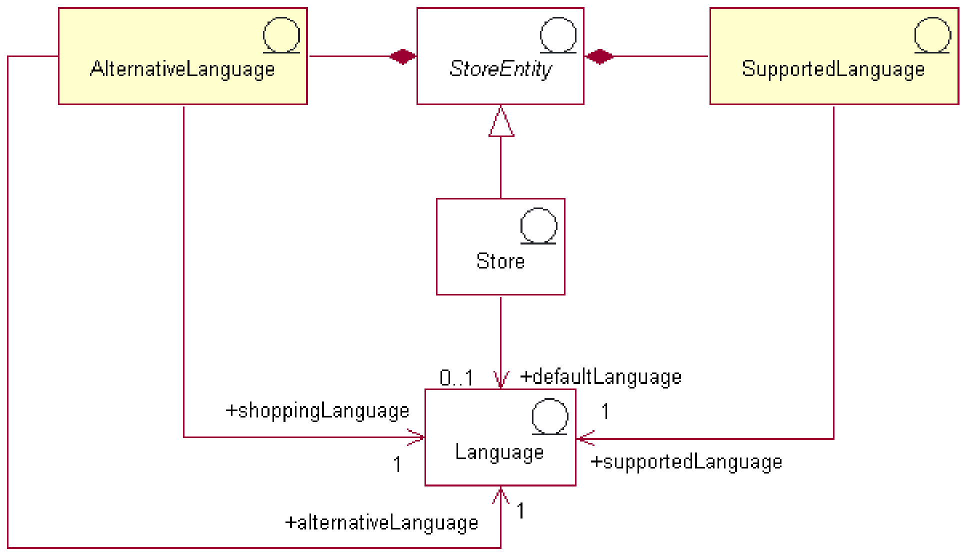
Chapter 14. Language assets

In WebSphere Commerce, your site can define many languages which can be used within it. The LANGUAGE table defines ten supported languages including German, Traditional and Simplified Chinese, Japanese, Korean, Italian, French, Spanish, Brazilian Portuguese, and English. Sites can define additional languages, or dialects of existing languages, to tailor the way information is presented to customers from different cultures or demographics.

Understanding language assets in WebSphere Commerce

In order to understand language assets, it is necessary to understand the relationships between languages and the store. This can be explained by the use of an information model below. The following section describes the relationships and associations language has to a store and other assets.

The diagram below depicts the language asset information model.



There are four classifications of languages in WebSphere Commerce. They are default languages, supported languages, alternative languages, and shopping languages. Each one of these classifications performs a different role in the store. All languages are stored in the LANGUAGE table.

Default language

A *default language* is associated with each store. This is the language that the store has chosen to use as it's main language, and will be the language displayed to customers that do not explicitly choose a shopping language. The default language for a store is implicitly supported by the store; that is, the store must always be able to display information in the default language, or one of its alternative languages, if any are defined in the LANGPAIR table. When information is not available in one of its supported languages, or alternative languages, the information will be displayed in the default language.

Supported language

The STORELANG table indicates the languages each store supports. A store must be able to display information in its *supported languages*, or one of their alternative languages, if any are defined in the LANGPAIR table. A store also supports all languages supported by its store group.

Alternative language

When information is not available in the one of the supported languages the store tries to display the information in an *alternative language*, if it is available. A store can specify the sequence in which to try each of its alternative languages. The alternative languages for a store include the alternative languages for its store group. Alternative languages can be useful when some information is available in only one language, but should be made available to customers shopping in a different, related, language. This might be the case when, for example, not all information has yet been translated into all supported languages, or when, for example, two very similar dialects of the same language are supported, sometimes with identical information.

For more detailed information on the structure of language assets in WebSphere Commerce Server, see the language object and data models in the WebSphere Commerce online help.

Creating language assets in WebSphere Commerce

You can define the languages your store supports in one of the following ways:

- Using the tools in Store Services
- In an XML file that will be loaded by the Loader package, or by the publishing tool in Store Services

Note: The Store Services tools work with pre-populated XML files in the form of a store archive.

For more information on defining store supported languages using Store Services, see the WebSphere Commerce online help. For more information on defining store supported languages in an XML file, see “Creating store data assets in an XML file” on page 42.

Chapter 15. Currency assets

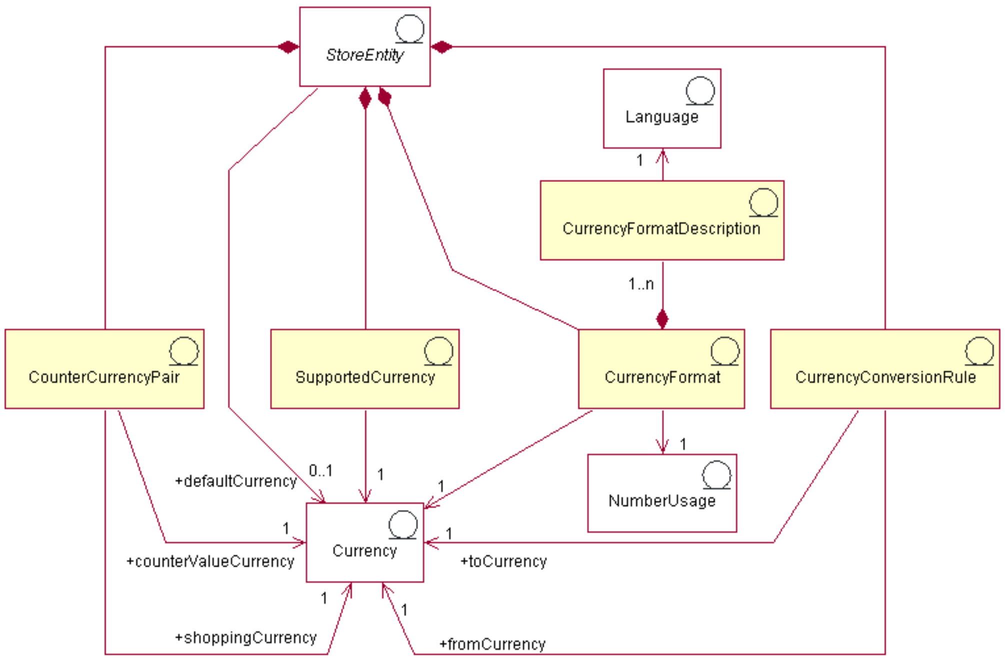
You can display prices in your site in one currency, or you can use multiple currencies by following the instructions provided for the euro. For a site with multiple stores, you can use different currencies for the stores, or you can assign currencies to the store group. Depending on the nature of the site that you are creating, you can specify what currencies you want to use and how they are displayed.

In WebSphere Commerce, you can allow customers to select a shopping currency. The shopping currency is the currency in which customers pay for products at a specific store. All monetary amounts on the store pages are displayed in this currency. When customers change their shopping currency, the prices for the items that they have added to their shopping carts and their order totals are automatically converted, recalculated, and displayed in the new shopping currency.

Customers can shop in many currencies, including the euro. The euro became the legal currency for the European Union on January 1, 1999, and is now used in financial markets. The conversion rates between the euro and the currencies of all participating countries are fixed. Notes™ and coins for the European Union's future single currency will become available on January 1, 2002. Six months later, the existing national currencies will be withdrawn from circulation permanently. During the transition period from 1999 to 2001, merchants must accept both the national currencies and the euro.

Understanding currency assets in WebSphere Commerce

The following diagram illustrates the currency structure in the WebSphere Commerce Server:



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

In the diagram above, currency is at the center of the information model. Each store, or group of stores, has a default currency.

Currency format

A store entity can have many *currency formatting* rules. If a store does not have a formatting rule for a particular currency, it uses the formatting rule of its store group. Currency formats are set up in the CURFORMAT table.

Number usage

Each formatted currency rule is associated with one *number usage*. Numbers such as quantities and monetary amounts can be rounded and formatted differently depending on their associated usage. Stores can specify different rounding and formatting rules for the numbers they display according to how they are used, such as a store may round unit prices to four decimal places by specifying the unit price usage, but other currency amounts to two decimal places by specifying the default usage. Number usage is stored in the NUMBRUSG table.

Currency format description

A currency format rule can have many *currency format descriptions*. A currency format description describes how to format (for display purposes) a monetary amount in a particular currency and particular language. Each description is associated with a language in the LANGUAGE table. For more information on language assets see, Chapter 14, "Language assets" on page 117. Currency format descriptions are stored in the CURFMTDESC table.

Supported currency

A store entity can have many *supported currencies*. A supported currency is one in which payment is accepted.

Currency conversion rule

All currencies have rules governing their conversions to and from other currencies. Each *currency conversion rule* can be used to convert a price (stored in the database in a particular currency) to an amount customers will be charged in a supported shopping currency.

Counter currency

Counter currencies are currency amounts that are displayed along with a supported currency. They cannot be used for purchases but are used for informational purposes. If customers decide to shop in the euro, they can have the European Monetary Union monetary amounts, and other currency amounts displayed in the store. Amounts in the shopping currency are converted to all the counter value currencies for that shopping currency. The counter currencies are paired with a supported currency such as the Netherlands guilder, and the euro. Counter currency pairs are stored in the CURCVLIST table.

For more detailed information on the structure of currency assets in WebSphere Commerce Server, see the currency data model in the WebSphere Commerce online help.

Creating currency assets in WebSphere Commerce

The Store Service tools in WebSphere Commerce allow you to add supported currencies to your store and to select a default currency for your store. For more information on which assets you can edit with the Store Services tools, see the WebSphere Commerce online help topic *"Changing store database assets."*

Note: The Store Services tools work with pre-populated XML files in the form of a store archive.

You can also add supported currencies and a default currency to your store using an XML file that can be loaded into the database using the Loader package. This method also allows you to create other types of currency assets, including defining currency conversion rates, and counter value currencies.

For information on editing the currency assets in an existing store archive, see the WebSphere Commerce online help. For information on creating new currency assets in the form of an XML file, see "Creating currency assets using an XML file".

Creating currency assets using an XML file

Create the currency assets for your store in the format of XML files that can be loaded into the database using the Loader package. For more information on the Loader package, see Part 7, "Publishing your store" on page 207.

To create currency assets for your store using an XML file, do the following:

1. Review the XML files used to create currency assets for the sample stores. All files for the sample stores are located in the corresponding store archive file.

   The store archive files are located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
   - AIX `/usr/WebSphere/CommerceServer/samplestores`
   - Solaris `/opt/WebSphere/CommerceServer/samplestores`
   - Linux `/opt/WebSphere/CommerceServer/samplestores`
   - 400 `/qibm/proddata/WebCommerce/samplestores`

   Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

   Each sample store includes a `currency.xml` file, which includes the currency information. To view the `currency.xml` files in the store archive, decompress it using a ZIP program. The `currency.xml` files are located in the data directory.
2. Review the information in Appendix B, "Creating your data" on page 305.

3. Create a `currency.xml` file, either by copying one of the `currency.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `currency.xml`. The DTD files 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/CommercServer/xml/sar`
   - Linux `/opt/WebSphere/CommercServer/xml/sar`
   - 400 `/qibm/proddata/WebCommerce/xml/sar`
4. Define the currencies supported by your store.
   a. Using the following example as your guide, define the currencies supported by your store in your XML file for the CURLIST table:
      ```
      <curlist currstr="USD" storeent_id="@storeent_id_1" />
      ```
      where:
      - `currstr` is the 3 character ISO 4217 currency code representing the supported currency. This code must appear in the SETCCURR column of the SETCURR table. A store must be able to accept payment in all its supported currencies.
      - `storeent_id` is the store entity.

   b. Repeat for each currency supported by store.

The default currency for the store is defined in the STOREENT table. For more information, see "Creating store data assets in an XML file" on page 42.

5. (Optional) What currency prices in your store display in depends on how you set up your prices. You can define prices for every currency used in your store, or you can define prices for the default currency only. For more information on setting up prices, "Creating pricing assets in WebSphere Commerce" on page 88.

   If when setting up prices, you defined prices for the default currency only, yet want to display prices in your store in other supported currencies, you must add conversion rates to your store. Use this conversion rate to convert from the default currency to the supported currency.

   a. Determine the currency from which you will be converting, for example, US dollar (USD), and the currency or currencies to which you are converting, for example the Yen (JPY). To determine the ISO currency codes for each currency, see ISO 4217 codes for international currencies.
   b. Using the following example as your guide, add conversion information to the CURCONVERT table:
      ```
      <curconvert
      storeent_id="@storeent_id_1"
      fromcurr="USD"
      tocurr="JPY"
      factor="105.10"
      multiplyordivide="M"
      bidirectional="Y"
      updatable="Y"
      curconvert_id="@curconvert_id_1" />
      ```

where:

- `storeent_id` is the store entity.
- `fromcurr` is the currency from which you are converting. An amount in the FROMCURR currency is normally part of a rule or other information used to determine a price, discount, shipping charge, or similar amount associated with a product offered for sale.
- `tocurr` is the currency to which you are converting. TOCURR is normally the currency in which the customer intends to pay. Amounts in this currency are normally part of an order item, such as a unit price, shipping charge, or tax amount.
- `factor` is the conversion factor.
- `multiplyordivide` is as follows: To convert from FROMCURR to TOCURR:
  - M = Multiply by FACTOR
  - D = Divide by FACTOR

  For bidirectional rules, conversion from TOCURR to FROMCURR is allowed using the inverse operation.
- `bidirectional` indicates whether the rule is bidirectional or unidirectional:
  - Y = bidirectional
  - N = unidirectional
- `updatable` is a flag intended to be used by a user interface that manages currency conversion rules. Valid values:
  - N = conversion rate is irrevocable - should never be changed
  - Y = conversion rate can be changed
- `curconvert_id` is a generated unique key.

c. Repeat steps a and b for all currencies in which you want to display prices.

Even if you have defined prices for all supported currencies in your pricing information, you may want to define the currency conversion rates for the supported currencies in your store.

---

6. (Optional) If you want to include display prices both in the shopping currency, and a counter currency (for example, display prices in both the Netherlands guilder and the euro), you must add information to the CURCVLIST table.

   a. Using the following example as your guide, add conversion information to the CURCVLIST table:

```
<curcvlist
storeent_id="@storeent_id_1"
currstr="NLG"
countervaluecurr="EUR"
displayseq="1" />
```

   where:

   - `storeent_id` is the store entity.
   - `currstr` is the three character ISO 4217 currency code representing the currency. This code must appear in the SETCCURR column of the SETCURR table is the currency from which you are converting. An amount in the FROMCURR currency is normally part of a rule or other information used to determine a price, discount, shipping charge, or similar amount associated with a product offered for sale.

- `countervaluecurr` is the three character ISO 4217 currency code representing the counter value currency. This code must appear in the SETCCURR column of the SETCURR table.
- `displayseq` is the number which indicates the presentation order of the counter value currency. Counter value currencies are displayed in ascending order based on the counter value display sequence specified in the DISPLAYSEQ column in the CURCVLIST table.

For more information about the use of **@** and **&** see Appendix B, "Creating your data" on page 305.

Other currency tasks

For more information on currency in general and on other currency tasks, including:

- Adding new currencies not currently supported by WebSphere Commerce
- Changing existing currency formats
- Adding new format rules for new currencies or for a specific store

see the WebSphere Commerce online help.

Chapter 16. Units of measure assets

Products can be sold, and inventory tracked, in a variety of quantity units, such as kilograms, inches, liters, and so on. Of these units, products can be ordered in minimum quantities, and by multiples of specific quantities.

The controller commands use the UOM (unit of measure) to specify the quantity unit. If a UOM parameter is not specified, then the customer's specified quantity is multiplied by the nominal quantity of the catalog entry in the CATENTSHIP database table. The result is known as the requested quantity.

The requested quantity is rounded up to the next highest quantity multiple for the catalog entry. For example, if the multiple is 2 kilograms and the requested quantity is 4.1 kilograms, the result of the rounding would be 6 kilograms. The rounded quantity is used when checking inventory, which has its own quantity unit. If the inventory quantity unit and the catalog entry quantity unit are different, there must be a conversion between the two units.

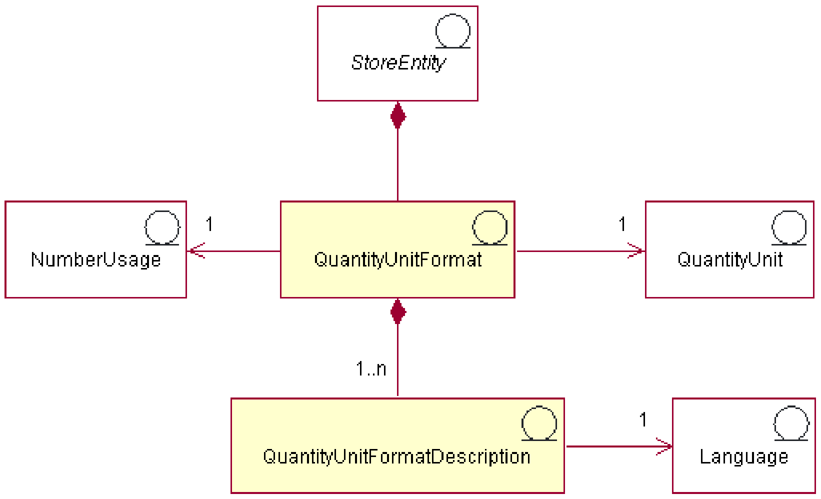
When Available to Promise (ATP) inventory is enabled (refer to the ALLOCATIONGOODFOR column of the STORE table), the inventory quantity unit is defined in the QUANTITYMEASURE column of the BASEITEM table. Otherwise, it is defined in the QUANTITYMEASURE column of the INVENTORY table.

The rounded quantity divided by the nominal quantity of the catalog entry is known as the normalized quantity. The normalized quantity is stored in the order item or the interest item, depending on the command being run. For example, if the rounded quantity is 6 kilograms and the nominal quantity is 2 kilograms, then the normalized quantity is 3.

When finding an offer for a catalog entry, the requested quantity can affect which offer gives the best price, and hence determines which offer will be used. For example, if the rounded quantity is 6 kg and there are two offers, one that specifies a price of $4.00 for the nominal quantity of 2 kilograms and a minimum quantity of 10 kilograms, and another that specifies a price of $4.50 for the nominal quantity of 2 kilograms and a minimum quantity of 2 kilograms, then only the second offer can be used.

Understanding units of measure in WebSphere Commerce

The following diagram illustrates the structure of units of measure in the WebSphere Commerce Server:



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

Quantity unit and quantity unit format

A *quantity unit* is the unit of measurement used in the store, for example, kilograms, pounds, meters, inches, liters, and so on. The quantity unit format is how this quantity unit is formatted in the store, for example how many decimal places are used when displaying the quantity unit.

Each *quantity unit format* is part of only one store entity, but each store entity may have several quantity unit formats.

A quantity unit format can exist for each quantity unit and number usage, and may have one or more quantity unit format descriptions, depending on how many languages the store supports.

Quantity unit format description

A *quantity unit format description* describes how to format (for display purposes) a quantity amount in a particular quantity unit, in a particular language.

Number usage

Number usage defines the way a number is used in an application. For example, by using number usage codes in your WebSphere Commerce code, you can choose the way you would like that number (currency or quantity) to be formatted or rounded. These codes (defined in the NUMBRUSG table) allow the number to be formatted according to the rules specified for that type of number usage in the CURFORMAT, CURFMTDESC, QTYFORMAT and QTYFMTDESC tables. This allows stores to format numbers in different ways to meet the requirements of a variety of situations.

For more detailed information on the structure of unit of measure assets in WebSphere Commerce Server, see the quantity unit data model in the WebSphere Commerce online help.

Creating units of measure in WebSphere Commerce

Units of measure are pre-populated in the WebSphere Commerce Server database when an instance is created. For more information, see Chapter 5, “Site assets” on page 37.

You can also define new units of measure in WebSphere Commerce for use in your store, or delete units of measure that you do not want to use in your store.

To define new units of measure for use in your store, add information to the following database tables:

- QTYUNIT
- QTUNITDSC
- QTYFORMAT
- QTYFMTDESC
- QTYUNITMAP
- QTYCONVERT

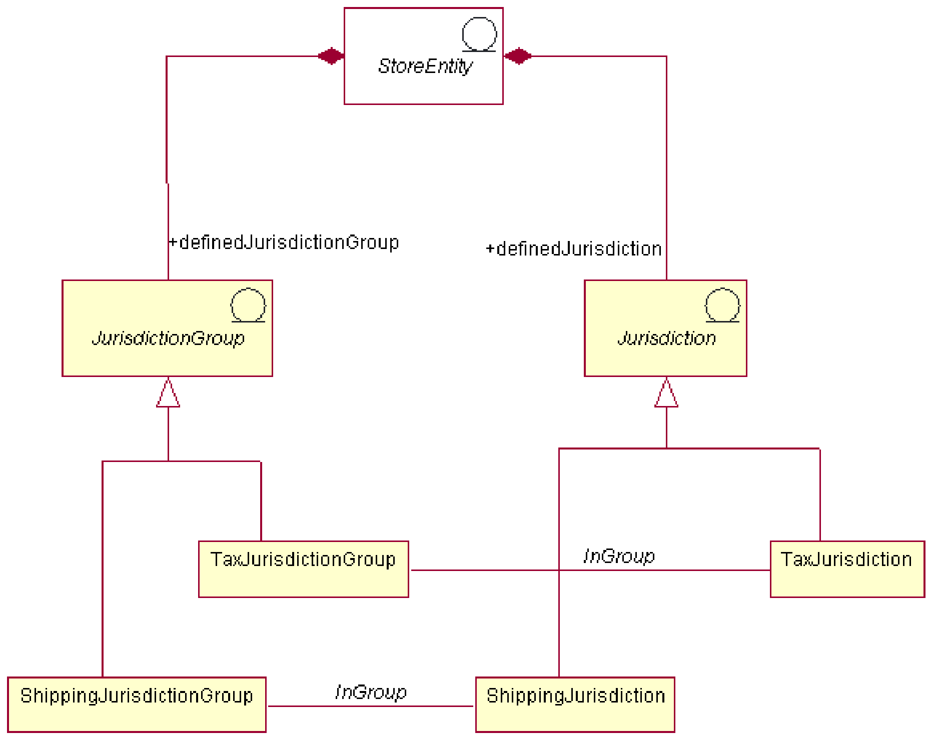
Chapter 17. Jurisdiction assets

Jurisdictions are geographical regions or zones representing a country or region, province or territory, or zip code range, to which you sell goods. Jurisdictions can be grouped together to form *jurisdiction groups*.

Jurisdiction groups are used in the calculation of the shipping charge and tax charges on orders. That is, a jurisdiction group can be used to qualify shipping charges and tax calculation rules used. These qualified calculation rules are applicable to items in an order only if the item is being shipped to an address within one of the jurisdictions in a jurisdiction group that is associated with the calculation rule. As a result, shipping charges and tax amounts may be calculated differently depending on the shipping addresses for the different items in the order.

Understanding jurisdiction assets in WebSphere Commerce

The following diagram illustrates how jurisdictions and jurisdictions groups fit into the WebSphere Commerce Server.



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

In WebSphere Commerce a jurisdiction or jurisdiction group is part of a store, and is exclusive to the store or store group for which it is created. For example, if you create three jurisdictions for your store, and then delete your store, the jurisdictions are also deleted. They are not available for use by any other existing stores, or any stores you might create in the future.

However, if you create jurisdictions for a store group, jurisdictions are not deleted when the stores in that group are deleted. The jurisdictions would be available for new stores created in that store group.

WebSphere Commerce supports two types of jurisdictions: shipping jurisdictions and tax jurisdictions. Shipping jurisdictions can be grouped together to form shipping jurisdiction groups, which qualify shipping charge calculation rules. Similarly, tax jurisdictions can be grouped together to form tax jurisdiction groups, which qualify tax calculation rules.

For more detailed information on the structure of jurisdiction assets in theWebSphere Commerce Server, see the jurisdiction data model in the WebSphere Commerce online help.

Creating jurisdiction assets in WebSphere Commerce

You must create jurisdiction assets for your store in order to apply tax and shipping charges. For more information on creating jurisdictions, see “Creating tax assets in WebSphere Commerce” on page 150 or “Creating shipping assets in WebSphere Commerce” on page 133.

Once jurisdictions have been created for your store, you can edit them or create new ones, using the Tax and Shipping notebooks in Store Services.

Note: Store Services automatically creates a jurisdiction group for every jurisdiction it creates. Store Services creates jurisdictions for stores, but not for store groups.

Note: Store Services only works on pre-published data in the form of a store archive.

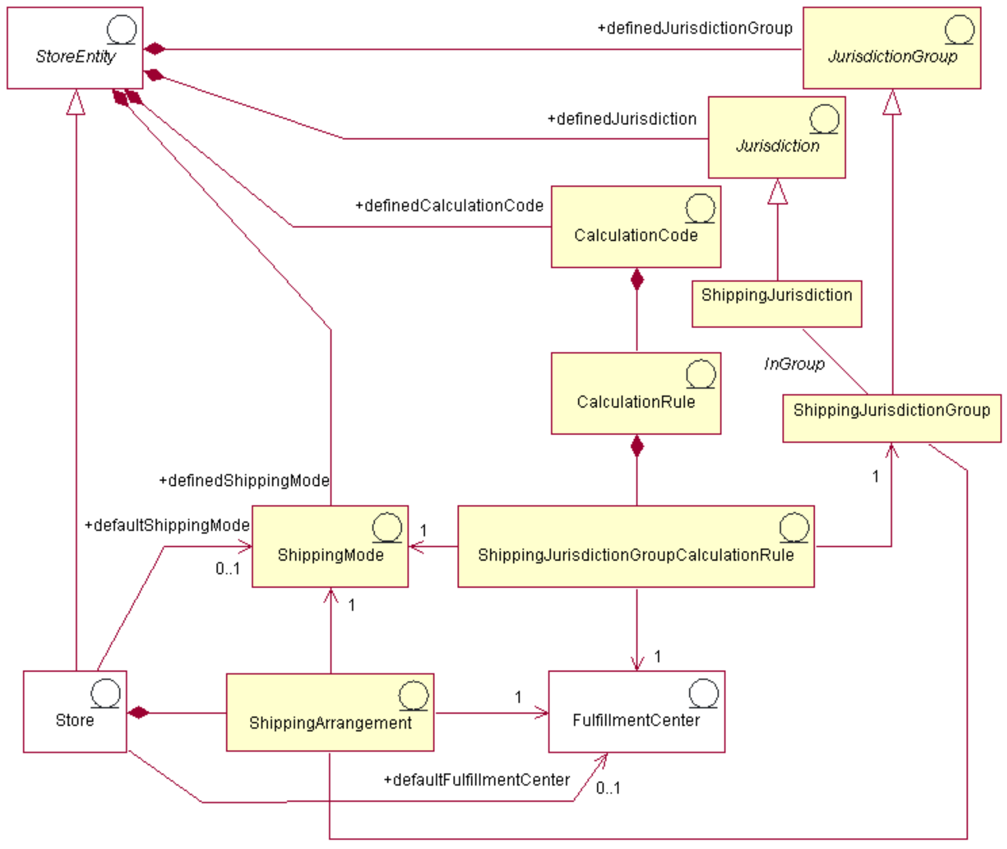
Chapter 18. Shipping assets

Shipping is how a store handles physically delivering goods to customers. In most cases, goods are shipped from a fulfillment center, a separate agency that is responsible for warehousing the store's goods.

In order to offer shipping services, and charge for these services, a store created with WebSphere Commerce should include the following:
- At least one shipping mode
- At least one shipping calculation code
- Jurisdictions and jurisdiction groups

Understanding shipping assets in WebSphere Commerce

The following diagram illustrates the shipping structure in the WebSphere Commerce Server.



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

Shipping modes

The *shipping mode* is a way of shipping goods. More specifically, a shipping mode is the combination of a shipping carrier (which is a company that provides shipping services from a fulfillment center to a customer), and the shipping service offered by that carrier. For example, ABC Shipping Company, Overnight service and ABC Shipping Company, Express delivery are shipping modes.

A shipping mode belongs to a store entity. If the store entity is deleted, the shipping modes defined within that store entity are also deleted. A store is not required to have a default shipping mode, but it is recommended.

Shipping arrangements

A *shipping arrangement* is an arrangement between the store and the fulfillment center, indicating that a fulfillment center will ship goods for a particular store using specified shipping modes. Certain restrictions can be placed on a shipping arrangement, including the time period for which the shipping arrangement is effective, and the shipping jurisdictions.

If a shipping arrangement is associated with a shipping mode, it applies only for that shipping mode. Otherwise, the shipping arrangement applies to all available shipping modes. A shipping arrangement is part of a store and will be deleted if the store is deleted.

Calculation codes

Calculation codes are used to calculate shipping charges, that is, a shipping calculation code indicates how shipping charges are calculated for order items. In order to calculate shipping charges on the order item, you must assign shipping calculation codes to either a catalog entry or a group of catalog entries.

A calculation code is part of a store entity. A calculation code can only be associated with one store entity, but a store entity may have several calculation codes. If the store entity is deleted, the calculation codes associated with that store entity are also deleted.

For more information about the use of calculation codes, see the *IBM WebSphere Commerce Calculation Framework Guide*.

Calculation rules

Each calculation code has a set of *calculation rules*. Shipping charges for an order item may vary depending on the shipping mode, fulfillment center, and which shipping jurisdictions. ShippingJurisdictionGroupCalculationRules are relationship objects that associate shipping calculation rules with jurisdictions, fulfillment centers, and shipping modes, to determine which calculation rules should be used for each order item.

If the calculation rule, or any of the other objects referred to by the ShippingJurisdictionGroupCalculationRules, is deleted, the ShippingJurisdictionGroupCalculation rule is also deleted.

For more information about the use of calculation rules, see the *IBM WebSphere Commerce Calculation Framework Guide*.

Jurisdictions and jurisdiction groups

Jurisdictions are geographical regions or zones representing a country or region, province or territory, or zip code range, to which you sell goods. Jurisdictions are grouped together to form *jurisdiction groups*.

WebSphere Commerce supports two types of jurisdictions: shipping jurisdictions and tax jurisdictions. Each of these jurisdictions is part of a corresponding group, for example, shipping jurisdictions are in the shipping jurisdictions group and tax jurisdictions are in the tax jurisdictions group.

Jurisdiction groups are associated with calculation rules. The calculation rule uses the jurisdiction group as part of the calculation to determine the shipping charge amount.

Jurisdictions and jurisdiction groups are part of a store entity. If the store entity is deleted, the jurisdictions and jurisdiction groups associated with that store entity are also deleted.

One shipping address may resolve to several shipping jurisdictions. For example, a shipping address in New York, United States will apply to the following shipping jurisdictions: ″New York, United States″, ″United States″, and ″World″. When a shipping address applies to multiple shipping jurisdictions, several shipping calculation rules will be applicable. In such cases, the precedence of the associated ShippingJurisdictionGroupCalculationRules is used to determine which rule or rules will be used.

For more detailed information on the structure of shipping assets in WebSphere Commerce Server, see the tax object and data models in the WebSphere Commerce online help.

Creating shipping assets in WebSphere Commerce

The Store Service tools in WebSphere Commerce allow you to create and edit certain shipping assets (for example shipping modes and jurisdictions) in a store archive, but not all shipping assets. For more information on which assets you can edit with the Store Services tools, see the WebSphere Commerce online help topic *″Changing store database assets.″*

Note: The Store Services tools work with pre-populated XML files in the form of a store archive.

You can also create your shipping assets in the format of XML files that can be loaded into the database using the Loader package. As a result, you have the following two options for creating shipping assets:

- Edit the existing shipping assets from one of the sample stores provided with WebSphere Commerce, or an existing store archive
- Create new shipping assets in the form of an XML file

For information on editing the shipping assets in an existing store archive, see the WebSphere Commerce online help. For information on creating new shipping assets in the form of an XML file, see “Creating shipping assets using an XML file” on page 134.

Creating shipping assets using an XML file

Create your shipping assets in the format of XML files that can be loaded into the database using the Loader package. For more information on the Loader package, see Part 7, "Publishing your store" on page 207. If you are creating a multicultural store, you may want to create separate XML files for each locale your store supports. The locale-specific file should specify all description information, so it can be easily translated.

The sample stores, from which many of the examples in these tasks are taken, use one `shipping.xml` file for all information that does not need to be translated, and another `shipping.xml` file for each locale the store supports, for the information that needs to be translated. The locale-specific files contain all the description information, so it can be easily translated.

To create shipping assets for your store using an XML file, do the following:

1. Review the *IBM WebSphere Commerce Calculation Framework Guide*. The WebSphere Commerce calculation framework calculates monetary amounts (for example, shipping) associated with the product or service a customer has selected to purchase.
2. Review the XML files used to create shipping assets for the sample stores. All files in the sample stores are located in the corresponding store archive file. Each sample store includes two or more `shipping.xml` files, which include the shipping information. The store archive files are located in the following directory:
   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
   - AIX `/usr/WebSphere/CommerceServer/samplestores`
   - Solaris `/opt/WebSphere/CommerceServer/samplestores`
   - Linux `/opt/WebSphere/CommerceServer/samplestores`
   - 400 `/qibm/proddata/WebCommerce/samplestores`

   Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

   To view the `shipping.xml` files in the store archive, decompress them using a ZIP program. The `shipping.xml` files are located in the data directory. The language-specific `shipping.xml` is in a locale-specific subdirectory of the data directory.
3. Review the information in Appendix B, "Creating your data" on page 305.
4. Create a `shipping.xml` file, either by copying one of the `shipping.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `shipping.xml`. The DTD files 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/CommercServer/xml/sar`
   - Linux `/opt/WebSphere/CommercServer/xml/sar`

- 400 /qibm/proddata/WebCommerce/xml/sar

5. Define the jurisdictions and jurisdiction group to which you are shipping goods and services. All jurisdictions must belong to a jurisdiction group.
   a. Using the following example as your guide, define a jurisdiction group in your XML file in the JURSTGROUP table:
      ```
      <jurstgroup
      jurstgroup_id="@jurstgroup_id_1"
      description="Jurisdiction Group1 for Shipping"
      subclass="1"
      storeent_id="@storeent_id_1"
      code="World"/>
      ```
      where
      - `jurstgroup_id` is a generated unique key
      - `description` is a brief description of the jurisdiction group, suitable for display in a user interface that manages jurisdiction groups.
      - `subclass` is the jurisdiction group subclass as follows:
        – 1 = ShippingJurisdictionGroup
        – 2 = TaxJurisdictionGroup
      - `storeent_id` is the store entity associated with this jurisdiction group.
      - `code` which, together with its store entity and subclass, uniquely identifies this jurisdiction group.
   b. Using the following example as your guide, define a jurisdiction in your XML file in the JURST table.
      ```
      < jurst
      jurst_id="@jurst_id_1"
      storeent_id="@storeent_id_1"
      code="World"
      subclass="1"/>
      ```
      where
      - `jurst_id` is a generated unique key
      - `storeent_id` is the store entity associated with this jurisdiction group.
      - `code` which, together with its store entity and subclass, uniquely identifies this jurisdiction group.
      - `subclass` is the jurisdiction subclass as follows:
        – 1 = ShippingJurisdiction
        – 2 = TaxJurisdiction
   c. Using the following example as your guide, associate the jurisdiction you created in step b with the jurisdiction group you defined in step a, by adding information to the JURSTGRPREL table.
      ```
      <jurstgprel
      jurst_id="@jurst_id_1"
      jurstgroup_id="@jurstgroup_id_1"
      subclass="1"/>
      ```
      where
      - `jurst_id` is the jurisdiction
      - `jurstgroup_id` is the jurisdiction group

- `subclass` is the subclass of the jurisdiction and of the jurisdiction group These should match:
  - 1 = ShippingJurisdiction[Group]
  - 2 = TaxJurisdiction[Group]

d. Repeat steps a through c for all jurisdictions and jurisdiction groups your store supports.

6. Define the shipping modes your store will use.

   a. Using the following example as your guide, define a shipping mode in your XML file for the SHIPMODE table:

```
<shipmode
shipmode_id="@shipmode_id_1"
field1
storeent_id="@storeent_id_1"
code="Ground 1 week"
carrier="XYZ Carrier"/>
```

   where:

   - `shipmode_id` is a generated unique key.
   - `field1` is a field available for customization.
   - `storeent_id` is the store entity associated with this shipping mode.
   - `code` is the merchant assigned code, unique for the store entity.
   - `carrier` is the name or identifier of the carrier.

   b. Using the following example as your guide, add information about the shipping mode to the SHPMODEDSC table. If you are creating a multicultural store, you should include this information in a locale-specific XML file:

```
< shpmodedsc
description="International mail"
field1="USD$5.00 per order plus USD$1.00 for each item"
field2="5 business days"
shipmode_id="@shipmode_id_1"
language_id="&en_US;"/>
```

   where:

   - `description` is a brief description of the ShippingMode, suitable for display to a customer for selection.
   - `field1 and field2` are fields available for customization.
   - `shipmode_id` is a generated unique key.
   - `language_id` is the language used.

   c. Repeat steps a and b for all shipping modes in your store.

7. Define the calculation codes to be used by your store.

   a. Using the following examples as your guide, define the calculation code in your XML file for the CALCODE table.

```
< calcode
calcode_id="@calcode_id_1"
code="shipping Code 1- per/order"
calusage_id="-2"
storeent_id="@storeent_id_1"
```

```
groupby="0″
published="1"
sequence="+0.00E+000"
calmethod_id="-23"
calmethod_id_app="-24"
calmethod_id_qfy="-22"
flags="0" />
```

where:

- `calcode_id` is a generated unique key.
- `code` is a character string that uniquely identifies this CalculationCode, given a particular CalculationUsage and StoreEntity.
- `calusage_id` indicates the kind of calculation this CalculationCode is used for. For example, the CalculationCode may be used to calculate one of the following monetary amounts:
  – Discounts (-1)
  – Shipping charges (-2)
  – Sales tax (-3)
  – Shipping tax (-4)
  – Coupons (-5)
- `storeent_id` is the store entity associated with this calculation code.
- `groupby` are bit flags indicating to the CalculationCodeCombineMethod how OrderItems should be grouped when performing calculations. 0 = No grouping. Place all applicable OrderItems in a single group. Refer to *CALCODE table: details* in the WebSphere Commerce online help for more information.
- `published` specifies whether or not the calculation code is published:
  – 0 = Not published (temporarily disabled)
  – 1 = Published
  – 2 = Marked for deletion (and not published)
- `sequence`CalculationCodes are calculated and applied in sequence from lowest to highest. If two calculation codes have the same sequence number, the calculation codes with the lower calcode_id will be calculated first.
- `calmethod_id` is the CalculationCodeCalculateMethod that defines how to calculate a monetary amount for this CalculationCode. calmethod_id=″-23″, the CalculationCodeCalculateMethod for shipping, is the only shipping calculation method provided with WebSphere Commerce.
- `calmethod_id_app` is the CalculationCodeApplyMethod that stores the calculated amount for the associated OrderItems. calmethod_id_app=″-24″, the CalculationCodeApplyMethod for shipping is the only shipping apply method provided with WebSphere Commerce.
- `calmethod_id_qfy` is the CalculationCodeQualifyMethod that defines which OrderItems are associated with this CalculationCode. calmethod_id_qfy=″-22″, the CalculationCodeQualifyMethod for shipping is the only shipping qualification method provided with WebSphere Commerce.
- `flags` specifies whether the CalculationCodeQualifyMethod of this CalculationCode should be invoked.

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

b. Using the following example as your guide, add the calculation code description information in your XML file for the CALCODEDSC table. If you are creating a multicultural store, you should include this information in a locale-specific XML file.

```
<calcodedsc
 calcode_id="@calcode_id_3"
 description="5.00USD per order"
 language_id="&en_US"
 longdescription= "This shipping calculation code charges 5.00USD per order."
 />
```

where

- `calcode_id` is the calculation code to which this information applies.
- `description` is a short description of the calculation code.
- `language_id` is the language for which this information applies.
- `longdescription` is the detailed description of the calculation code.

c. Repeat steps a and b for each calculation code used in your store.

8. Define the calculation rules for your store.

a. Using the following example as your guide, set up the calculation rule in your XML file for the CALRULE table:

```
<calrule
calrule_id="@calrule_id_1"
calcode_id="@calcode_id_1"
startdate="1900-01-01 00:00:00.000000"
enddate="2100-01-01 00:00:00.000000"
sequence="+1.00000000000000E+000"
combination="2"
calmethod_id="-27"
calmethod_id_qfy="-26"
flags="1"
identifier="1" />
```

where

- `calrule_id` is a generated unique identifier.
- `calcode_id` is the calculation code this calculation rule is part of.
- `startdate` is the time this calculation rule becomes effective.
- `enddate` is the time this calculation rule stops being effective.
- `sequence` is the order this calculation rule will be processed in. Calculation rules for the same calculation code are processed in sequence from lowest to highest value.
- `combination` specifies the bit flag for special processing to be performed by the default CalculationRuleCombineMethod implementation. Refer to the CALRULE table in the WebSphere Commerce online help for more information.
- `calmethod_id` is the CalculationRuleCalculateMethod that calculates a monetary result for a set of OrderItems.
- `calmethod_id_qfy` is the CalculationRuleQualifyMethod that determines which of a set of OrderItems should be sent to the CalculationRuleCalculateMethod.

- `flags` are used by CalculationRuleCombineMethod to determine how this calculation rule may be combined with other calculation rules. Refer to CALRULE table for more information.
- `identifier` identifies this calculation rule, in combination with its calculation code.

For more information see the CALRULE table in the WebSphere Commerce online help.

b. Repeat step a for each calculation rule used in your store. Note that each calculation code may have several calculation rules. For example, calcode_id=″@calcode_id_1″ may be associated with several calrule_ids.

9. Define calculation scales for your store.

   A calculation scale is the set of ranges that will apply 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. These ranges create a scale.

   a. Using the following example as your guide, set up the calculation scale in your XML file for the CALSCALE table:

```
<calscale
calscale_id="@calscale_id_1"
code="Scale Code 1 per order USD"
storeent_id="@storeent_id_1"
calusage_id="-2"
setccurr="USD"
calmethod_id="-28"/>
```

   where

   - `calscale_id` is a generated unique identifier.
   - `code` is a character string that uniquely identifies this calculation scale, given a particular calculation usage and store entity.
   - `storeent_id` is the store entity that this calculation scale is part of.
   - `calusage_id` indicates the kind of calculation this CalculationScale is used for. For example, the CalculationScale may be used to calculate one of the following monetary amounts:
     - Discounts (-1)
     - Shipping charges (-2)
     - Sales tax (-3)
     - Shipping tax (-4)
     - Coupons (-5)
   - `setccurr` if specified, indicates the currency for the range start values of the calculation range objects for this calculation scale. The CalculationScaleLookupMethod should return a ″lookup number″ in this currency.
   - `calmethod_id` is the CalculationScaleLookupMethod that given a set of order items determines a lookup value, a base monetary value, a result multiplier, and a set of mathematical weights that can be used by the calculation scale to calculate a monetary amount. To determine which CalculationScaleLookupMethod to use, do the following:
     - Refer to the CALMETHOD table in the WebSphere Commerce online help. Refer to the description for the SUBCLASS column. Click the link

for the CALMETHOD table: details. This table lists the types of calculation methods available. The MonetaryCalculationScaleLookupMethod method is 9.

- Open the bootstrap file `wcs.bootstrap_xx_XX.xml`, where `xx_XX` is the code for the locale. The bootstrap files are located in the following directory:
  - NT `drive:\WebSphere\CommerceServer\schema\xml`
  - 2000 `drive:\Program Files\WebSphere\CommerceServer\schema\xml`
  - AIX `/usr/WebSphere/CommerceServer/schema/xml`
  - Solaris `/opt/WebSphere/CommerceServer/schema/xml`
  - Linux `/opt/WebSphere/CommerceServer/schema/xml`
  - 400 `/qibm/proddata/WebCommerce/schema/xml`
- Locate the section listing the available calculation methods (CALMETHOD).
- Locate the calculation methods with the calusage_ID value for tax (-3 for sales tax and -4 for shipping tax).
- Locate the calculation methods, which have a subclass of 7; there are several. Pick the one which meets your needs.

For more information, see the CALSCALE table in the WebSphere Commerce online help.

b. Repeat step a for each calculation scale used in your store. For example, for shipping, InFashion creates a cost per order scale and a cost per item scale.

10. Define calculation ranges for the calculation scales.

    a. Using the following example as your guide, set up the calculation range in your XML file for the CALRANGE table.

```
<calrange
calrange_id="@calrange_id_1"
calscale_id="@calscale_id_1"
calmethod_id="-33"
rangestart="0.00000"
cumulative="0"/>
```

where

- `calrange_id` is a generated unique identifier.
- `calscale_id` is the calculation scale this calculation range is part of.
- `calmethod_id` is the CalculationRangeMethod that determines a monetary amount from the CalculationRangeLookupResult. For example, FixedAmountCalculationRangeCmd, PerUnitAmountCalculationRangeCmd, or PercentageCalculationRangeCmd. To determine the CalculationRangeMethod, do the following:
  - Refer to the CALMETHOD table in the WebSphere Commerce online help. Refer to the description for the SUBCLASS column. Click the link for the CALMETHOD table: details. This table lists the types of calculation methods available. The CalculationRangeMethod is 10.

– Open the bootstrap file `wcs.bootstrap_xx_XX.xml`, where `xx_XX` is the code for the locale. The bootstrap files are located in the following directory:
  - NT `drive:\WebSphere\CommerceServer\schema\xml`
  - 2000 `drive:\Program Files\WebSphere\CommerceServer\schema\xml`
  - AIX `/usr/WebSphere/CommerceServer/schema/xml`
  - Solaris `/opt/WebSphere/CommerceServer/schema/xml`
  - Linux `/opt/WebSphere/CommerceServer/schema/xml`
  - 400 `/qibm/proddata/WebCommerce/schema/xml`

– Locate the section listing the available calculation methods (CALMETHOD).

– Locate the calculation methods with the calusage_ID value for tax (-3 for sales tax and -4 for shipping tax).

– Locate the calculation methods which have a subclass of 9; there are several. Pick the one which meets your needs.

• `cumulative` are the valid values:
  – 0 = only the matching CalculationRange with the highest RANGESTART value is used.
  – 1 = all matching CalculationRanges are used. The calculated monetary amounts are summed to arrive at the final result.

For more information, see the CALRANGE table in the WebSphere Commerce online help.

b. Repeat step a for each calculation range associated with the calculation scale used in your store.

11. Define the calculation lookup values for the calculation scales. The calculation lookup values are the values associated with the calculation scale. For example, a calculation scale includes the following weight ranges and associated prices for shipping:
   - 0 to 5 kg costs $10.00
   - 5 to 10 kg costs $15.00

   The lookup values are $10.00 and $15.00.

   a. Using the following examples as your guide, set up the calculation lookup values in your XML file for the CALRLOOKUP table. If you are creating a multicultural store, you should include this information in a locale-specific XML file, that is, one file per locale that your store supports. For example, if your store ships to customers in the United States and Japan, you should add the US dollar lookup values in one XML file, and the Yen lookup values in another XML file.

```
<calrlookup
calrlookup_id="@calrlookup_id_1"
setccurr="USD"
calrange_id="@calrange_id_1"
value="5.00"/>
```

   where

   • `calrlookup_id` is a generated unique identifier.

- `calrange_id` is the calculation range this calculation range lookup result is part of.
- `value` is the value of the calculation range lookup result, used by the calculation range method of the calculation range to determine a monetary result.

For more information, see the CALRLOOKUP table in the WebSphere Commerce online help.

b. Repeat step a for each lookup value associated with the calculation scale used in your store.

12. Associate the calculation rule and calculation scale

a. Using the following examples as your guide, associate the calculate scale with the calculation rule in your XML file for the CRULESCALE table.

```
< crulescale
calrule_id="@calrule_id_1"
calscale_id="@calscale_id_1" />
```

where

- `calrule_id` is the calculation rule.
- `calscale_id` is the calculation scale.

b. Repeat step a for each calculation scale and rule association.

For more information about the use of @ and & see Appendix B, "Creating your data" on page 305.

Creating shipping fulfillment assets

In order for your shipping assets to work correctly in your store, you must associate the shipping jurisdiction groups to the calculation rules and the fulfillment centers to the shipping modes used in the store.

You must create your fulfillment assets before you can associate your shipping assets to a fulfillment center. For more information on creating fulfillment assets, see "Creating fulfillment assets in WebSphere Commerce" on page 108.

After you have created the fulfillment assets, associate shipping assets to them by adding information to the SHPJCRULE and SHPARRANGE tables. Do the following:

1. Review the *IBM WebSphere Commerce Calculation Framework Guide*. The WebSphere Commerce calculation framework calculates monetary amounts (for example, shipping) associated with the product or service a customer has selected to purchase.
2. Review the XML files used to create shipping fulfillment assets for the sample stores. All files for the sample stores are located in the corresponding store archive file. Each sample store includes a `shipfulfill.xml` file, which includes the shipping fulfillment information. To view the `shipfulfill.xml` file in the store archive, decompress it using a ZIP program. The `shipfulfill.xml` file is located in the data directory.

   The store archive files are located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`

- AIX `/usr/WebSphere/CommerceServer/samplestores`
- Solaris `/opt/WebSphere/CommerceServer/samplestores`
- Linux `/opt/WebSphere/CommerceServer/samplestores`
- 400 `/qibm/proddata/WebCommerce/samplestores`

Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

3. Review the information in Appendix B, "Creating your data" on page 305.
4. Create a `shipfulfill.xml` file, either by copying one of the `shipfulfill.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `shipfulfill.xml`. The DTD files 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`
   - Linux `/opt/WebSphere/CommerceServer/xml/sar`
   - 400 `/qibm/proddata/WebCommerce/xml/sar`
5. Associate calculation rules to a shipping jurisdiction group by adding information to the SHPJCRULE table. Use the following example as your guide. If you are creating a multicultural store, also create an XML file for each locale your store supports.
   ```
   <shpjcrule
   calrule_id="@calrule_id_1"
   ffmcenter_id="@ffmcenter_id_1"
   jurstgroup_id="@jurstgroup_id_1"
   precedence="0"
   shipmode_id="@shipmode_id_1"
   shpjcrule_id="@shpjcrule_id_1"
   ```
   where
   - `calrule_id` is the calculation rule used.
   - `ffmcenter_id` is the fulfillment center. If this is NULL then this association applies to all fulfillment centers.
   - `jurstgroup_id` is the shipping jurisdiction group. If this is NULL, then this association applies to all shipping jurisdiction groups.
   - `precedence` is when a shipping address falls within more than one of the specified shipping jurisdiction groups for the same fulfillment center and shipping mode. Only the calculation rule with the highest SHPJCRULE.PRECEDENCE value qualifies.
   - `shipmode_id` is the shipping mode.
   - `shpjcrule_id` is a generated unique identifier.
6. Repeat step 3 for each jurisdiction group, fulfillment center and rule association in your store.
7. Associate the shipping mode and a fulfillment center to your store, by adding information to the SHPARRANGE table. Use the following example as your guide:

```
<shparrange
 shparrange_id="@shparrange_id_2"
 store_id="@storeent_id_1"
 ffmcenter_id="@ffmcenter_id_1"
 shipmode_id= "@shipmode_id_2"
 startdate="1970-06-22 23:00:00.000000"
 enddate= "2008-06-22 23:00:00.000000"
 precedence= "0"
 flags="0"
/>
```

where

- `shparrange_id` is a generated unique identifier.
- `store_id` is the store.
- `ffmcenter_id` is the fulfillment center.
- `shipmode_id` is the shipping mode. NULL indicates this shipping arrangement can be used regardless of shipping mode.
- `startdate` is the time this shipping arrangement starts being effective.
- `enddate` is the time this shipping arrangement stops being effective.
- `precedence` is when more than one shipping arrangement (for the same store and shipping mode) is effective at a particular time; the one with the highest PRECEDENCE is used.
- `flags` contains bit flags:
  – 1 = restricted - This shipping arrangement applies only to order items whose shipping address matches one of the shipping jurisdiction groups associated (through the SHPARJURGP table) with this shipping arrangement.

8. Repeat step 5 for all shipping modes used in your store.

For more information about the use of @ and & see Appendix B, "Creating your data" on page 305.

Creating store-catalog-shipping assets

In order to associate shipping modes with your store, you must associate a calculation code with the catalog entries in your store for each contract your store includes.

You must create your store and catalog assets before you can create store-catalog-shipping assets. For more information on creating store assets, see "Creating store data assets in an XML file" on page 42. For more information on creating catalog assets, see "Displaying store catalog assets" on page 74.

To create store-catalog-shipping assets, do the following:

1. Review the *IBM WebSphere Commerce Calculation Framework Guide*. The WebSphere Commerce calculation framework calculates monetary amounts (for example, shipping) associated with the product or service a customer has selected to purchase.
2. Review the XML files used to create shipping fulfillment assets for the sample stores. All files for the sample stores are located in the corresponding store archive file.

   The store archive files are located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\samplestores`

- 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
- AIX `/usr/WebSphere/CommerceServer/samplestores`
- Solaris `/opt/WebSphere/CommerceServer/samplestores`
- Linux `/opt/WebSphere/CommerceServer/samplestores`
- 400 `/qibm/proddata/WebCommerce/samplestores`

  Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

  Each sample store includes a `store-catalog-shipping.xml` file, which includes the shipping fulfillment information. To view the `store-catalog-shipping.xml` file in the store archive, decompress it using a ZIP program. The `store-catalog-shipping.xml` file is located in the data directory.

3. Review the information in Appendix B, "Creating your data" on page 305.
4. Create a `store-catalog-shipping.xml` file, either by copying one of the `store-catalog-shipping.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `store-catalog-shipping.xml`. The DTD files 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`
   - Linux `/opt/WebSphere/CommerceServer/xml/sar`
   - 400 `/qibm/proddata/WebCommerce/xml/sar`
5. Create the store-catalog-shipping relationship by adding information to the CATENCALCD table. Use the following example as your guide:

```
<catencalcd
 calcode_id="@calcode_id_1"
 catencalcd_id="@catencalcd_id_1"
 store_id="@storeent_id_1"
/>
```

   where
   - `calcode_id` is the calculation code.
   - `catencalcd_id` is a generated unique identifier.
   - `store_id` is the store.

For more information about the use of @ and & see Appendix B, "Creating your data" on page 305.

Creating a default shipping mode

In order to set a default shipping mode for the store, you must add information to the STOREDEF table. To add information to the STOREDEF table, do the following:

1. Review the XML files used to create store default assets for the sample stores. All files for the sample stores are located in the corresponding store archive file.

The store archive files are located in the following directory:

- NT `drive:\WebSphere\CommerceServer\samplestores`
- 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
- AIX `/usr/WebSphere/CommerceServer/samplestores`
- Solaris `/opt/WebSphere/CommerceServer/samplestores`
- Linux `/opt/WebSphere/CommerceServer/samplestores`
- 400 `/qibm/proddata/WebCommerce/samplestores`

Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

Each sample store includes a `store-defaults.xml` file, which includes the default shipping information. To view the `store-defaults.xml` file in the store archive, decompress it using a ZIP program. The `store-defaults.xml` file is located in the data directory.

2. Review the information in Appendix B, "Creating your data" on page 305.
3. Create a `store-defaults.xml` file, either by copying one of the `store-defaults.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `store-defaults.xml`. The DTD files 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`
   - Linux `/opt/WebSphere/CommerceServer/xml/sar`
   - 400 `/qibm/proddata/WebCommerce/xml/sar`
4. Using the following example as your guide, in your XML file, specify the default shipping mode for the store by adding information to the STOREDEF table:

```
<storedef
   store_id="@storeent_id_1"
   shipmode_id="@shipmode_id_1"
/>
```

   where
   - `store_id` is the store.
   - `shipmode_id` is the default shipping mode for the store.

For more information about the use of @ and & see Appendix B, "Creating your data" on page 305.

Chapter 19. Tax assets

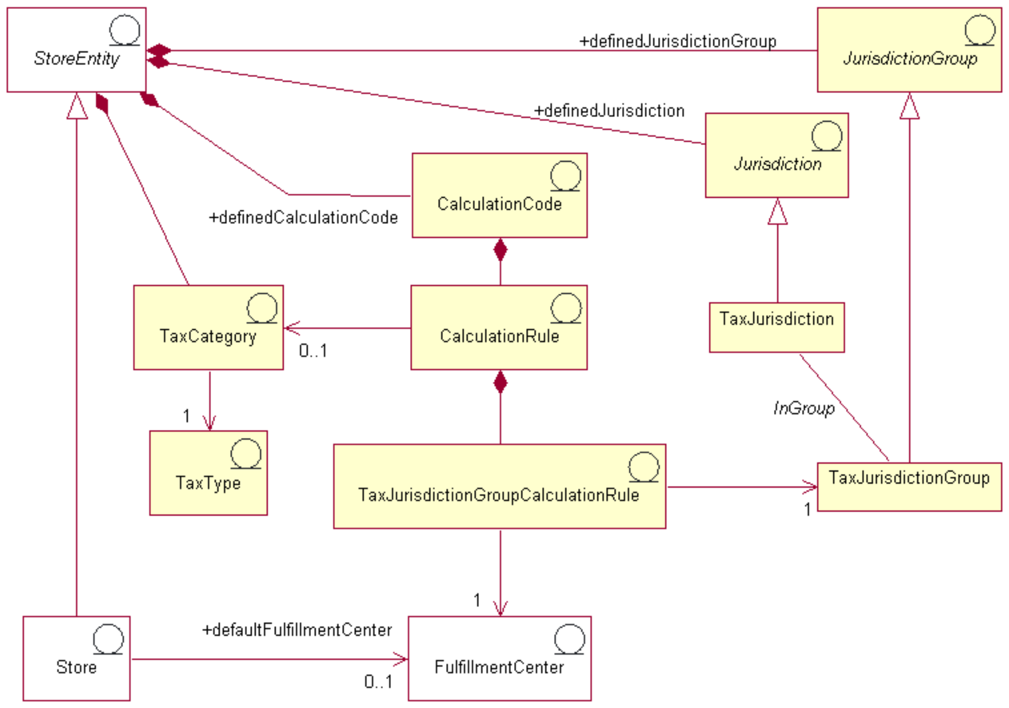
In order to charge and collect taxes on the goods and services your store provides, a store created with WebSphere Commerce must include the following:

- Tax categories
- Calculation codes
- Jurisdictions and jurisdiction groups

The combination of the tax categories, calculation codes, and jurisdictions and jurisdiction groups create the tax charges for the store.

Understanding tax assets in WebSphere Commerce

The following diagram illustrates the taxation structure in WebSphere Commerce Server.



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

Tax category

Tax categories correspond to the different kinds of tax a store may be required to collect, such as federal, state or provincial, and municipal.

A tax category is part of one store entity, although a store entity may have several tax categories. If the store entity is deleted, the tax categories associated with that store entity are also deleted.

Tax type

A store typically collects two type of taxes: sales or use tax, and shipping tax. Each tax category has one *tax type*. Although each tax category may only be of one tax type, (for example the tax category federal is a sales tax type), several different tax categories may belong to the same tax type (for example, the tax type sales tax, applies to the categories federal, provincial, and municipal).

Calculation code

Calculation codes are used to calculate tax charges, that is, a tax calculation code indicates how tax is calculated for order items. In order to calculate tax on the order item, you must assign sales tax and shipping tax calculation codes to either a catalog entry or a group of catalog entries. Only one tax calculation code of each tax type can be applied to a particular catalog entry or group of catalog entries. Typically, sales or use tax is levied on the net price, and shipping tax is levied on shipping charges.

A calculation code is part of a store entity. A calculation code can only be associated with one store entity, but a store entity may have several calculation codes. If the store entity is deleted, the calculation codes associated with that store entity are also deleted.

For more information about the use of calculation codes, see the *IBM WebSphere Commerce Calculation Framework Guide*.

Calculation rules

Each calculation code has at least one *calculation rule*, which defines the calculations for each tax category, and specifies the conditions under which the calculations will be done. Each tax calculation rule is associated with a tax category, a jurisdiction group and a fulfillment center, which together define the conditions under which the calculation rule is used. For example, a different rule may be selected to calculate an amount for a particular tax category depending on the shipping address and fulfillment center specified in the order.

Each calculation rule belongs to exactly one calculation code.

A particular tax calculation code can have several calculation rules, one for each combination of tax category, tax jurisdiction group, and fulfillment center associated with the store. Each sales tax and shipping tax calculation rule can be associated with multiple TaxJurisdictionGroupCalculationRules (TaxRules). For example in the chart below, calculation rule 10001 is applicable to both jurisdiction groups 1234 and 1235.

TAXJCRULE_ID	CALRULE_ID	FFMCENTER_ID	JURSTGROUP_ID	PRECEDENCE
10001	10001	NULL	1234	0
10002	10001	NULL	1235	0

Each TaxRule defines the conditions under which the calculation rule should be applied. For example, you may define a calculation rule for each jurisdiction group to which the store ships. In the example below, calculation rule 10001 is applicable to both jurisdiction group 1234 and 1235.

In the following example, the tax calculation code uses calculation rule A for the provincial sales tax category, when the tax jurisdiction is Alberta, and rule C when the tax jurisdiction is British Columbia.

Tax jurisdiction	**Federal sales tax**	**Provincial sales tax**
Alberta, Canada	Calculation rule B, which gives Y%	calculation rule A, which gives X%
British Columbia, Canada	Calculation rule B, which gives Y%	calculation rule C, which gives Z%

When a shipping address matches more than one tax jurisdiction group, the calculation rule with the highest associated TAXJCRULE.PRECEDENCE column value is used.

The association of TaxJurisdictionGroupCalculationRules (TaxRule) with a calculation rule determines when the calculation rule is applicable. A sales tax or shipping tax calculation rule is applicable when any one of the conditions given by the TaxRules is met. In the example below, calculation rule 10001 is applicable when you are shipping to jurisdiction group 1001, or when you are shipping from fulfillment center 1001, or you are shipping to jurisdiction group 1001.

CALRULE_ID	**FFMCENTER_ID**	**JURSTGROUP_ID**
10001	NULL	1001
10001	1001	1001

Each TaxJurisdictionGroupCalculationRule is associated with at most 1 jurisdiction group. Calculation rules themselves are not directly associated with jurisdiction groups.

For more information about the use of calculation rules, see the *IBM WebSphere Commerce Calculation Framework Guide*.

Jurisdictions and jurisdiction groups

Jurisdictions are geographical regions or zones representing a country or region, province or territory, or zip code range, to which you sell goods. Jurisdictions are grouped together to form *jurisdiction groups*.

WebSphere Commerce supports two types of jurisdictions: shipping jurisdictions and tax jurisdictions. Each of these jurisdictions is part of a corresponding group, for example, shipping jurisdictions are in the shipping jurisdictions group and tax jurisdictions are in the tax jurisdictions group.

Jurisdictions and jurisdiction groups determine which calculation rules are used to calculate the tax charges.

Jurisdictions and jurisdiction groups are part of a store entity. Each jurisdiction and jurisdiction group is part of one store entity, however a store entity may have several jurisdictions or jurisdiction groups. If the store entity is deleted, the jurisdictions and jurisdiction groups associated with that store entity are also deleted.

For more detailed information on the structure of tax assets in WebSphere Commerce Server, see the tax object and data models in the WebSphere Commerce online help.

Creating tax assets in WebSphere Commerce

The Store Service tools in WebSphere Commerce allow you to create and edit certain tax assets (for example tax categories and jurisdictions) in a store archive, but not all tax assets. For more information on which assets you can edit with the Store Services tools, see the WebSphere Commerce online help topic *"Changing store database assets."*

Note: The Store Services tools work with pre-populated XML files in the form of a store archive.

You can also create your tax assets in the format of XML files that can be loaded into the database using the Loader package. As a result, you have the following two options for creating shipping assets:

- Edit the existing tax assets from one of the sample stores provided with WebSphere Commerce, or an existing store archive
- Create new tax assets in the form of an XML file which can be published as part of a store archive, or loaded using the Loader package.

For information on editing the tax assets in an existing store archive, see the WebSphere Commerce online help. For information on creating new tax assets in the form of an XML file, see "Creating tax assets using an XML file"

Creating tax assets using an XML file

Create your tax assets in the format of XML files that can be loaded into the database using the Loader package. For more information on the Loader package, see Part 7, "Publishing your store" on page 207. If you are creating a multicultural store, you may want to create separate XML files for each locale your store supports. The locale-specific file should specify all description information, so it can be easily translated.

The sample stores, from which many of the examples in these tasks are taken, use one `tax.xml` file for all information that does not need to be translated, and another `tax.xml` file for each locale the store supports, for the information that needs to be translated. The locale-specific files contain all the description information

To create tax assets for your store using an XML file, do the following:

1. Review the *IBM WebSphere Commerce Calculation Framework Guide*. The WebSphere Commerce calculation framework calculates monetary amounts (for example, taxes) associated with the product or service a customer has selected to purchase.
2. Review the XML files used to create tax assets for the sample stores. All files for the sample stores are located in the corresponding store archive file.

   The store archive files are located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`

- AIX `/usr/WebSphere/CommerceServer/samplestores`
- Solaris `/opt/WebSphere/CommerceServer/samplestores`
- Linux `/opt/WebSphere/CommerceServer/samplestores`
- 400 `/qibm/proddata/WebCommerce/samplestores`

Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

Each sample store includes two `tax.xml` files, which include the tax information. To view the `tax.xml` files in the store archive, decompress it using a ZIP program. The `tax.xml` files are located in the data directory. The language-specific `tax.xml` is in a locale-specific subdirectory of the data directory.

3. Review the information in Appendix B, “Creating your data” on page 305.
4. Create a `tax.xml` file, either by copying one of the `tax.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `tax.xml`. The DTD files 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`
   - Linux `/opt/WebSphere/CommerceServer/xml/sar`
   - 400 `/qibm/proddata/WebCommerce/xml/sar`
5. Define the jurisdictions and jurisdiction groups to which you are shipping goods and services. Assign your tax jurisdictions to tax jurisdiction groups according to their applicable tax category calculation rules.
   a. Using the following example as your guide, define a jurisdiction group in your XML file in the JURSTGROUP table:
   ```
   <jurstgroup
   jurstgroup_id="@jurstgroup_id_2"
   description="Tax Jurstiction Group 1"
   subclass="2"
   storeent_id="@storeent_id_1"
   code="World"/>
   ```
   where
   - `jurstgroup_id` is a generated unique key
   - `description` is a brief description of the jurisdiction group, suitable for display in a user interface that manages jurisdiction groups.
   - `subclass` is the jurisdiction group subclass as follows:
     – 1 = ShippingJurisdictionGroup
     – 2 = TaxJurisdictionGroup
   - `storeent_id` is the store entity associated with this jurisdiction group.
   - `code` which, together with its store entity and subclass, uniquely identifies this jurisdiction group.

b. Using the following example as your guide, define a jurisdiction in your XML file in the JURST table.

```
<jurst
jurst_id="@jurst_id_2"
storeent_id="@storeent_id_1"
code="World"
subclass="2"/>
```

where

- `jurst_id` is a generated unique key
- `storeent_id` is the store entity associated with this jurisdiction group.
- `code` which, together with its store entity and subclass, uniquely identifies this jurisdiction group.
- `subclass` is the jurisdiction subclass as follows:
  - 1 = ShippingJurisdiction
  - 2 = TaxJurisdiction

c. Using the following example as your guide, associate the jurisdiction you created in step b with the jurisdiction group you defined in step a, by adding information to the JURSTGRPREL table.

```
<jurstgprel
jurst_id="@jurst_id_2"
jurstgroup_id="@jurstgroup_id_1"
subclass="2"/>
```

where

- `jurst_id` is the jurisdiction
- `jurstgroup_id`is the jurisdiction group
- `subclass` is the subclass of the jurisdiction and of the jurisdiction group These should match:
  - 1 = ShippingJurisdiction[Group]
  - 2 = TaxJurisdiction[Group]

d. Repeat steps a through c for all jurisdictions and jurisdiction groups your store supports.

6. Define the tax categories your store will use.

a. Using the following example as your guide, define a tax category in your XML file for the TAXCGRY table:

```
<taxcgry
 taxcgry_id="@taxcgry_id_1"
 taxtype_id="-3"
 storeent_id="@storeent_id_1"
 name="Sales Tax"
 displayseq="0"
 displayusage="0"/>
```

where:

- `taxcgry_id` is a generated unique key.
- `taxtype_id="-3"` is the tax type for this tax category. WebSphere Commerce supports two tax types:
  - sales or use tax (-3)
  - shipping tax (-4)
- `storeent_id` is the store entity associated with this tax category.

- name is the name of the tax category. Along with the store entity, the name uniquely identifies this tax category.
- displayseq specifies the sequence, from lowest to highest, of tax amounts when displayed, for example, in an order.
- displayusage specifies that this tax category in relation to the PriceDataBean as follows:
  - 0 = is not calculated
  - 1 = is calculated

  The PriceDataBean can be used to obtain tax amounts that should be shown along with the product price.

b. Repeat step a for each tax category used in your store.

c. Using the following example as your guide, add the tax category description information in your XML file for the TAXCGRYDS table. If you are creating a multicultural store, you should include this information in a locale-specific XML file.

```
<taxcgryds
 taxcgry_id="@taxcgry_id_1"
 description="Sales Tax"
 language_id="&en_US"/>
```

where

- taxcgry_id is the tax category.
- description is a brief description of the tax category, suitable for display to customers.
- language_id is the language in which this information will display.

d. Repeat step c for each tax category used in your store.

7. Define the calculation codes to be used by your store.

a. Using the following examples as your guide, define the calculation code in your XML file for the CALCODE table.

```
<calcode
 calcode_id="@calcode_id_3"
 code="Tax Code 1"
 calusage_id="-3"
 storeent_id="@storeent_id_1"
 groupby="0"
 published="1"
 sequence="0"
 calmethod_id="-43"
 calmethod_id_app="-44"
 calmethod_id_qfy="-42"
 displaylevel="0"
 flags="0"
 precedence="0"
 />
```

where:

- calcode_id is a generated unique key.
- code is a character string that uniquely identifies this calculation code, given a particular calculation usage and store entity.
- calusage_id indicates the kind of calculation this calculation code is used for. For example, the calculation code may be used to calculate one of the following monetary amounts:
  - Discounts (-1)
  - Shipping charges (-2)

  – Sales tax (-3)
  – Shipping tax (-4)
  – Coupons (-5)
- `storeent_id` is the store entity associated with this calculation code.
- `groupby` are bit flags indicating to the calculation code combine method how order items should be grouped when performing calculations. Zero specifies no grouping (all applicable order items are in a single group). Refer to *CALCODE table: details* in the WebSphere Commerce online help for more information.
- `published` specifies whether or not the calculation code is published:
  – 0 = not published (temporarily disabled)
  – 1 = published
  – 2 = marked for deletion (and not published)
- `sequence` is the order in which the calculation code is calculated. Calculation codes are calculated and applied in sequence from lowest to highest. If two calculation codes have the same sequence number, the calculation codes with the lower calcode_id will be calculated first.
- `calmethod_id`The calculation code calculate method that defines how to calculate the tax amounts for this calculation code. In order to determine which calculation code calculate method to use, do the following:
  – Refer to the CALMETHOD table in the WebSphere Commerce online help. Refer to the description for the SUBCLASS column. Click the link for the CALMETHOD table: details. This table lists the types of CALMETHODs available. The calculation code calculate method type is 3.
  – Open the bootstrap file `wcs.bootstrap_xx_XX.xml`, where `xx_XX` is the code for the locale. The bootstrap files are located in the following directory:
    - NT `drive:\WebSphere\CommerceServer\schema\xml`
    - 2000 `drive:\Program Files\WebSphere\CommerceServer\schema\xml`
    - AIX `/usr/WebSphere/CommerceServer/schema/xml`
    - Solaris `/opt/WebSphere/CommerceServer/schema/xml`
    - Linux `/opt/WebSphere/CommerceServer/schema/xml`
    - 400 `/qibm/proddata/WebCommerce/schema/xml`
  – Locate the section listing the available calculation methods (CALMETHOD).
  – Locate the calculation methods with the calusage_ID value for tax (-3 for sales tax and —4 for shipping tax).
  – Locate the calculation method which has the subclass of 3. This calculation method is —43.
- `calmethod_id_app` is the CalculationCodeApplyMethod that stores the calculated amount for the associated OrderItems. Use the method described in `calmethod_id` to determine which calculation code apply method to use.
  – calmethod_id_app="-44" is the CalculationCodeApplyMethod for Sales tax

- `calmethod_id_qfy` is the CalculationCodeQualifyMethod that defines which order items are associated with this calculation code. Use the method described in `calmethod_id` to determine which calculation code qualify method to use.
  - calmethod_id_qfy=″-42″ is the CalculationCodeQualifyMethod for Sales tax.
- `display level` determines if amounts calculated by this calculation code should be displayed with each:
  - 0 = OrderItem
  - 1 = Order
  - 2 = product
  - 3 = item
  - 4 = contract
- `flags` specifies whether the CalculationCodeQualifyMethod of this calculation code should be invoked.
  - 0 = unrestricted. The method will not be invoked
  - 1 = restricted. The method will be invoked.

b. Using the following example as your guide, add the calculation code description information in your XML file for the CALCODEDSC table. If you are creating a multicultural store, you should include this information in a locale-specific XML file.

```
<calcodedsc
 calcode_id="@calcode_id_3"
 description="Vitamins
 language_id="&en_US"
 longdescription= "In Ontario vitamins are taxed federally, but
not provincially."
 />
```

where

- `calcode_id` is the calculation code to which this information applies.
- `description` is a short description of the calculation code.
- `language_id` is the language for which this information applies.
- `longdescription`is the detailed description of the calculation code.

c. Repeat steps a and b for each calculation code used in your store.

8. Define the calculation rules for your store.

a. Using the following example as your guide, set up the calculation rule in your XML file for the CALRULE table:

```
<calrule
 calrule_id="@calrule_id_10"
 calcode_id="@calcode_id_3"
 startdate="1900-01-01 00:00:00.000000"
 taxcgry_id="@taxcgry_id_1"
 enddate="2100-01-01 00:00:00.000000"
 flags="1"
 identifier="1"
 combination="2"
 calmethod_id="-47"
 calmethod_id_qfy="-46"
 />
```

where

- `calrule_id` is a generated unique identifier.

- `calcode_id` is the calculation code this calculation rule is part of.
- `startdate` is the time this calculation rule becomes effective.
- `taxcgry_id` is the tax category for which this calculation rule is effective.
- `enddate` is the time this calculation rule stops being effective.
- `combination` are used by CalculationRuleCombineMethod to determine how this calculation rule may be combined with other calculation rules. Refer to CALRULE table for more information.
- `identifier` identifies this calculation rule, in combination with its calculation code.
- `flags`specifies the bit flag to indicate special processing to be performed by the default CalculationRuleCombineMethod implementation. Refer to the CALRULE table in the WebSphere Commerce online help for more information.
- `calmethod_id` is the CalculationRuleCalculateMethod that calculates a monetary result for a set of order items. To determine which calculation rule calculate method to use, do the following:
  - Refer to the CALMETHOD table in the WebSphere Commerce online help. Refer to the description for the SUBCLASS column. Click the link for the CALMETHOD table: details. This table lists the types of CALMETHODs available. The calculation rule calculate method is 7.
  - Open the bootstrap file `wcs.bootstrap_xx_XX.xml`, where `xx_XX` is the code for the locale. The bootstrap files are located in the following directory:
    - NT `drive:\WebSphere\CommerceServer\schema\xml`
    - 2000 `drive:\Program Files\WebSphere\CommerceServer \schema\xml`
    - AIX `/usr/WebSphere/CommerceServer/schema/xml`
    - Solaris `/opt/WebSphere/CommerceServer/schema/xml`
    - Linux `/opt/WebSphere/CommerceServer/schema/xml`
    - 400 `/qibm/proddata/WebCommerce/schema/xml`
  - Locate the section listing the available calculation methods (CALMETHOD).
  - Locate the calculation methods with the calusage_ID value for tax (-3 for sales tax and -4 for shipping tax).
  - Locate the calculation method which has the subclass of 7. This calculation method is -47.
- `calmethod_id_qfy` is the CalculationRuleQualifyMethod that determines which of a set of OrderItems should be sent to the CalculationRuleCalculateMethod. Use the method described in `calmethod_id` to determine which calculation rule qualify method to use.

b. Repeat step a for each calculation rule used in your store. Note that each calculation code may have several calculation rules, one for each applicable tax category. For example, calcode_id="@calcode_id_1" may be associated with several calrule_ids.

9. Define calculation scales for your store.

   A calculation scale is the set of ranges that will apply to the calculation. These ranges create a scale.

a. Using the following example as your guide, set up the calculation scale in your XML file for the CALSCALE table:

```
<calscale
 calscale_id="@calscale_id_19"
 code="Sales Tax 1"
 storeent_id="@storeent_id_1"
 calusage_id="-3"
 setccurr="USD"
 calmethod_id="-53"
 />
```

where

- `calscale_id` is a generated unique identifier.
- `code` is a character string that uniquely identifies this calculation scale, given a particular calculation usage and store entity.
- `storeent_id` is the store entity that this calculation scale is part of.
- `calusage_id` indicates the kind of calculation this CalculationScale is used for. For example, the CalculationScale may be used to calculate one of the following monetary amounts:
  - discounts (-1)
  - shipping charges (-2)
  - sales tax (-3)
  - shipping tax (-4)
  - coupons (-5)
- `setccurr` if specified, indicates the currency for the range start values of the calculation range objects for this calculation scale. The CalculationScaleLookupMethod will return a ″lookup number″ in this currency. In this case, it is not specified; the CalculationScaleLookupMethod will return a lookup number in the currency of the order. The currency does not need to be specified unless the scale range start values are non-zero.
- `calmethod_id` is the CalculationScaleLookupMethod that given a set of order items determines a lookup number, a base monetary value, a result multiplier, and a set of mathematical weights that can be used by the calculation scale to calculate a monetary amount. To determine which CalculationScaleLookupMethod to use, do the following:
  - Refer to the CALMETHOD table in the WebSphere Commerce online help. Refer to the description for the SUBCLASS column. Click the link for the CALMETHOD table: details. This table lists the types of CALMETHODs available. The MonetaryCalculationScaleLookupMethod method is 9.
  - Open the bootstrap file `wcs.bootstrap_xx_XX.xml`, where `xx_XX` is the code for the locale. The bootstrap files are located in the following directory:
    - NT `drive:\WebSphere\CommerceServer\schema\xml`
    - 2000 `drive:\Program Files\WebSphere\CommerceServer\schema\xml`
    - AIX `/usr/WebSphere/CommerceServer/schema/xml`
    - Solaris `/opt/WebSphere/CommerceServer/schema/xml`
    - Linux `/opt/WebSphere/CommerceServer/schema/xml`

- 400 /qibm/proddata/WebCommerce/schema/xml
- Locate the section listing the available calculation methods (CALMETHOD).
- Locate the calculation methods with the calusage_ID value for tax (-3 for sales tax and -4 for shipping tax).
- Locate the calculation method which has the subclass of 9. There are several calculation methods with the subclass of 9. Pick the one which meets your needs.

For more information, see the CALSCALE table in the WebSphere Commerce online help.

b. Repeat step a for each calculation scale used in your store.

c. Using the following example as your guide, add the calculation scale description information in your XML file for the CALSCALDS table. If you are creating a multicultural store, you should include this information in a locale-specific XML file.

```
<calscaleds
 calscale_id="@calscale_id_19"
 description="Sales Tax 5% "
 language_id="&en_US"
 />
```

where

- `calscale_id` is the calculation scale to which this description applies.
- `description`is a brief description of the calculation scale, suitable for display to customers to explain how a calculation is performed. For example, ″$.10 per kilogram, minimum charge of $5.00.″ or ″10% off quantities of 5 or more.″
- `language_id` is the language in which this information will display.

d. Repeat step c for each calculation scale used in your store.

10. Define calculation ranges for the calculation scales.

a. Using the following example as your guide, set up the calculation range in your XML file for the CALRANGE table.

```
<calrange
 calrange_id="@calrange_id_37"
 calscale_id="@calscale_id_19"
 calmethod_id="-59"
 rangestart="0.00000"
 cumulative="0"
 />
```

where

- `calrange_id` is a generated unique identifier.
- `calscale_id` is the calculation scale this calculation range is part of.
- `calmethod_id` is the CalculationRangeMethod that determines a monetary amount from the CalculationRangeLookupResult. For example, FixedAmountCalculationRangeCmd, PerUnitAmountCalculationRangeCmd, or PercentageCalculationRangeCmd. To determine the CalculationRangeMethod, do the following:
  - Refer to the CALMETHOD table in the WebSphere Commerce online help. Refer to the description for the SUBCLASS column. Click the link for the CALMETHOD table: details. This table lists the types of CALMETHODs available. The CalculationRangeMethod is 10.

– Open the bootstrap file `wcs.bootstrap_xx_XX.xml`, where `xx_XX` is the code for the locale. The bootstrap files are located in the following directory:
  - NT `drive:\WebSphere\CommerceServer\schema\xml`
  - 2000 `drive:\Program Files\WebSphere\CommerceServer\schema\xml`
  - AIX `/usr/WebSphere/CommerceServer/schema/xml`
  - Solaris `/opt/WebSphere/CommerceServer/schema/xml`
  - Linux `/opt/WebSphere/CommerceServer/schema/xml`
  - 400 `/qibm/proddata/WebCommerce/schema/xml`

– Locate the section listing the available calculation methods (CALMETHOD).

– Locate the calculation methods with the calusage_ID value for tax (-3 for sales tax and -4 for shipping tax).

– Locate the calculation method which has the subclass of 10. There are several calculation methods with the subclass of 10. Pick the one which meets your needs.

- `rangestart` is if a lookup number is greater than or equal to RANGESTART, or if RANGESTART is NULL, this row matches the lookup number.
- `cumulative` is the following:
  – 0 = only the matching CalculationRange with the highest RANGESTART value is used.
  – 1 = all matching CalculationRanges are used. The calculated monetary amounts are summed to arrive at the final result.

For more information, see the CALRANGE table in the WebSphere Commerce online help.

b. Repeat step a for each calculation range associated with the calculation scale used in your store. In the example above there is only one range, since all amounts are taxed at the same rate.

11. Define the calculation lookup values for the calculation scales. The calculation lookup values are the values associated with the calculation scale. For example, a calculation scale includes the following ranges and associated tax rates for Ontario provincial sales tax on meals served in a restaurant:
- $0.00 - $3.99 taxed at the rate of 0.00%
- $4.00 and up taxed at the rate of 8.00%

The lookup values are 0.00 and 8.00.

a. Using the following examples as your guide, set up the calculation lookup in your XML file for the CALRLOOKUP table.

```
<calrlookup
 calrlookup_id="@calrlookup_id_37"
 calrange_id="@calrange_id_37"
 value="5.00"
 />
```

where
- `calrlookup_id` is a generated unique identifier.
- `calrange_id` is the calculation range this calculation range lookup result is part of.

- `value` is the value of the calculation range lookup result, used by the calculation range method of the calculation range to determine a monetary result. In this example, the tax rate is 5.00%.

  For more information, see the CALRLOOKUP table in the WebSphere Commerce online help.

b. Repeat steps a and b for each lookup value associated with the calculation scale used in your store. In this example, there is only one CALRLOOKUP value, since CALRLOOKUP.SETCCURR is NULL, and there is only one CALRANGE, since the tax rate is the same for all amounts.

12. Associate the calculation rule and calculation scale.

    a. Using the following examples as your guide, associate the calculate scale with the calculation rule in your XML file for the CRULESCALE table.

```
<crulescale
  calrule_id="@calrule_id_10"
  calscale_id="@calscale_id_19"
 />
```

    where

    - `calrule_id` is the calculation rule.
    - `calscale_id` is the calculation scale.

    b. Repeat step a for each calculation scale and rule association. In example used above, there is only one calculation scale for each calculation rule.

    Note: If the tax rate varies depending on the amount purchased, you will need to create scales with non-zero rangestart values. Then, you will need to create a calculation scale for each supported currency (setting CALSCALE.SETCCURR to the appropriate currency) for which you have not established a conversion rate (refer to the CURCONVERT table) and associate them all with the calculation rule for that particular tax category. For example, there is no Ontario provincial sales tax on meals under $4.00. If your store supported selling meals in US dollars, you would need to either establish a conversion from US dollars to Canadian dollars, or create a separate tax calculation scale with an appropriate rangestart value, perhaps $6.00 USD, and associate it with the same tax calculation rule. Only the appropriate calculation scale would be used, according to the currency of the order.

For more information about the use of **@** and **&** see Appendix B, "Creating your data" on page 305.

Creating tax fulfillment assets

In order for your tax assets to work correctly in your store, you must associate the tax jurisdiction groups in your store to the fulfillment center used by your store, and then associate a calculation rule to both.

You must create your fulfillment assets before you can associate your tax assets to a fulfillment center. For more information on creating fulfillment assets, see "Creating fulfillment assets in WebSphere Commerce" on page 108.

After you have created the fulfillment assets, associate your tax assets to them, by adding add information to the TAXJCRULE table. Do the following:

1. Review the *IBM WebSphere Commerce Calculation Framework Guide*. The WebSphere Commerce calculation framework calculates monetary amounts (for example, taxes) associated with the product or service a customer has selected to purchase.
2. Review the XML files used to create tax fulfillment assets for the sample stores. All files for the sample stores are located in the corresponding store archive file.

   The store archive files are located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
   - AIX `/usr/WebSphere/CommerceServer/samplestores`
   - Solaris `/opt/WebSphere/CommerceServer/samplestores`
   - Linux `/opt/WebSphere/CommerceServer/samplestores`
   - 400 `/qibm/proddata/WebCommerce/samplestores`

   Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.

   Each sample store includes a `taxfulfill.xml` file, which include the tax information. To view the `taxfulfill.xml` file in the store archive, decompress it using a ZIP program. The `taxfulfill.xml` file is located in the data directory.
3. Review the information in Appendix B, "Creating your data" on page 305.
4. Create a `taxfulfill.xml` file, either by copying one of the `taxfulfill.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `taxfulfill.xml`. The DTD files 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`
   - Linux `/opt/WebSphere/CommerceServer/xml/sar`
   - 400 `/qibm/proddata/WebCommerce/xml/sar`
5. Using the following example as your guide, in your XML file add information for the TAXJCRULE table:

   ```
   <taxjcrule
   taxjcrule_id="@taxjcrule_id_1"
   calrule_id="@calrule_id_10"
   ffmcenter_id="@ffmcenter_id_1"
   jurstgroup_id="@jurstgroup_id_2"
   precedence="0"
    />
   ```

   where
   - `taxjcrule_id` is a generated unique identifier.
   - `calrule_id` is the calculation rule used.
   - `ffmcenter_id` is the fulfillment center. If this is NULL then this association applies to all fulfillment centers.

- `jurstgroup_id` is the tax jurisdiction group. If this is NULL, then this association applies to all tax jurisdiction groups.
- `precedence` is when a shipping address falls within more than one of the specified tax jurisdiction groups, for the same fulfillment center, only the calculation rule with the highest TAXJCRULE.PRECEDENCE value qualifies.

6. Repeat step 3 for each jurisdiction group, fulfillment center and rule association in your store.

For more information about the use of @ and & see Appendix B, "Creating your data" on page 305.

Creating store-catalog-tax assets

In order to associate taxes with the goods and services in your store, you must associate a calculation code with the catalog entries in your store for each contract your store includes.

You must create your store and catalog assets before you can create store-catalog-tax assets. For more information on creating store assets, see "Creating store data assets in an XML file" on page 42. For more information on creating catalog assets, see "Displaying store catalog assets" on page 74.

To create store-catalog-tax assets, do the following:

1. Review the *IBM WebSphere Commerce Calculation Framework Guide*. The WebSphere Commerce calculation framework calculates monetary amounts (for example, shipping) associated with the product or service a customer has selected to purchase.
2. Review the XML files used to create store-catalog-tax assets for the sample stores. All files for the sample stores are located in the corresponding store archive file.

   The store archive files are located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
   - AIX `/usr/WebSphere/CommerceServer/samplestores`
   - Solaris `/opt/WebSphere/CommerceServer/samplestores`
   - Linux `/opt/WebSphere/CommerceServer/samplestores`
   - 400 `/qibm/proddata/WebCommerce/samplestores`

   Each sample store includes a `store-catalog-tax.xml` file, which includes the shipping fulfillment information. To view the `store-catalog-tax.xml` file in the store archive, decompress it using a ZIP program. The `store-catalog-tax.xml` file is located in the data directory.

   Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.
3. Review the information in Appendix B, "Creating your data" on page 305.
4. Create a `store-catalog-tax.xml` file, either by copying one of the `store-catalog-tax.xml` files in the sample store archives, or by creating a new one. For more information, see the DTD file that corresponds to `store-catalog-tax.xml`. The DTD files 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`
- Linux `/opt/WebSphere/CommerceServer/xml/sar`
- 400 `/qibm/proddata/WebCommerce/xml/sar`

5. Create the store-catalog-tax relationship by adding information to the CATENCALCD table. Use the following example as your guide:

```
<catencalcd
   calcode_id="@calcode_id_3"
   catencalcd_id="@catencalcd_id_3"
   store_id="@storeent_id_1"
/>
```

   where

   - `calcode_id` is the calculation code.
   - `catencalcd_id` is a generated unique identifier.
   - `store_id` is the store.

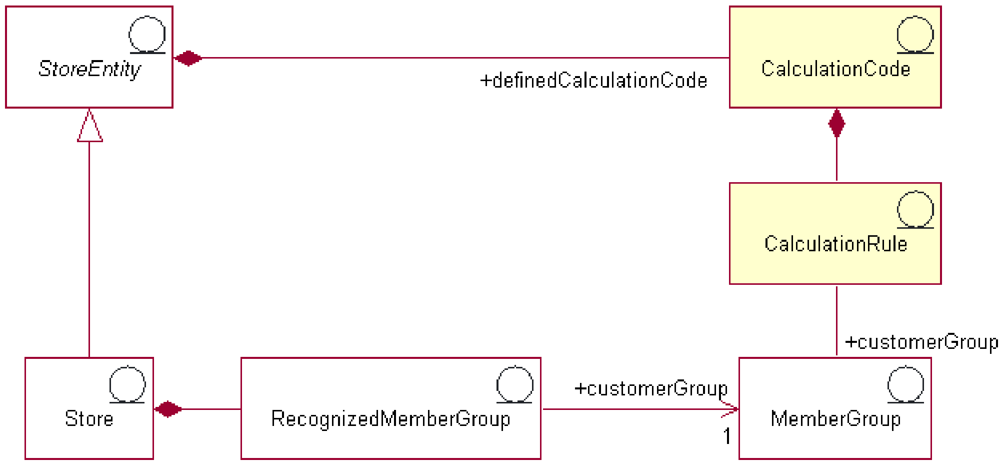
For more information about the use of @ and & see Appendix B, “Creating your data” on page 305.

Chapter 20. Discount assets

Discounts allow you to offer customers price incentives to promote a purchase. You can offer percentage discounts (such as 10% off), or fixed-amount discounts (such as $15 off). Discounts can apply to specific products or to the total purchase. For example, you can offer a 20% reduction to senior citizens; or if you have many red baseball caps in stock, you can offer a 25% discount on the caps for a limited time.

Understanding discounts in WebSphere Commerce

The following diagram illustrates the discount structure in the WebSphere Commerce Server.



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

Calculation code

A *discount* is represented by and calculated using a discount *calculation code*. A discount calculation code indicates how the discount is calculated for order items.

A calculation code belongs to a store entity. Multiple calculation codes can be defined within a store entity. If the store entity is deleted, the calculation codes defined within that store entity are also deleted.

Each discount calculation code can have a start date and an end date, which define the time period in which the discount is effective. The discount calculation code can also be associated with one or more member groups, which define the eligible member groups.

The discount calculation code can be attached to one or more catalog entries, and catalog groups. Attaching a calculation code to a catalog group has the same effect as attaching it to all the catalog entries directly in the catalog group. However, discount calculation codes attached to catalog group A are not attached to products and items in catalog group B if catalog group A contains catalog group B.

Catalog entries or catalog groups may have more than one discount associated with them. When more than one discount calculation code is applicable to an order, discount calculations are performed in ascending sequence of their calculation code sequence attributes.

Note: Define discount sequence orders to implement discounts on discounts.

The order items are grouped for calculation in one of the following ways:
- Per trading agreement
- Per product
- Per offer
- Per shipping address

For more information, see the WebSphere Commerce online help.

For more information about the use of calculation codes, see the *IBM WebSphere Commerce Calculation Framework Guide*.

Calculation rules

Each calculation code has a set of calculation rules, which define the conditions under which the calculation will be done. Each discount calculation rule is associated with one or more member groups, for whom the discount is effective. Member groups may be eligible for more than one discount at a time.

Note: If an eligible member group is defined at the calculation code level, it does not need to be defined again at the calculation rule level.

For more information about the use of calculation rules, see the *IBM WebSphere Commerce Calculation Framework Guide*.

Creating discount assets in WebSphere Commerce

The primary method of creating discounts in a store created with WebSphere Commerce is using the Discount wizard in the WebSphere Commerce Accelerator. For more information on creating discounts using the WebSphere Commerce Accelerator, see the WebSphere Commerce online help.

Discounts can also be created by using an XML file and then loaded by the Loader package, or published by Stores Services. However, discounts created in this manner, as well as discounts imported during migration from previous versions, will function correctly, but may not display properly in the WebSphere Commerce Accelerator.

Chapter 21. Inventory assets

Inventory includes anything that can be physically accounted for in a fulfillment center. There are specific definitions of types of inventory that can be fulfilled, such as items, products, SKUs, bundles, and packages; but these are all considered inventory. Products are configured for fulfillment in the Product wizard and the Product notebook. This includes options to track inventory, allow backorder, force backorder, release separately, and specify that the product should not be returned. The WebSphere Commerce Accelerator distinguishes between two major types of inventory that can be received:

- Expected inventory that has an associated expected inventory record.
- Ad hoc inventory, or inventory not recorded as expected.

Expected inventory is received from a vendor and typically paid for with a purchase order. The WebSphere Commerce Accelerator tracks expected inventory with expected inventory records, and allows you to record an external identifier, typically a purchase order number from an external system. In this way, you can easily keep track of the inventory you have ordered, as well as what has and has not arrived. Expected inventory details are the specifics about products in an expected inventory record, such as the fulfillment center expecting the product, the expected receipt date, quantity expected, and comments.

An expected inventory record cannot be deleted once inventory has been received against it, and expected inventory details cannot be changed or deleted once any of that inventory has been received.

When orders are placed for inventory that is available in a fulfillment center, the order system allocates inventory to those orders. Allocating inventory to an order makes it unavailable to the order system. If the order is canceled, the inventory becomes available again. If an order is placed for inventory that is not available, a backorder can be created. If there is expected inventory that could be used to fulfill the backorder, then the expected inventory is allocated to the backorder and the customer can be provided with an expected ship date.

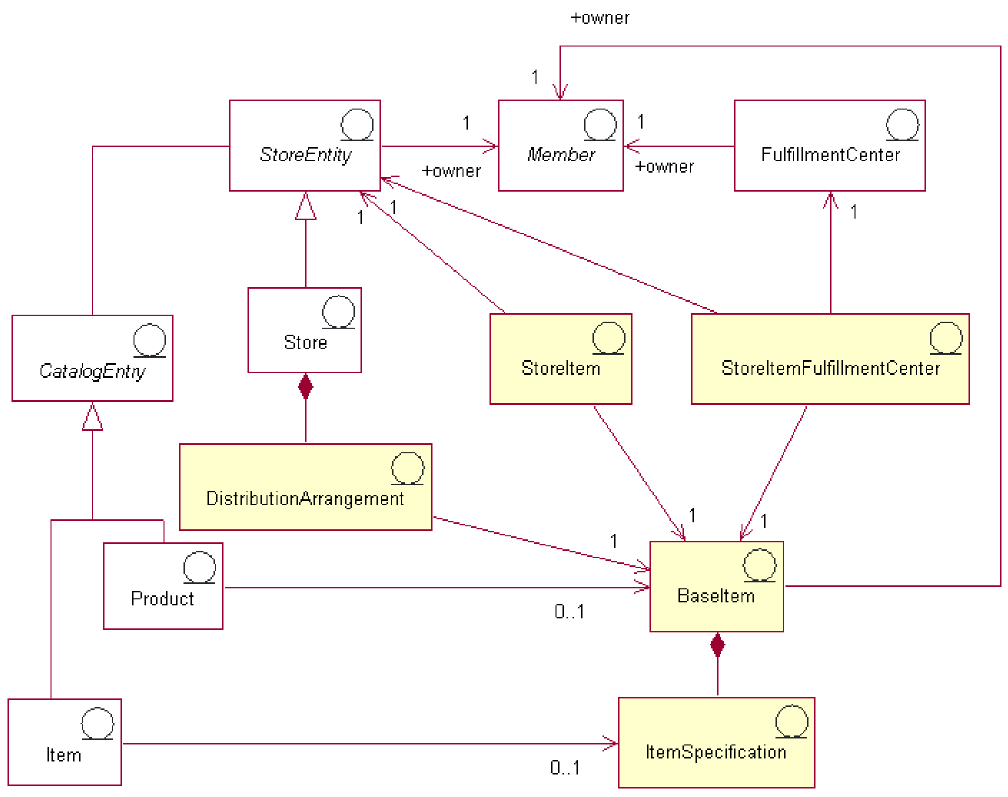
Ad hoc inventory receipts are created when inventory arrives at a fulfillment center without a corresponding expected inventory record. This could be due to an unexpected inventory arrival, or it could be the choice of the merchant or seller not to use expected inventory records to record inventory receipts.

Note: Products must exist in the WebSphere Commerce system in order to be received, whether the inventory receipt is expected or ad hoc.

Understanding inventory assets in WebSphere Commerce

In order to understand inventory assets, it is necessary to understand the relationships between inventory and the store. This can be explained by the use of an information model. The following sections describe the relationships and associations inventory has to a store and other assets. The diagrams below depict the relationships and associations for ATP inventory and non-ATP inventory. Non-ATP inventory is the way previous versions of the product handled inventory, and can still be used if a store chooses not to use the ATP features. Each diagram and its associations are described below. For more information on ATP, see the online help.

ATP inventory



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

Base item

The *base item* is the center of the inventory diagram and represents a general family of goods with a common name and description. Base items are used exclusively for fulfillment and are not particular to any store. Each catalog entry that represents a product in the catalog, has a corresponding base item for fulfillment purposes. Base items are defined in the BASEITEM table.

Item specification

An *item specification* is a base item with values defined for all its attributes. Each catalog entry that represents an item in the catalog has a corresponding item specification for fulfillment purposes.

Catalog entries

Products and items are *catalog entries*. Catalog entries are associated with store entities, meaning catalog entries, such as products and items, are found in stores.

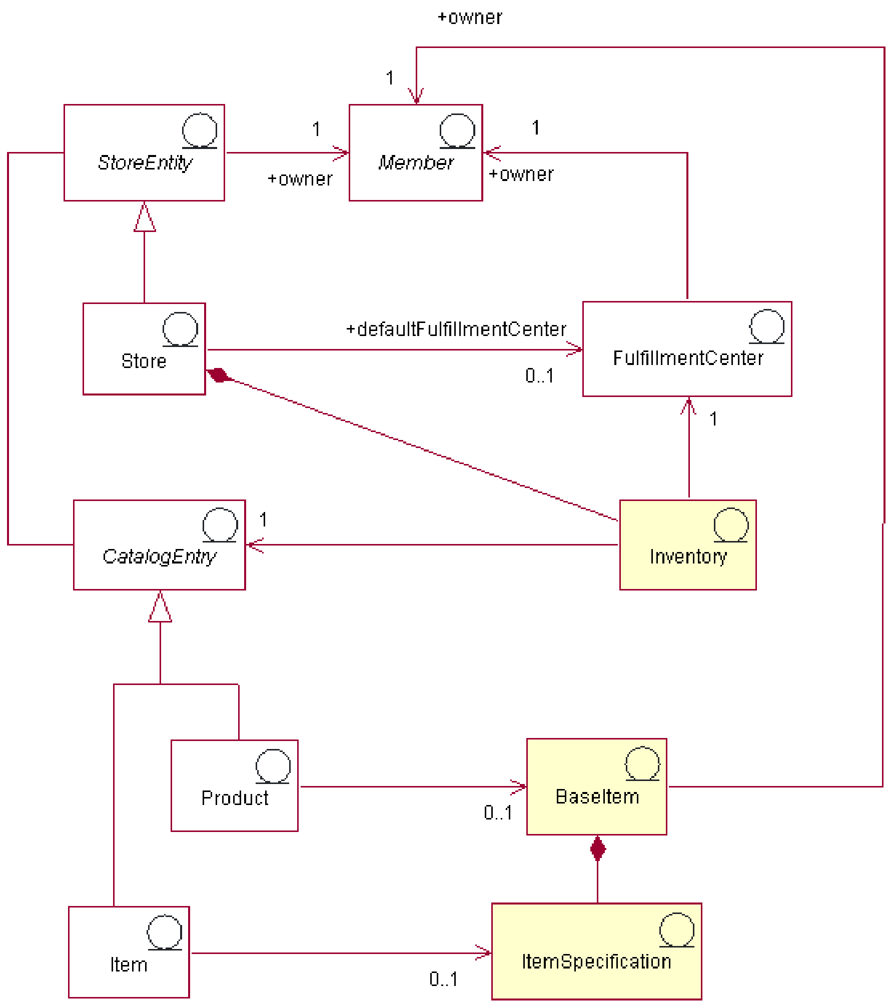
Distribution arrangement

A *distribution arrangement* is associated with a base item, enabling a store to sell its own inventory. Distribution arrangements are stored in the DISTARRANG table.

Store item

A *store item* represents attributes that affect the way a particular store or store group allocates inventory for the specified items of a particular base item, including whether to allow backorders and track inventory. The STORITMFFC table defines an estimate of the number of seconds it takes from the time an order item is released for fulfillment, until it is shipped to the customer. This table is only populated if a store wishes to define an override to the FFMCENTER default shipping offset for a store item.

Non-ATP inventory



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

The base item is also the center of the non-ATP inventory diagram. The relationship of the base item to products, items, and catalog entries, is the same as for the general inventory diagram. A base item is still owned by a member, and once defined by that member, can be sold in the store. In this case however, there is no distribution arrangement, store item association, or store item fulfillment center.

Fulfillment center

Inventory is associated with one *fulfillment center* and one store. A store can designate one default fulfillment center. Like base items, fulfillment centers are owned by members, and share that ownership with the store. For more information on fulfillment assets, seeChapter 11, “Fulfillment assets” on page 105.

For more detailed information on the structure of inventory assets in the WebSphere Commerce Server, see the inventory object and data models in the WebSphere Commerce online help.

Creating inventory assets in WebSphere Commerce

Since inventory is operational data, it changes daily, as your customers purchase products from your store, or return items to it. As a result your inventory levels go up and down as you sell products, and as your fulfillment centers receive new inventory from suppliers. The WebSphere Commerce Accelerator allows you to complete the following inventory related tasks:

- Record expected inventory
- Receive expected and ad hoc inventory from vendors
- Adjust inventory
- Maintain return records
- Maintain return reasons
- Receive returned inventory from customers
- Manage returned inventory disposition

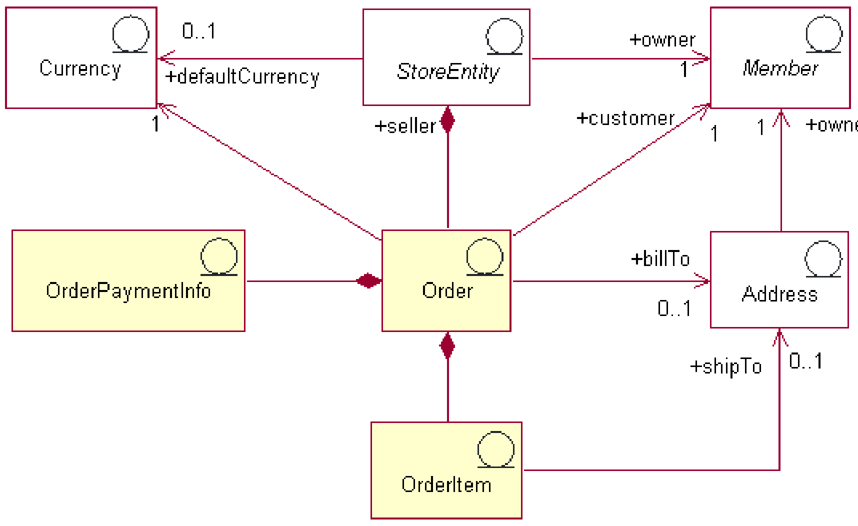
For more information on using the WebSphere Commerce Accelerator to manage inventory, see the WebSphere Commerce online help.

Chapter 22. Order assets

Order assets in the WebSphere Commerce system provide shopping cart, order management, and order processing functionality. Order processing capabilities include quick order or buy, scheduled orders, multiple pending orders, reorders, splitting orders and backorders. Related services, such as pricing, taxation, payment, inventory, and fulfillment, are also part of the order assets.

Understanding order assets in WebSphere Commerce

The following diagram illustrates the order assets in the WebSphere Commerce Server. Descriptions of each asset follows the diagram.



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25.

Orders and order items

In the WebSphere Commerce system, for a customer or shopper, an *order* is a list of selected products (for example, an order can contain two books and a CD) and each product on that list is an *order item* (for example, each book and CD is an order item of the same order). When a customer places an order with the store, the customer must provide a billing address to which the store sends the invoice. A single currency identifier is associated with each order. From a store perspective, an order is a list of order items. It is part of the store's data.

Currency

A store can display prices in one *currency*, or use multiple currencies. Each store must also define a *default currency*. You can also allow customers to select a *shopping currency*. If the shopping currency is the same as the default currency for the store, it is already supported in the STOREENT table. If the shopping currency is not the default currency for the store, then you must add the currency to the CURLIST table. Customers use the shopping currency to place orders at your store.

Payment information

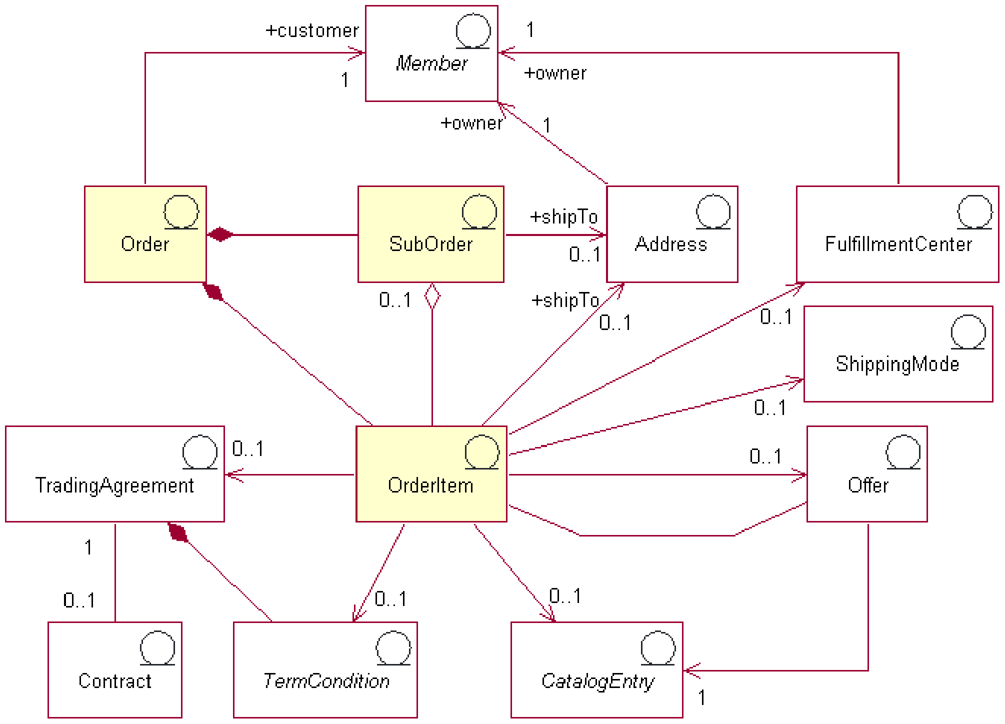
Once a customer has selected a preferred shopping currency, all payment will be processed in that currency. Depending on the store's payment support and policies, customers can pay for purchases using online payment (where a customer provides payment information over the Internet on the store's site) or offline payment (where the customer provides payment information without Internet channels, such as through phone or fax). Regardless of online or offline payment methods, customers must provide *payment information* when placing orders, including any of the following:

- Payment method: The customer's method of payment for the order. Depending on the payment cassettes configured in Payment Manager for the store, you can set up the store to accept offline payment, CyberCash for online payments, SET Secure Electronic Transaction™ and Merchant Initiated Authorization (MIA) for online payments that do not require customers to use an online wallet, or a custom payment method.
- For credit card payments, information about the card: The customer's credit card brand, number, and expiry date used to pay for the order. Credit card information is typically required if the store supports online payment.
- Purchase order number: The purchase order number, which the customer may have provided when ordering at the store. The purchase order number authenticates the customer as one that is authorized to order from the store, as stipulated in the terms within the contract between the store and the customer.

Order items

Order items are the individual products or items that make up an order. An order must have at least one order item. Each order item represents something that a customer has selected for purchase. In addition, each order item has a reference to a trading agreement (usually a contract), a shipping mode, a fulfillment center, and a price offer. Discounts, shipping charges, and total tax are stored with each order item.

The following diagram illustrates the WebSphere Commerce order item assets. Descriptions of each asset follows the diagram.



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

Suborders

Order items are grouped to form *suborders*. A suborder is the part of an order that is being shipped to a specific address. For example, a customer may indicate different shipping addresses for different products in the shopping cart. Each shipping address and the products associated with it constitute a suborder. Order items in a suborder have the same shipping address, and can be used to display sub-totals of their order item amounts.

The quantity attribute of the OrderItem object is a unitless number that can be multiplied by the nominal quantity attribute of the CatalogEntryShippingInformation object associated with the CatalogEntry object to arrive at the actual quantity represented by the OrderItem. The CatalogEntryShippingInformation object specifies the unit of measurement in which quantities are stated.

Although orders are usually associated with a single store, a special type of order that can be associated either with a store or a store group is the order profile. The order profile is represented in the object model as an Order with status of 'Q'. The order profile holds default information about a customer, such as payment information, shipping address, shipping mode, and billing address.

Other order item assets

An order item can be associated with zero or one of each of the following objects.

- A shipping address for the customer who placed the order containing the order item. A customer must specify a shipping address during the order process, so that the store’s fulfillment center can use this address to ship the order item appropriately.
- A fulfillment center for shipping and receiving order items required by customer orders, and for storing inventory for the order item.
- A shipping mode for the order item, which is a combination of a shipping carrier (a company that provides shipping services from a fulfillment center to a customer), and the shipping service offered by that carrier. For example, ABC Shipping Company, Overnight service and ABC Shipping Company, Express delivery are shipping modes.
- A price offer associated with the order item. By including different offers in different price lists (or ″trading position containers″), stores can present different prices for the same product or SKU to different customers. For example, a travel agency may offer plane tickets in four different price lists: adult pricing, seniors pricing, children’s pricing, and student pricing.
- A catalog entry for the order item; that is, each order item orders an item from a catalog.
- A trading agreement that defines the terms and conditions under which the item is ordered. This is normally a contract, but may be a Request for Quotation (RFQ), representing a negotiation, until the order has been submitted for processing.

For more detailed information on the structure of order assets in the WebSphere Commerce Server, see the order object and data models in the WebSphere Commerce online help.

Creating order assets in WebSphere Commerce

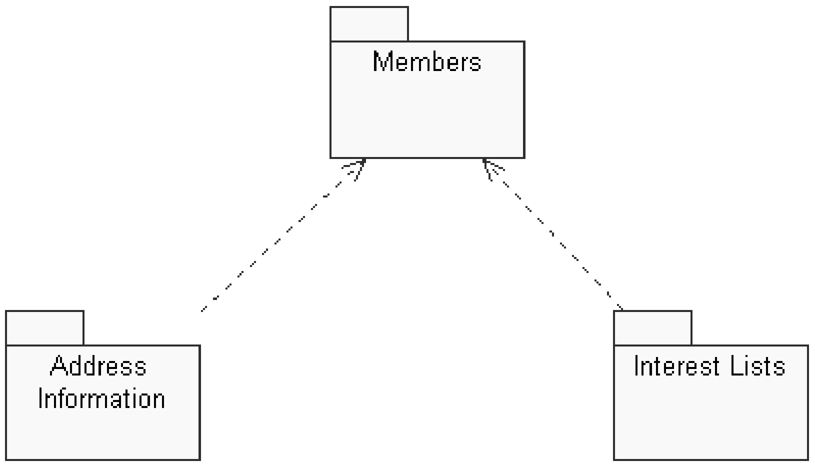
A customer can place orders from a store, or request that a Customer Service Representative for the store to help complete this task (using the WebSphere Commerce Accelerator). To create an order on behalf of a B2C customer, see the WebSphere Commerce online help topic ″Creating an order for a registered customer″ and ″Creating an order for a non-registered customer″. To create an order on behalf of a B2B customer, see the help topic ″Creating an order for a business user″.

Chapter 23. Customer and Seller assets

Although a store can include several players who participate in store activities, at a minimum, a store has a customer and a Seller.

Understanding customer assets in WebSphere Commerce

A *customer* is the person who shops at the store, creates an interest list if desired, places orders, and purchases from the store or the Seller. The following diagram illustrates the assets that a customer requires to place an order from a store.



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

As shown in the preceding diagram, the WebSphere Commerce system contains members. Each user member and organizational entity member can be assigned a role.

Note: In WebSphere Commerce, a *member* can be either an organizational entity, user, or member group. Refer to "Members" on page 180 for more details.

In this case, the user member is a customer. A customer must provide address information and can have an interest item list. The diagram illustrates the reciprocal relationship between a member (customer) and the customer assets associated with it: a customer must own and provide an address and can have an interest list to shop at a store; the address and interest list depend on the existence of a customer.

Address information

A customer must provide three types of address information, when purchasing from a store: the contact address, billing address, and shipping address. The following describes these address types; each address can be unique or the same:

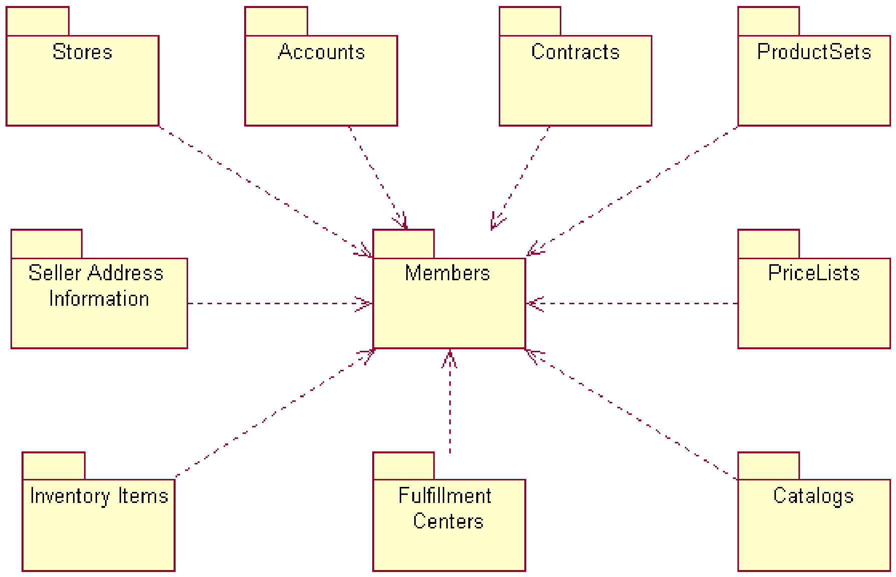
- A contact address is used to notify the customer for various purposes, such as regarding the status or changes to an order, and notices about upcoming store events (such as promotions or store maintenance). The customer's contact address includes the street name and number, city, state or province, ZIP or postal code, country or region, e-mail address, phone number, and fax number. Typically, the contact address is where the customer can be reached most easily, such as a work address.
- A billing address is used to send a bill or invoice for purchases. A billing address includes the street name and number, city, state or province, ZIP or postal code, and country or region, phone number, and e-mail address. The billing address may or may not be the same as the contact or shipping addresses.
- A shipping address is used for delivering purchased goods. A shipping address includes the street name and number, city, state or province, ZIP or postal code, and country or region, phone number, and e-mail address. The shipping address may or may not be the same as the contact or billing addresses.

Interest lists

Stores can support *interest lists*. That is, customers add products, that they may like to order in the future, to their interest lists. An interest list is not a shopping cart; a interest list can contain items from multiple stores, and does not contain prices, shipping addresses, shipping modes, inventory availability information, or calculated amounts such as discounts, shipping charges, and taxes.

Understanding Seller assets in WebSphere Commerce

A *Seller* supervises the overall store objectives and management, in addition to tracking the store sales. A Seller sells the goods and services to the customer. The Seller role is equivalent to a merchant and has access to all WebSphere Commerce Accelerator capabilities. The following diagram illustrates the assets that a Seller requires to maintain a store and to sell to customers.



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see "The store data information model" on page 25. For more information on the conventions used in this diagram, see Appendix A, "UML legend" on page 303.

As shown in the preceding diagram, the WebSphere Commerce system contains members. Each member is assigned a role, such as Customer Service Representative for the store, or Inventory Receiver at a warehouse. The Seller role can maintain the following assets in order to sell to customers:

- Stores
- Accounts (optional)
- Contracts (or at least the WebSphere Commerce default contract)
- Product sets
- Price lists
- Catalogs
- Fulfillment centers
- Inventory items

The preceding diagram illustrates the relationship between a member (Seller) and the Seller assets; that is, a Seller can have the assets listed above to maintain a store and the assets need to have a Seller got deployment.

Stores

A WebSphere Commerce online *store* is comprised of a set of HTML and JavaServer Pages files, as well as tax, shipping, payment, catalog and other database assets, which are contained in a store archive. A store also contains store data, which is the information populated into the WebSphere Commerce database to allow a store to function.

For more information about WebSphere Commerce stores, refer to Chapter 6, "Store assets" on page 41 and Part 4, "Developing your store data" on page 35.

Accounts

A store can set up business *accounts* for customers to allow them to purchase from the store. An account contains the following information:

- The account name, which is often the name of the organization with which the customer is associated. This organization has defined contracts with the store, stipulating terms for the customer to shop at the store. For example, the organization IBM may have contracts with the ABC Office Supplies Company.
- The representative name, which is the name of the representative organization within the Seller's organization that is responsible for the account.
- The number of contracts that belong to the account.

For more information about WebSphere Commerce accounts, refer to "Account (business account)" on page 94 and the WebSphere Commerce online help.

Contracts

Typically, in WebSphere Commerce, all customers must shop under a *contract*. Each account between the customer and the Seller must be associated with one or more contracts (or at least a *default contract* for non-registered customers or customers to shop at the store, or if you want customers to be able to purchase products not covered by other contracts). A contract allows the customer to purchase products from a store at a specified price for a specified period of time, under terms and conditions, and business policies, stipulated in the contract. The Seller deploys the contract so that customers can buy from the store.

For more information about WebSphere Commerce contracts and the default contract a Seller can use, refer to "Contract" on page 95.

Product sets

Product sets provide a mechanism for a Seller to categorize online catalogs into logical subsets so that a Seller can allow various customers to take advantage of different catalog views. Furthermore, a Seller can create a contract for a customer and stipulate that the customer can only purchase products under a predefined product set.

For more information about WebSphere Commerce product sets, refer to "Product sets" on page 56.

Price lists

A *price list* is associated with the price a Seller offers or presents to a customer. A Seller can list different prices for the same product to different customers. In WebSphere Commerce, a price offer is also known as a *trading position* and represents the price of a catalog entry and criteria that the customer must satisfy in order to qualify for that price.

In WebSphere Commerce, an Offer object is part of a TradingPositionContainer, which is owned by a member. A TradingPositionContainer contains TradingPositions, and can be made available to all customers, or to only customers in certain groups through the trading agreements or contracts. Sometimes a TradingPositionContainer is referred to as a price list. There are two kinds of price lists: a standard price list which contains the base prices for the products in the store catalog or a custom price list which specifies the list of products and their customized prices.

For more information about WebSphere Commerce price lists, refer to Chapter 9, "Pricing assets" on page 85.

Catalogs

A WebSphere Commerce store uses at least one online *catalog* to showcase the goods and services that the Seller offers for sale. Typically, an online catalog contains prices, images, and descriptions of the items for sale. An online catalog may also present merchandise into distinct categories to facilitate navigation.

Each store in the WebSphere Commerce system must have a *master catalog*, which is used for catalog management. The master catalog is the central location to manage a Seller's merchandise; it is the single catalog containing all products, items, relationships, and standard prices for everything that is for sale in the store. If a Seller has more than one store, the master catalog can be shared between these stores.

For more information about WebSphere Commerce product sets, refer to Chapter 8, "Catalog assets" on page 53.

Fulfillment centers

Fulfillment centers are used by stores as both inventory warehouses and shipping and receiving centers. A Seller may have one or many fulfillment centers.

From a WebSphere Commerce server perspective, a FulfillmentCenter object is separate from the Store object. It manages product inventory and shipping. To ship an order, the fulfillment center relies on a ShippingMode object that is specified by the customer. The ShippingMode object indicates the shipping carrier and method of shipping for fulfilling orders. In a fulfillment center, the ShippingArrangement object indicates that a Store object has arranged with a FulfillmentCenter object to ship products using a certain ShippingMode.

For more information about WebSphere Commerce fulfillment centers, refer to Chapter 11, “Fulfillment assets” on page 105 and Chapter 18, “Shipping assets” on page 131.

Inventory items

Inventory items include anything that can be physically accounted for in a Seller’s fulfillment center. The WebSphere Commerce system defines specific types of inventory that can be fulfilled, such as items, products, SKUs, bundles, and packages; but these are all considered inventory. Products are configured for fulfillment using the Product Management tools on WebSphere Commerce Accelerator.

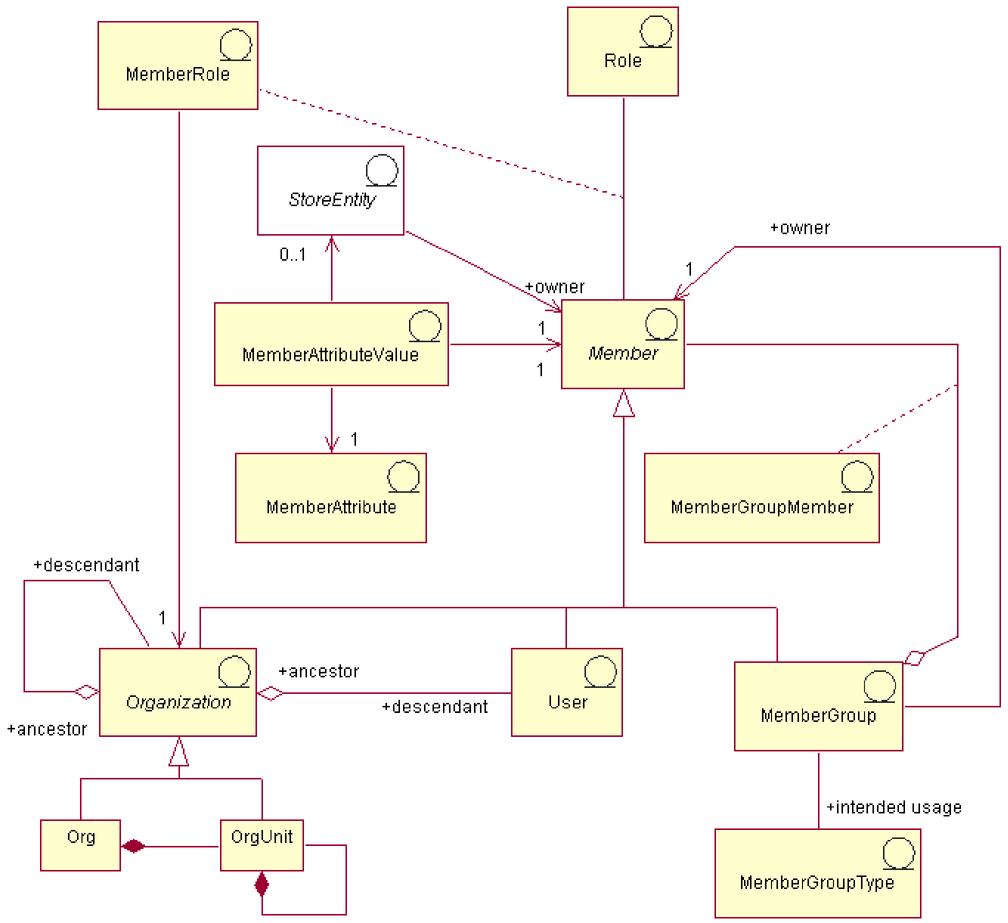
For more information about WebSphere Commerce inventory items, refer to the WebSphere Commerce online help and Chapter 21, “Inventory assets” on page 167.

Understanding member assets in WebSphere Commerce

WebSphere Commerce member assets include data for participants of the WebSphere Commerce system. A member can be a user, a group of users, or an organizational entity. An administrator, such as a Site Administrator, assigns roles to users and organizational entity members. Once a member is assigned a role, the access control component authorizes the member to participate in activities. For example, an organization can be a buyer or a Seller, or both. A user can also be assigned multiple roles. An administrator can create member groups, which are groups of users categorized for various business reasons. Use the WebSphere Commerce Administration Console to create and work with organizations, users, roles, and member groups.

Business logic for the member assets provides member registration and profile management services. Other services which are closely related to the member assets include access control, authentication, and session management. For more details about these topics, refer to the WebSphere Commerce online help.

The following diagram illustrates the WebSphere Commerce member assets. Descriptions of each asset follow the diagram.



This diagram, and all others in the store data section are part of the WebSphere Commerce Server information model. For more information on the information model, see “The store data information model” on page 25. For more information on the conventions used in this diagram, see Appendix A, “UML legend” on page 303.

Members

A *member* in WebSphere Commerce can be any of the following:

- An *organizational entity*. This can be an organization, such as ″IBM″ or an organizational unit within a large organization, such as the ″Electronic Commerce Division″ within IBM.
- A *user* (either registered or non-registered). A registered user has a unique identifier, and a password, and is required to provide profile data for registration purposes. Registered users can be classified according to their profile type: type ’B’ denotes a business user (or a B2B customer) and type of ’C’ denotes a retail user (or a B2C customer). For more information about registered and non-registered users, refer to ″Members″ in the WebSphere Commerce online help.

- A *member group*. This is a group of users categorized for various business reasons. The groupings can be used for access control purposes, for approval purposes, as well as for marketing purposes (such as calculating discounts, prices, and displaying products).

Each *store entity* (that is, a store or store group) is owned by a member.

Member attributes

A WebSphere Commerce member has a set of *attributes* and each attribute has a *value* associated with it. A basic user profile for a member incorporates registration information, demographics, address information, purchase history, and other miscellaneous attributes.

A business user profile contains the same information as a basic user profile, as well as employment information, such as an employee number or a job title, or a job description. During registration, business users should identify their business organization to which they belong. Profiles for organizational entities include this additional information, such as organization name and business category.

Access control rules enforce user authority for performing profile management. Member profiles can contain a variety of personal and business-related attributes (such as roles, payment information, addresses, preferred languages and currencies, and pervasive computing devices). Attributes can be store-sensitive. These attributes are not supported for member groups.

Roles

Each user member can perform one or more roles in an organization. A Site Administrator assigns a role or roles to each user member. For example, as a member of the IBM organization, John Smith's role as a Customer Service Representative means that John performs tasks on behalf of IBM customers and assists them with inquiries or concerns regarding their registration information, orders, or returns. John may also have the role of a Customer Service Supervisor, who has all the responsibilities of the tasks described above, as well as approval and supervisor authority over other Customer Service Representatives.

The WebSphere Commerce system provides the following set of default role types:
- Site and content development roles
- Technical operations roles
- Marketing management roles
- Product management roles
- Business relationship management roles
- Logistics and operations management roles
- Organizational management roles

For details about each of these roles, refer to the WebSphere Commerce online help topic Roles. A Site Administrator can assign these roles, as well as any new roles created by the Site Administrator, by organizational entity; that is, users who belong to an organizational entity can assume roles assigned to that organizational entity.

When a user is assigned a role, the role is scoped to organizational entity. When a user is assigned a role, the user does not necessarily perform that role for the organizational entity to which the user belongs; that is, when an administrator performs the assignment, the administrator can select the organizational entity for which the user performs that role. If the administrator selects the Root Organization, the user plays that role for all the organizational entities.

For more detailed information on the structure of member assets in WebSphere Commerce, see the member object and data models in the WebSphere Commerce online help.

Creating member assets in WebSphere Commerce

In order to create a Seller (an organization that acts as the store owner) and to maintain information about the Seller, use the Administration Console. For more information, see the WebSphere Commerce online help topic ″Creating an organization″.

A customer is not created by the Store Developer; when a customer registers with a store, registration information is collected and maintained by the WebSphere Commerce system.

Part 5. Adding access control to your store

Chapter 24. Access control in your store

WebSphere Commerce allows you to determine, through access control, which tasks a particular user, be they customers or administrators, can perform. This chapter focuses on how you can add access control to your store, thus restricting which pages your customers can see, and which tasks in the store they can perform.

For more information on the access control model in WebSphere Commerce, as well as applying access control at the site level, refer to the *IBM WebSphere Commerce Access Control Guide*. For more information on implementing access control in customized code, refer to the *IBM WebSphere Commerce Programmer's Guide*. These guides are available from the following URLs:

▶Business `http://www.ibm.com/software/webservers/commerce/wc_be/lit-tech-general.html`

▶Professional `http://www.ibm.com/software/webservers/commerce/wc_pe/lit-tech-general.html`

Understanding access control in WebSphere Commerce

The access control model for WebSphere Commerce is covered in detail in the *IBM WebSphere Commerce Access Control Guide*. However, in order to understand how access control affects store development, a brief summary is provided here.

Access control in WebSphere Commerce is composed of the following elements: users, actions, resources, and relationships.

- Users are the people that use the system. For access control purposes, users must be grouped into relevant access groups. For example, in your store you might group users into following access groups: registered customers, guest customers, or administrative groups like customer service representatives.
- Actions are the activities that users can perform on the resource. For access control purposes, actions must also be grouped into relevant action groups. For example, a common action used in a store is a view. A view is invoked to display a store page to customers. The views used in your store must be assigned to an action group.
- Resources are the entities that are protected. For example, if the action is a view, the resource for the action is the class `com.ibm.commerce.command.ViewCommand`. For access control purposes, resources are grouped into resource groups.
- Relationships are the relationship between the user and the resource. Access control policies may require that a relationship between the user and the resource be satisfied.

Access control policies

Access control policies authorize access groups to perform particular actions on the resources of WebSphere Commerce, as long as the users in the access group satisfy a particular relationship with respect to the resource.

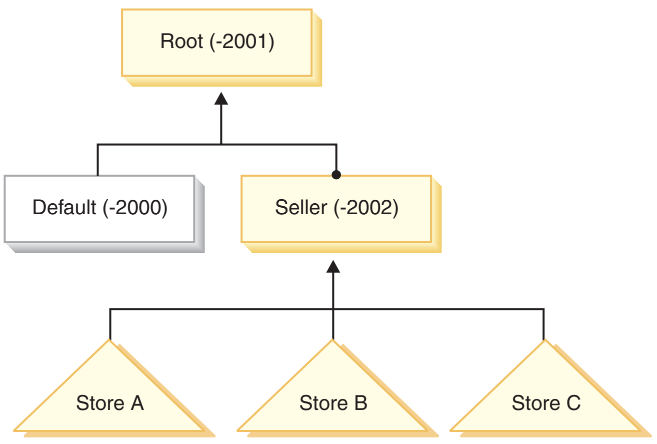
WebSphere Commerce provides over two hundred default access control policies that are loaded during instance creation. These policies cover a wide range of common business activities, including order creation and processing, and trading, such as ▶Business request for quotes and ▶Business contracts. The default policies are documented in the *IBM WebSphere Commerce Access Control Guide*

Each access control policy is owned by an organizational entity. An access control policy can only be applied to resources that are owned by the access control policy owner.

Note: In terms of access control, ownership of resources has a special meaning. All resources must implement the `com.ibm.commerce.security.Protectable` interface. One of the methods on this interface is `getOwner()`, which returns the member ID of the owner of the resource. For example, the Order entity bean is a resource that is protected by having its remote interface extend the Protectable interface. The Order's implementation of `getOwner()` is such that a specific Order resource returns the owner of the store where the order was placed. For policies where the resource is a command, for example, `com.ibm.commerce.command.ViewCommand`, the default implementation of `getOwner()` is to return the owner of the store that is currently in the command context. If there is no store in the command context, then Root Organization is used as the owner. For more information, see the *IBM WebSphere Commerce Programmer's Guide*.

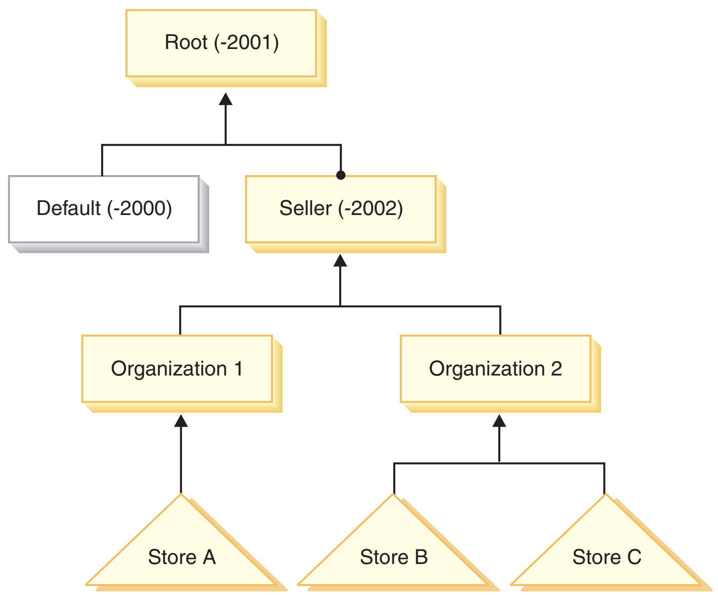
Consider the following diagrams:

▶Professional



In WebSphere Commerce Professional Edition the Root Organization is the highest level organization. It owns all other organizational entities. As a result, access control policies that are owned by the Root Organization apply to all resources in the site. If an access control policy is owned by the Seller Organization, then the access control policy only applies to the resources owned by the seller.

Business



In WebSphere Commerce Business Edition the Root Organization is the highest level organization. It owns all other organizational entities. As a result, access control policies that are owned by the Root Organization apply to all resources in the site. If an access control policy is owned by the Seller Organization, then the access control policy only applies to the resources owned by the seller, that is, it would apply to the resources owned by Organizations 1 and 2, including Stores A, B, and C. If the access control policy was owned by Organization 1, then it would only apply to the resources including Store A. If the access control policy was owned by Organization 2, it would apply to the resources in Stores B and C.

Business Since access control policies are owned by organizational entities, if you are creating multiple stores in your site, and want to apply different access control policies to individual stores, you must create separate organizations to own each store.

Note: Most of the default access control policies provided with WebSphere Commerce are owned by the Root Organization, and apply to all resources in the site. Access control policies owned by the Root Organization are referred to as site-level policies. Policies owned by other organizational entities are known as organization-level policies.

Access control in stores

All stores created in WebSphere Commerce are subject to the default access control policies owned by the Root Organization. The resources in stores are also subject to any access control policies owned by the organization that owns the store, and any access control policies owned by that organization's ancestors.

By default, the Root Organization owns most of the access control policies. However, if you create your store based on one of the sample stores provided with WebSphere Commerce, you will create new access control policies that are owned by the organization that owns the store. For more information about the access control policies in the sample stores, see "Access control in the samples stores" on page 188.

When you are creating your own store, regardless of whether it is based on a sample, you may want to create new access control policies or modify existing policies, which will only apply to stores owned by that organization. For example, if you create new views to display your store pages, you must assign access control policies to these views.

Access control data at the organization level is defined in high level access control policy files. These files define the possible actions, action groups, resources, resource groups, and relationships that can be used by any policy. They also define policies specific to a particular organization. The sample stores provided with WebSphere Commerce contain these high level access control policy files. The following section illustrates how the samples stores use these access control policy files to define organization information.

Access control in the samples stores

All of the sample stores contain the high level access control policy files, which define access control policies created specifically for the stores. These policies are owned by the organization that owns the store. The access control policies defined for the NewFashion and Business ToolTech sample stores are discussed below.

The access control policies created for Business ToolTech are as follows:
- AllUsersForToolTechExecuteToolTechAllUsersViews
- RegisteredApprovedUsersForToolTechExecuteToolTech RegisteredApprovedUsersViews

The access control policy created for NewFashion is as follows:
- AllUsersExecuteNewFashionAllUsersViews

These access control policies determine which users will see which views in the sample stores, or in stores based on the samples.

The access control information for these policies is defined in two high level access control policy files, which define the possible actions, action groups, resources, and policy definitions used in the sample stores: *samplestorename*`AccessPolicies.xml` and *samplestorename*`AccessPolicies_`*locale*`.xml`. These files are located in the following directory:

- NT `drive:\WebSphere\CommerceServer\samples\stores\`*`samplestorename`*
- 2000 `drive:\ProgramFiles\WebSphere\CommerceServer\samples\stores\`*`samplestorename`*
- AIX `/usr/WebSphere/CommerceServer/samples/stores/`*`samplestorename`*
- Solaris `/opt/WebSphere/CommerceServer/samples/stores/`*`samplestorename`*
- Linux `/opt/WebSphere/CommerceServer/samples/stores/`*`samplestorename`*
- 400 `/QIBM/ProdData/WebCommerce/samples/stores/`*`samplestorename`*

  where *samplestorename* is the name of the sample store archive on which you based your store, for example NewFashion.

Before packaging the sample stores as store archive, these two high level access control policy files are transformed, which results in two XML files, suitable for use with the Loader package. These transformed XML files are then packaged in the store archive and published with the rest of the store archive.

Understanding the sample store access control policy files: To understand how access control is added at the store level, familiarize yourself with the high level sample store access control policy files. The following examples are taken from the Business `ToolTechAccessPolicies.xml` file.

Defining actions: The first section of the Business `ToolTechAccessPolicies.xml` file defines the new actions in the store, which are not covered by existing access control policies. In this case, the actions are all views used in the store. In order to display a page in your store using a view that can be called directly from a URL, or that can be launched by a redirect from another command (in contrast to being launched by forwarding to the view) you must define it as an action. Consider the following example:

```
<!-- [Start of Action definitions] -->
<!-- [this is the dictionary of possible actions -->
<Action Name="GenericApplicationError"
   CommandName="GenericApplicationError">
 </Action>

 <Action Name="GenericSystemError"
   CommandName="GenericSystemError">
 </Action>

 <Action Name="OrderOptionsView"
   CommandName="OrderOptionsView">
 </Action>

  <!--[End of Action definitions] -->
```

where

- `Action Name` is the label used to reference this action in the XML file. In these examples, the label is the same as the view name.
- `CommandName` is the name of the view that is stored in the VIEWNAME column of the VIEWREG table. The CommandName will be stored in the Action column of the ACACTION table.

Note: If an action is already defined in the default policies or any other policy in WebSphere Commerce, you do not have to redefine it for each policy that uses the action.

Defining resource categories: The second section in the file defines the resource categories. A resource category refers to a class of resources. The resource categories identified for ToolTech are the classes `com.ibm.commerce.command.ViewCommand` and `com.ibm.commerce.tools.command.ToolsForwardViewCommand`.

```
<!-- [Start of Resource Category definitions] -->
 <!-- the dictionary of Protectable resources -->

  <ResourceCategory Name="com.ibm.commerce.command.ViewCommandResourceCategory"
   ResourceBeanClass="com.ibm.commerce.command.ViewCommand">

 </ResourceCategory>

 <ResourceCategory Name="com.ibm.commerce.tools.command.
ToolsForwardViewCommandResourceCategory"
   ResourceBeanClass="com.ibm.commerce.tools.command.ToolsForwardViewCommand">

 </ResourceCategory>
 <!-- [End of Resource Category definitions] -->
```

where

- `ResourceCategory Name` label used to reference this resource category in the XML file.
- `ResourceBeanClass` is the name of class.

Note: If a resource category is already defined in the default policies or any other policy in WebSphere Commerce, you do not have to redefine it for each policy that uses the resource category. The resource category defined in the example above is already defined in the default policies, but is defined again here for illustration purposes.

Defining action groups: The third section defines the action group. The action group is a grouping of the actions defined in the first section of the file. In the ToolTech example, all new user views are grouped into the group ToolTechAllUserViews, which will be used in a policy that will allow all users to access those views, or the ToolTechRegisteredApprovedUserviews, which will be used in a policy that will allow only registered users to access those views.

Note: You can also add actions that are defined elsewhere in WebSphere Commerce to your action groups. If defined elsewhere in WebSphere Commerce, these actions do not have to be defined in the Action list discussed in "Defining actions" on page 189.

```
<!-- [Start of Action Group definitions] -->
 <!-- Dictionary of grouped actions usable in policies -->
 <!-- cross-component view-related action groups -->
  <ActionGroup Name="ToolTechAllUsersViews"
    OwnerID="RootOrganization">

  <ActionGroupAction Name="UserRegistrationForm"/>
  <ActionGroupAction Name="UserRegistrationErrorView"/>
  <ActionGroupAction Name="GenericApplicationError"/>
  <ActionGroupAction Name="GenericSystemError"/>
  <ActionGroupAction Name="LogonForm"/>
    </ActionGroup>

  <!-- [End of Action Group definitions] -->
```

where

- `ActionGroup Name` is the name of the action group.
- `OwnerID` is the owner of the action group. The Root Organization must be the owner of the action group.
- `ActionGroupAction Name` is the name of an action that belongs to this group. The ActionGroupAction Name must match the name defined in the Action Name element in "Defining actions" on page 189.

Defining resource groups: The next section defines the resource groups. A resource group identifies a set of related resources.

```
<!-- [Start of Resource Group definitions] -->
 <!-- Dictionary of grouped resources usable in policies -->

 <!-- Grouped resources permitting view execution,
  by any command extending com.ibm.commerce.command.ViewCommand
 (with or without the Tools Framework) -->
 <ResourceGroup Name="ViewCommandResourceGroup"  OwnerID="RootOrganization">

  <ResourceGroupResource Name="com.ibm.commerce.command.ViewCommandResourceCategory"/>
 </ResourceGroup>

  </ResourceGroup> !-- [End of Resource Group definitions] -->
```

where

- ResourceGroup Name is the name of the resource group.
- OwnerID is the owner of the resource group. The Root Organization must own the resource group.
- ResourceGroupResource Name is the name of a resource category included in the group.

Defining policies: The final section defines the new policies used in the store.

```
!-- [Start of Policy definitions] -->

 !-- AllUsers for ToolTech can execute ToolTechAllUsersViews -->


<Policy Name="AllUsersForToolTechExecuteToolTechAllUsersViews"
   OwnerID="MEMBER_ID"
   UserGroup="AllUsers"
   UserGroupOwner="RootOrganization"
   ActionGroupName="ToolTechAllUsersViews"
   ResourceGroupName="ViewCommandResourceGroup">
    </Policy>
 !-- RegisteredApprovedUsers for ToolTech can execute
ToolTechRegisteredApprovedUsersViews -->

  <Policy Name="RegisteredApprovedUsersFor
ToolTechExecuteToolTechRegisteredApprovedUsersViews"
   OwnerID="MEMBER_ID"
   UserGroup="RegisteredApprovedUsers"
   UserGroupOwner="RootOrganization"
   ActionGroupName="ToolTechRegisteredApprovedUsersViews"
   ResourceGroupName="ViewCommandResourceGroup">
    </Policy>

 !-- [End of of Policy definitions] -->
```

where

- `Policy Name` is the name of the policy being defined.
- `OwnerId` is the owner of the policy. In this case the owner of the policy is the organization that owns the store.
- `UserGroup` is the group of users (the access group) to whom the policy applies.
- `UserGroupOwner` is the owner of the access group. In this example, the owner of the access group is different than the policy owner. If the the policy owner and the UserGroupOwner are the same, this element can be omitted.
- `ActionGroupName` is the group of actions to which the policy applies.
- `ResourceGroupName` is the group of resources to which the policy applies.

Adding access control to your store

From a store development perspective the most common types of access control needed are for the new views and commands you create for your store. However, you may want to add other types of access control to your store. For more information on access control for views, commands and other features, see the *IBM WebSphere Commerce Access Control Guide*. Before continuing with the next steps outlined in the this guide, ensure you review the *IBM WebSphere Commerce Access Control Guide*.

If you are adding new access control features to a store based on a sample store, edit the existing high level access control policy XML file. For detailed instructions, see “Editing access control files in the store archive” on page 195. If you are adding access control to a store not based on a sample store, you will need to create a new high level access control policy XML file, and then transform it. For detailed instructions, see "Creating access control in your store".

Creating access control in your store

The access control assets are different than the other assets in the store, in that for access control you create two high level access control XML files and then transform them. The resulting XML files can then be loaded in the the database using the Loader package or used in the store archive.

To create access control assets, do the following:

1. Review the high level XML files used to create store assets for the sample stores: *samplestorename*`AccessPolicies.xml` and *samplestorename*`AccessPolicies_`*locale*`.xml`. These files are located in the following directory:
   - NT `drive:\WebSphere\CommerceServer\samples\stores\`*samplestorename*
   - 2000 `drive:\ProgramFiles\WebSphere\CommerceServer\samples\stores\`*samplestorename*
   - AIX `/usr/WebSphere/CommerceServer/samples/stores/`*samplestorename*
   - Solaris `/opt/WebSphere/CommerceServer/samples/stores/`*samplestorename*
   - Linux `/opt/WebSphere/CommerceServer/samples/stores/`*samplestorename*
   - 400 `/QIBM/ProdData/WebCommerce/samples/stores/`*samplestorename*

   where *samplestorename* is the name of the sample store archive on which you based your store, for example NewFashion.
2. Review the information in Appendix B, "Creating your data" on page 305.
3. Create a *storename*`AccessPolicies.xml` file, either by copying one of the *samplestorename*`AccessPolicies.xml` files or by creating a new one. For more information, see the DTD file that corresponds to *samplestorename*`AccessPolicies.xml`. The DTD files are located in the following directory:
   - NT `drive:\WebSphere\CommerceServer\xml\policies\dtd`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\xml\policies\dtd`
   - AIX `/usr/WebSphere/CommerceServer/xml/policies/dtd`
   - Solaris `/opt/WebSphere/CommerceServer/xml/policies/dtd`
   - Linux `/opt/WebSphere/CommerceServer/xml/policies/dtd`
   - 400 `/QIBM/ProdData/WebCommerce/xml/policies/dtd`
4. Create a separate high level XML file for each locale your store supports. The locale-specific file should specify all description and display name information, so it can be easily translated. Create this file, *storename*`AccessPolicies_`*locale*`.xml` for each language in your store, either by copying one of the *samplestorename*`AccessPolicies_locale.xml` files or by creating new ones.

   Note: You will need to create a locale-specific file even if your store only supports one language.

5. Add the appropriate access control information to the file. For more information see "Access control in the samples stores" on page 188 and the *IBM WebSphere Commerce Access Control Guide*.
6. Copy *storename*`AccessPolicies.xml` and *storename*`AccessPolicies_locale.xml` to the following directory:
   - NT `drive:\WebSphere\CommerceServer\xml\policies\xml`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\xml\policies\xml`
   - AIX `/usr/WebSphere/CommerceServer/xml/policies/xml`
   - Solaris `/opt/WebSphere/CommerceServer/xml/policies/xml`
   - Linux `/opt/WebSphere/CommerceServer/xml/policies/xml`
   - 400 Copy to any user data directory. Specify the full path to the DTD in the existing XML files. The access control DTD files are located in the following directory: `/QIBM/ProdData/WebCommerce/xml/policies/dtd`
7. Run the `xmltransform` command to transform *storename*`AccessPolicies.xml`.
   a. At a command prompt, change the directory to the following:
      - NT `drive:\WebSphere\CommerceServer\bin`
      - 2000 `drive:\ProgramFiles\WebSphere\CommerceServer\bin`
      - AIX `/usr/WebSphere/CommerceServer\bin`
      - Solaris `/opt/WebSphere/CommerceServer/bin`
      - Linux `/opt/WebSphere/CommerceServer/bin`

   b. Then type: `xmltransform -infile ..\xml\policies\xml\samplestorenameAccessPolicies.xml -transform ..\xml\policies\xsl\accesscontrol.xsl -outfile ..\xml\policies\xml\samplestorenameAccessPoliciesOut.xml`

      400 `TRNWCSXML INFILE (input file) TRANSFORM('/QIBM/ProdData/WebCommerce/xml/policies/xsl /accesscontrol.xsl') INSTROOT(instance_root) OUTFILE(output_file)`

   c. Check the following log file to ensure that the transform has completed successfully:
      - NT `drive:\WebSphere\CommerceServer\bin\xmltransform.db2.log`
      - 2000 `drive:\Program Files\WebSphere\CommerceServer\bin\xmltransform.db2.log`
      - AIX `/usr/WebSphere/CommerceServer/bin/xmltransform.db2.log`
      - Solaris `/opt/WebSphere/CommerceServer/bin/xmltransform.db2.log`
      - Linux `/opt/WebSphere/CommerceServer/bin/xmltransform.db2.log`
      - 400 `/QIBM/UserData/WebCommerce/instances/instancename /logs/TRNWCSXML.tx`

      If the transform was successful, the following message displays: `"<DATE> <TIME> java.lang.Class main XMLTransformer Transform Successful"`
8. Run the `xmltransform` command to transform *storename*`AccessPolicies_locale.xml`.
   a. At a command prompt, change the directory to the following:

- NT `drive:\WebSphere\CommerceServer\bin`
- 2000 `drive:\ProgramFiles\WebSphere\CommerceServer\bin`
- AIX `/usr/WebSphere/CommerceServer\bin`
- Solaris `/opt/WebSphere/CommerceServer/bin`
- Linux `/opt/WebSphere/CommerceServer/bin`

b. Then type: `xmltransform -infile ..\xml\policies\xml\`*`storename`*`AccessPolicies_locale.xml -transform ..\xml\policies\xsl\accesscontrolnls.xsl -outfile ..\xml\policies\xml\`*`storename`*`AccessPoliciesOut_locale.xml`

c. 400 `TRNWCSXML INFILE(input file) TRANSFORM('/QIBM/ProdData/WebCommerce/xml/policies/xsl/accesscontrolnls.xsl') INSTROOT(instance_root) OUTFILE(output_file)`

9. Make the following changes to the resulting XML files:

a. In *storename*`AccessPolicesOut.xml`, replace the opening and closing tags with the following:

```
<?xml version="1.0"?>
<!DOCTYPE accesscontrol-asset SYSTEM "accesscontrol.dtd">
<accesscontrol-asset>
</accesscontrol-asset>
```

b. In *storename*`AccessPolicesOut_locale.xml`, replace the opening and closing tags with the following:

```
<?xml version="1.0" encoding="correct language code for the file"?>
<!DOCTYPE accesscontrol-asset SYSTEM "../accesscontrol.dtd">
<accesscontrol-asset>
</accesscontrol-asset>
```

c. In *storename*`AccessPolicesOut_locale.xml`, replace the @locale with the &locale; for example change LANGUAGE_ID="@en_US" to LANGUAGE_ID="&en_US;"

d. In *storename*`AccessPolicesOut_locale.xml`, locate the reference to the "acpoldesc" table. Remove the @ at the end of the ACPOLICY_ID value. For example, change `"@AllUsersExecuteInFashionAllUsersViews@"` to `"@AllUsersExecuteInFashionAllUsersViews"`.

e. In *storename*`AccessPolicesOut.xml`, replace MEMBER_ID="MEMBER_ID" with MEMBER_ID="&MEMBER_ID;"

f. In *storename*`AccessPolicesOut.xml` locate the reference to the "acpolicy" table. Remove the "@MEMBER_ID" at the end of the ACPOLICY_ID value. For example, change `"@AllUsersExecuteInFashionAllUsersViews@MEMBER_ID"` to `"@AllUsersExecuteInFashionAllUsersViews"`

10. At this point you have two options:
- Load the access control data using the Loader package. For more information, see Part 7, "Publishing your store" on page 207.
- Add the access control data to the store archive. For more information on creating a store archive, see Part 6, "Packaging your store" on page 199.

For more information about the use of @ and & see Appendix B, "Creating your data" on page 305.

Editing access control files in the store archive

WebSphere Commerce provides two pre-transformed access control XML files, one that applies to all languages (`*samplestorename*AccessPolicies.xml`), and one that contains locale specific information (`*samplestorename*AccessPolicies_locale.xml`), for each sample store. You must transform each of these files, which results in two XML files, one that applies to all languages (`*samplestorename*AccessPoliciesOut.xml`), and one that contains locale specific information (`*samplestorename*AccessPoliciesOut_locale.xml`)

To edit the access control database asset in the store archive, do the following:

1. Locate the pre-transformed access control XML files for the sample store on which you based your store. These files are called *samplestorename*`AccessPolicies.xml` and *samplestorename*`AccessPolicies_locale.xml`. The files are located by default in the following directory:
   - NT `drive:\WebSphere\CommerceServer\samples\stores\`*samplestorename*
   - 2000 `drive:\ProgramFiles\WebSphere\CommerceServer\samples\stores\`*samplestorename*
   - AIX `/usr/WebSphere/CommerceServer/samples/stores/`*samplestorename*
   - Solaris `/opt/WebSphere/CommerceServer/samples/stores/`*samplestorename*
   - Linux `/opt/WebSphere/CommerceServer/samples/stores/`*samplestorename*
   - 400 `/QIBM/ProdData/WebCommerce/samples/stores/`*samplestorename*

   where *samplestorename* is the name of the sample store archive on which you based your store, for example NewFashion.

   Note: Changing the corresponding DTD files may result in unusable policies.
2. Make the necessary changes to the file. For more information on the existing file, see "Understanding the sample store access control policy files" on page 189. For more information on methods of access control available in the WebSphere Commerce, see the *IBM WebSphere Commerce Access Control Guide.*
3. Copy *samplestorename*`AccessPolicies.xml` and *samplestorename*`AccessPolicies_locale.xml` to the following directory:
   - NT `drive:\WebSphere\CommerceServer\xml\policies\xml`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\xml\policies\xml`
   - AIX `/usr/WebSphere/CommerceServer/xml/policies/xml`
   - Solaris `/opt/WebSphere/CommerceServer/xml/policies/xml`
   - Linux `/opt/WebSphere/CommerceServer/xml/policies/xml`
   - 400 Copy to any user data directory. Specify the full path to the DTD in the existing XML files. The access control DTD files are located in the following directory: `/QIBM/ProdData/WebCommerce/xml/policies/dtd`
4. Run the xmltransform command to transform *samplestorename*`AccessPolicies.xml`.
   a. At a command prompt, change directories to the following:

- NT `drive:\WebSphere\CommerceServer\bin`
- 2000 `drive:\ProgramFiles\WebSphere\CommerceServer\bin`
- AIX `/usr/WebSphere/CommerceServer\bin`
- Solaris `/opt/WebSphere/CommerceServer/bin`
- Linux `/opt/WebSphere/CommerceServer/bin`

b. Then type:`xmltransform -infile ..\xml\policies\xml\samplestorenameAccessPolicies -transform ..\xml\policies\xsl\accesscontrol.xsl -outfile ..\xml\policies\xml\samplestorenameAccessPoliciesOut.xml`

400 `TRANSFORM('/QIBM/ProdData/WebCommerce/xml/policies /xsl/accesscontrol.xsl') INSTROOT(instance_root) OUTFILE(output_file)`

c. Check the following log file to ensure that the transform has completed successfully:

- NT `drive:\WebSphere\CommerceServer\bin\xmltransform.db2.log`
- 2000 `drive:\Program Files\WebSphere\CommerceServer\bin\xmltransform.db2.log`
- AIX `/usr/WebSphere/CommerceServer/bin/xmltransform.db2.log`
- Solaris `/opt/WebSphere/CommerceServer/bin/xmltransform.db2.log`
- Linux `/opt/WebSphere/CommerceServer/bin/xmltransform.db2.log`
- 400 `/QIBM/UserData/WebCommerce/instances/ instancename/logs/TRNWCSXML.tx`

  If the transform was successful, the following message displays: `"<DATE> <TIME> java.lang.Class main XMLTransformer Transform Successful"`

5. Run the `xmltransform` command to transform *`samplestorename`*`AccessPolicies_locale.xml`.

   a. At a command prompt, change directories to the following:

   - NT `drive:\WebSphere\CommerceServer\bin`
   - 2000 `drive:\ProgramFiles\WebSphere\CommerceServer\bin`
   - AIX `/usr/WebSphere/CommerceServer\bin`
   - Solaris `/opt/WebSphere/CommerceServer/bin`
   - Linux `/opt/WebSphere/CommerceServer/bin`

   b. Then type: `xmltransform -infile ..\xml\policies\xml\samplestorenameAccessPolicies_locale.xml -transform ..\xml\policies\xsl\accesscontrolnls.xsl -outfile ..\xml\policies\xml\samplestorenameAccessPoliciesOut_locale.xml`

   c. 400 `TRNWCSXML INFILE(input file) TRANSFORM('/QIBM/ProdData/WebCommerce/xml/policies/ xsl/accesscontrolnls.xsl') INSTROOT(instance_root) OUTFILE(output_file)`

6. Make the following changes to the resulting XML files:

   a. In *`samplestorename`*`AccessPolicesOut.xml`, replace the opening and closing tags with the following:

```
<?xml version="1.0"?>
<!DOCTYPE accesscontrol-asset SYSTEM "accesscontrol.dtd">
<accesscontrol-asset>
</accesscontrol-asset>
```

b. In *samplestorename*`AccessPolicesOut_locale.xml`, replace the opening and closing tags with the following:

```
<?xml version="1.0" encoding="correct language code for the file"?>
<!DOCTYPE accesscontrol-asset SYSTEM "../accesscontrol.dtd">
<accesscontrol-asset>
</accesscontrol-asset>
```

c. In *samplestorename*`AccessPolicesOut_locale.xml`, replace the @locale with the &locale; for example change LANGUAGE_ID="@en_US" to LANGUAGE_ID="&en_US;"

d. In *samplestorename*`AccessPolicesOut_locale.xml`, locate the reference to the "acpoldesc" table. Remove the @ at the end of the ACPOLICY_ID value. For example, change `"@AllUsersExecuteInFashionAllUsersViews@"` to `"@AllUsersExecuteInFashionAllUsersViews"`.

e. In *samplestorename*`AccessPolicesOut.xml`, replace MEMBER_ID="MEMBER_ID" with MEMBER_ID="&MEMBER_ID;"

f. In *samplestorename*`AccessPolicesOut.xml` locate the reference to the "acpolicy" table. Remove the "@MEMBER_ID" at the end of the ACPOLICY_ID value. For example, change `"@AllUsersExecuteInFashionAllUsersViews@MEMBER_ID"` to `"@AllUsersExecuteInFashionAllUsersViews"`

7. Locate the store archive file for your store, (for example, `mystore.sar`) The store archive files are located by default in the following directory:

   - NT `drive:\WebSphere\CommerceServer\instances\instancename\sar`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\instances\instancename \sar`
   - AIX `/usr/WebSphere/CommerceServer/instances/instancename/sar`
   - Solaris `/opt/WebSphere/CommerceServer/instances/instancename/sar`
   - Linux `/opt/WebSphere/CommerceServer/instances/instancename/sar`
   - 400 `/QIBM/UserData/WebCommerce/instances/instancename/sar`

8. Rename *samplestorename*`AccessPolicesOut.xml` and *samplestorename*`AccessPoliciesOut_locale.xml` to the following: `accesscontrol.xml`

   Note: The locale specific `accesscontrol.xml` file is located by default in the data/locale directory, for example, data/en_US.

9. Open the store archive file using a ZIP program.

10. Replace the existing `accesscontrol.xml` and `locale specific accesscontrol.xml` in the store archive file with the ones you renamed in step 8.

11. Save the store archive file.

Part 6. Packaging your store

Chapter 25. Packaging a store

If you want to use your store as a sample to be delivered to others, to deploy it on another server or platform, or to use it as the basis of creating other stores, you can package it in the store archive form.

The sample stores provided with WebSphere Commerce are packaged as store archives. The files in the sample store archives are grouped as follows:

- Web assets: The files used to create your store pages, such as HTML files, JSP files, images, graphics, and include files. Web assets are grouped together as a compressed file (`webapp.zip`) in the store archive.
- Property resource bundles (optional): Contains the text for your store pages. If your store supports more than one language, the resource bundle will contain multiple bundles; that is, one bundle per language. Property resource bundles are grouped together as a compressed archive file (`properties.zip`) in the store archive.
- Store data assets: The data to be loaded into the database. Store data assets include data such as campaigns, catalog, currencies, fulfillment information, pricing, shipping, store, and taxation information.
- Payment assets: Configuration information for the IBM Payment Manager.
- A descriptor: An XML file, `sarinfo.xml`, that describes the store archive, including the names of the Web assets compressed archive file, the resource bundles, and the store database asset XML files. The `sarinfo.xml` file also contains the names of include files and consistency checking files, as well as information about the archive file that is needed during the publishing process. The `sarinfo.xml` is the only mandatory file in a store archive.

Creating a store archive

To package your store as a store archive, do the following:

1. Review the structure and content of the sample store archives provided with WebSphere Commerce.

   The store archive files are located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
   - AIX `/usr/WeSphere/CommerceServer/samplestores`
   - Solaris `/opt/WebSphere/CommerceServer/samplestores`
   - Linux `/opt/WebSphere/CommerceServer/samplestores`
   - 400 `/qibm/proddata/WebCommerce/samplestores`

   To view the store archive, use a decompression program.
2. Create a temporary directory on the WebSphere Commerce Server for your store. For example, *mystore*.
3. Group your Web assets (JSP files, HTML, images) together in a directory called `webapp`, in the temporary directory. Create a compressed archive file, using a ZIP program, of the `webapp` folder.

4. (Optional) Group your property resource bundles together in a directory called `properties` in the temporary directory. If your store supports more than one language, the resource bundle will contain multiple bundles; that is, one bundle per language. Create a compressed archive file, using a ZIP program, of the `properties` folder.
5. Group your store data assets together in a directory called `data` in the temporary directory. If your store supports several languages, create subdirectories for language-specific information using locale names. For example, en_US.
6. (Optional) Copy the `sarrule.xml` file from an existing store archive into your `data` directory. The `sarrule.xml` file is located in the `data` directory in the sample store archives. The `sarrule.xml` acts a consistency checker when you publish using Store Services. For more information on the sarrule file, see Appendix D, "sarrule.xml" on page 313.
7. Create a directory called `SAR-INF` in the temporary directory.
8. Create a `sarinfo.xml` file for your store archive. For more information on the XML specifications, see the archive descriptor, `sarinfo.dtd` in the following directory:
   - NT `drive:\WebSphere\CommerceServer\xml\sar`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\xml\sar`
   - AIX `/usr/WeSphere/CommerceServer/xml/sar`
   - Solaris `/opt/WebSphere/CommercServer/xml/sar`
   - Linux `/opt/WebSphere/CommercServer/xml/sar`
   - 400 `/qibm/proddata/WebCommerce/xml/sar`

   a. Using the example and information in Appendix C, "sarinfo.xml" on page 307 as your guide, create a `sarinfo.xml` file. The order in which the data assets are published is important, since some data assets must be published before others. As a result, the order of your assets, as specified in your `sarinfo.xml` file, should match the order of the assets specified in the `sarinfo.xml` files for the sample stores.

      Note: If you choose to include a `sarrule.xml` file in your store archive (see step 6), you should include information about `sarrule.xml` in your `sarinfo.xml` file. If you choose not to include a `sarrule.xml` file, ensure that you do not mention it in your `sarinfo.xml` file.

   b. Save `sarinfo.xml` in the `SAR-INF` directory you created in step 6.
9. Create a ZIP file composed of the Web assets ZIP file, the property resource bundles, the store data assets and the `sar-inf` directory. Name this ZIP file *storearchivename.sar*.
10. If you want to edit or publish your store archive using Store Services, save *storearchivename.sar* to the following directory:
    - NT `drive:\WebSphere\CommerceServer\instances\`*`instancename`*`\sar`
    - 2000 `drive:\Program Files\WebSphere\CommerceServer\instances\`*`instancename`*`\sar`
    - AIX `/usr/WebSphere/CommerceServer/instances/`*`instancename`*`/sar`
    - Solaris `/opt/WebSphere/CommerceServer/instances/`*`instancename`*`/sar`
    - Linux `/opt/WebSphere/CommerceServer/instances/`*`instancename`*`/sar`

- 400 /QIBM/UserData/WebCommerce/instances/*instancename*/sar

11. Your store archive will now display in the list of store archives in Store Services.

Creating a sample store archive

After packaging your store as a store archive, you may choose to use it as a sample store in Store Services. A sample store archive is a store archive that is meant to be copied and used as a base upon which to create new stores. In order to use your store archive as a sample store archive, do the following:

1. Save the store archive file to the following directory:
   - NT drive:\WebSphere\CommerceServer\samplestores\storeachivename
   - 2000 drive:\Program Files\WebSphere\CommerceServer\samplestores\storeachivename
   - AIX /usr/WeSphere/CommerceServer/samplestores/*storeachivename*
   - Solaris /opt/WebSphere/CommerceServer/samplestores/*storeachivename*
   - Linux /opt/WebSphere/CommerceServer/samplestores/*storeachivename*
   - 400 /qibm/proddata/WebCommerce/samplestores/*storeachivename*
2. (Optional) Create preview pages. In order for previews of your store pages to display in Store Services, you must create preview pages. Do the following:
   a. (Optional) In Store Services, select **New**. The Create Store Archive page displays. From the **Sample** list, select one of the sample stores, then click **Preview**. The pages that display are called preview pages. These pages are HTML files that present a pre-defined sample shopping flow, and act as a preview of the sample store.
   b. Determine the shopping flow you want to show in your preview pages.
   c. (Optional) Create some sample data in a published store. For example, add items into the shopping cart, and create a few shipping addresses and billing addresses. You will be creating the preview pages from this store, and data makes the pages look more realistic.
   d. Using Internet Explorer, browse the store. Save the HTML for each page, by selecting File, Save As. You should also save the style sheet (.css) and images. Save the files to the following directories:
      - *stylesheet*.css
        – NT drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instancename*.ear\wcstools.war\tools\devtools\storearchivename\preview
        – 2000 drive:\Program Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instancename*.ear\wcstools.war\tools\devtools\storearchivename\preview
        – AIX /usr/WebSphere/AppServer/installedApps WC_Enterprise_App_*instancename*.ear/wcstools.war/tools/devtools/*storearchivename*/preview
        – Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename.ear*/wcstools.war/tools/devtools/*storearchivename*/preview

  - Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename.ear*/wcstools.war/tools/devtools/*storearchivename*/preview
  - 400 /Qibm/UserData/WebAsAdv4/WAS*instancename*/installedApps/WC_Enterprise_App_*instancename*.ear/wcstools.war/tools/devtools/*storearchivename*/preview
- HTML
  - NT drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instancename*.ear\wcstools.war\tools\devtools\storearchivename\preview*locale*
  - 2000 drive:\Program Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instancename*.ear\wcstools.war\tools\devtools\storearchivename\preview*locale*
  - AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename*.ear/wcstools.war/tools/devtools/*storearchivename*/preview/*locale*
  - Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename.ear*/wcstools.war/tools/devtools/*storearchivename*/preview/*locale*
  - Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename.ear*/wcstools.war/tools/devtools/*storearchivename*/preview/*locale*
  - 400 /Qibm/UserData/WebAsAdv4/WAS*instancename*/installedApps/WC_Enterprise_App_*instancename*.ear/wcstools.war/tools/devtools/*storearchivename*/preview/*locale*
- locale independent images
  - NT drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instancename*.ear\wcstools.war\tools\devtools\storearchivename\preview\images
  - 2000 drive:\Program Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instancename*.ear\wcstools.war\tools\devtools\storearchivename\preview\images
  - AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename*.ear/wcstools.war/tools/devtools/*storearchivename*/preview/images
  - Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename.ear*/wcstools.war/tools/devtools/*storearchivename*/preview/images
  - Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename.ear*/wcstools.war/tools/devtools/*storearchivename*/preview/images
  - 400 /Qibm/UserData/WebAsAdv4/WAS*instancename*/installedApps/WC_Enterprise_App_*instancename*.ear/wcstools.war/tools/devtools/*storearchivename*/preview/images
- locale dependent images

– NT drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instancename*.ear\wcstools.war\tools\devtools\storearchivename\preview*locale*\images

– 2000 drive:\Program Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instancename*.ear\wcstools.war\tools\devtools\storearchivename\preview*locale*\images

– AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename*.ear/wcstools.war/tools/devtools/*storearchivename*/preview/*locale*/images

– Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename.ear*/wcstools.war/tools/devtools/*storearchivename*/preview/*locale*/images

– Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename.ear*/wcstools.war/tools/devtools/*storearchivename*/preview/*locale*/images

– 400 /Qibm/UserData/WebAsAdv4/WAS*instancename*/installedApps/WC_Enterprise_App_*instancename*.ear/wcstools.war/tools/devtools*storearchivename*/preview/*locale*/images

e. Since the location of the images and the css file have changed, you must change the references to the images and css file in the HTML pages. After changing the references, ensure that you can view the images when you open the HTML pages in a browser.

f. Change the links in the HTML pages from commands to links that reference the HTML files.

3. Create an HTML file that summarizes the store archive. This information will display in the **Sample description** in the Create Store Archive page in Store Services.

   a. Using the following example as your guide, create the new file.

```
<doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
 Describe store here
</body>
</html>
```

   b. Save this file as Feature_*locale*.html, where locale is the abbreviation for the language you are using. For example, en_US. Save this file to the following directory:

   - NT drive:\WebSphere\CommerceServer\samplestores\storeachivename
   - 2000 drive:\Program Files\WebSphere\CommerceServer\samplestores\storeachivename
   - AIX /usr/WeSphere/CommerceServer/samplestores/*storeachivename*
   - Solaris /opt/WebSphere/CommerceServer/samplestores/*storeachivename*
   - Linux /opt/WebSphere/CommerceServer/samplestores/*storeachivename*

- 400 `/qibm/proddata/WebCommerce/samplestores/`*`storeachivename`*

4. Add the store archive to the `sarregistry.xml` file. The `sarregistry.xml` file determines which store archives display in the **Sample** list in the Create Store Archive page in Store Services. The `sarregistry.xml` also determines which preview pages and feature file are associated with each store archive. `sarregistry.xml` is located in the following directory:

   - NT `drive:\WebSphere\CommerceServer\xml\tools\devtools`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\xml\tools\devtools`
   - AIX `/usr/WebSphere/CommerceServer/xml/tools/devtools`
   - Solaris `/opt/WebSphere/CommerceServer/xml/tools/devtools`
   - Linux `/opt/WebSphere/CommerceServer/xml/tools/devtools`
   - 400 `/QIBM/ProdData/WebCommerce/xml/tools/devtools`

   a. Using the following example as your guide, add the new store archive to the `sarregistry.xml`.

   ```
   <SampleSAR fileName="infashion_en_US_es_ES.sar" relativePath="InFashion">
           <html locale="es_ES" featureFile="InFashion/Feature_es_ES.html"
             sampleSite="RetailModel/preview/es_ES/index.html"/>
           <html locale="en_US" featureFile="InFashion/Feature_en_US.html"
             sampleSite="RetailModel/preview/en_US/index.html"/>
   </SampleSAR>
   ```

   This example defines the preview pages for the English and Spanish InFashion store archives. The lines in bold, define the home page for the English and Spanish preview pages.

5. You should now be able to see your store archive in the **Sample** list in the Create Store Archive page in Store Services.

Part 7. Publishing your store

In order to create a functioning store, the store front Web assets must be published to the WebSphere Commerce Server, and the store data must be published to the WebSphere Commerce database.

The chapters in this section discuss the publishing options WebSphere Commerce provides:

- Chapter 26, “Publishing a complete store” on page 209 - This chapter discusses publishing an entire store (store front and store data assets), if the store is in the form of a store archive, using either Store Services or the command line publish.
- Chapter 27, “Overview of loading store data” on page 221 - This chapter discusses publishing the store data assets to the database using the Loader package and other components of WebSphere Catalog Manager.
- Chapter 28, “Loading WebSphere Commerce database asset groups” on page 259 - This chapter discusses publishing groups of store data assets or all of the store data to the database using the Loader package and other components of WebSphere Catalog Manager.
- Chapter 29, “Publishing business accounts and contracts” on page 273 - This chapter discusses publishing the account, contract and product set assets.
- Chapter 30, “Publishing store front assets and store configuration files” on page 277 - This chapter discusses publishing the store front assets and the store configuration files.

Chapter 26. Publishing a complete store

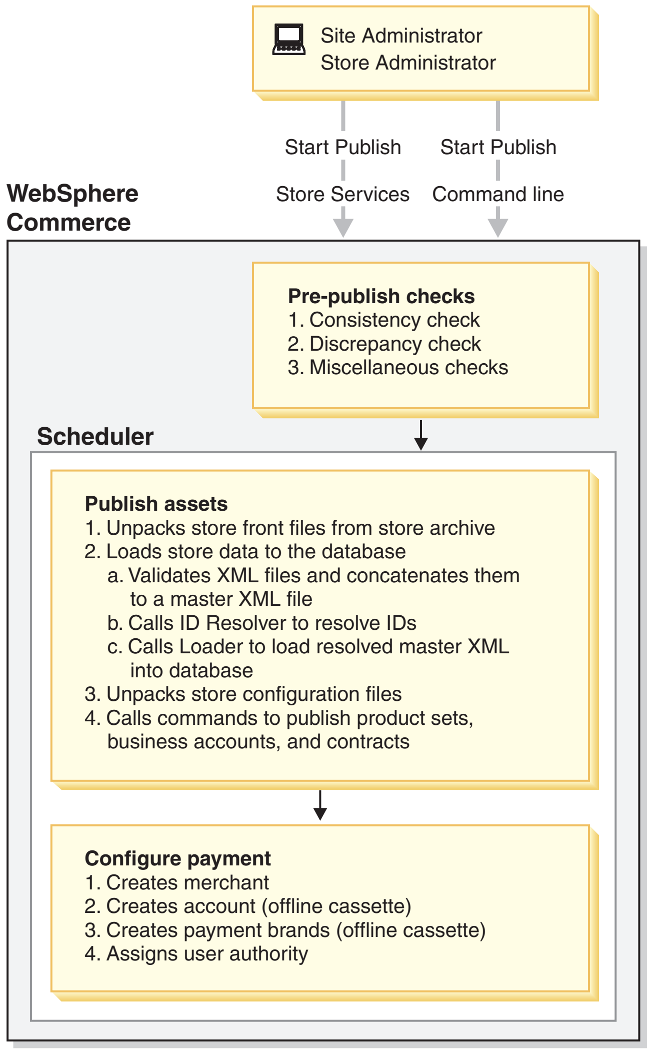
In order to create a functioning store, the store front Web assets must be published to the WebSphere Commerce Server, and the store data must be published to the WebSphere Commerce database. This chapter discusses publishing an entire store (store front and store data assets), if the store is in the form of a store archive, using either Store Services or the command line publish.

Note: If you prefer not to package your store as a store archive, you can publish the assets individually. For more information, seeChapter 27, “Overview of loading store data” on page 221, Chapter 28, “Loading WebSphere Commerce database asset groups” on page 259, Chapter 29, “Publishing business accounts and contracts” on page 273, and Chapter 30, “Publishing store front assets and store configuration files” on page 277.

Understanding publish in WebSphere Commerce

The publish option that is available from Store Services or from the command line allows you to publish a complete store (store front and store data assets) all at once. In order to use this option, your store assets must be packaged in the form of a store archive. For more information on packaging your store as a store archive, see Part 6, “Packaging your store” on page 199.

The following diagram outlines the steps in the publishing process.


Site Administrator
Store Administrator
Start Publish
Store Services
Start Publish
Command line
WebSphere Commerce
Pre-publish checks
1. Consistency check
2. Discrepancy check
3. Miscellaneous checks
Scheduler
Publish assets
1. Unpacks store front files from store archive
2. Loads store data to the database
a. Validates XML files and concatenates them to a master XML file
b. Calls ID Resolver to resolve IDs
c. Calls Loader to load resolved master XML into database
3. Unpacks store configuration files
4. Calls commands to publish product sets, business accounts, and contracts
Configure payment
1. Creates merchant
2. Creates account (offline cassette)
3. Creates payment brands (offline cassette)
4. Assigns user authority

Start publish

In order to publish a store, you must have Site Administrator or Store Administrator (for all stores) authority. Site Administrators or Store Administrators can initiate the publish process using either of the following methods:

- Store Services
- Command line

Both methods of publishing require you to select the store archive you want to publish. Using either method, you can select how much of the store archive you would like to publish. For example, you can select any combination of the following for publishing:

- Store database assets, with or without the online catalog data
- Web assets such as JSP files, HTML files and images
- Property files (the text for the store)

The first time you publish, it is recommended that you publish the entire store archive (all of the above), so that you can view a functional store. In subsequent publishes however, you may want to update only one of the following: the database assets, the Web assets or the property resource bundles, rather than republishing the entire store archive.

If you choose not to publish the online catalog data the first time you publish a store archive, some of the data assets (contracts and accounts) may not be published properly as they rely on data in the catalog. Also, if you do not publish both the catalog data and the Web assets the first time you publish, the store ID, catalog ID and language ID will not be created, and you will not be able to launch the store from Store Services. For more information, see "Creates parameters.jsp file" on page 218.

The online catalog data is composed of the following XML files in the store archive:

- `catalog.xml`
- `offering.xml`
- `store-catalog.xml`
- `store-catalog-shipping.xml`
- `store-catalog-tax.xml`
- `storefulfill.xml`

If you choose to publish this information at a later date, you can publish it using Store Services, or through the Loader package. For more information on loading a subset of data using database asset groups, for example the catalog group data listed above, see "Loading database asset groups" on page 267.

For more detailed information on how to publish a store archive using either Store Services or the command line, see the WebSphere Commerce online help.

Note: After you have initiated the publish process using either Store Services or the command line, you do not have to do anything else. All other steps listed in the preceding diagram, and in this chapter are completed by the WebSphere Commerce system.

Pre-publish checks

Once publish is initiated by the Site or Store Administrator, WebSphere Commerce performs several checks before beginning the actual publishing process. These checks include the following:

- Consistency check
- Discrepancy check
- Miscellaneous checks

Note: The command line publish checks the parameters passed to it, and completes consistency and discrepancy checks. However, if you publish using the command line publish, you will not see the messages described in the following paragraphs. Instead, if you publish using the update mode, the command line overwrites the existing store without prompting you. If you publish using the insert mode, the command line loads the new store. If the publish fails, a publish fail error message displays.

Consistency check

The pre-publish check uses the rules in the `sarrule.xml` file, to ensure that the information in the XML files is consistent with the Web assets in store archive. For example, if the `command.xml` file references a particular JSP file, the check ensures that the JSP file is in the `webapp.zip` in the store archive. If the consistency check finds an error, the error will be written to the log, but publishing continues as normal. For more information on the log files, see "Publish log files" on page 219.

For an example of a `sarrule.xml` file, see Appendix D, "sarrule.xml" on page 313.

Discrepancy check

During the pre-publish check, WebSphere Commerce checks for discrepancies between the store and the catalog. In particular, it checks the database for the following:

- An existing store with the same identifier in the STOREENT table: If the store already exists, a message displays asking if you want to overwrite the store, or cancel the publish.
- An existing catalog with the same catalog identifier: If the catalog already exists, a message displays asking if you want to overwrite the catalog, or cancel the publish. If the catalog belongs to others stores, a message displays stating which stores the catalog belongs to and asks if you want to continue the publish and overwrite the catalog, or cancel the publish.

Miscellaneous checks

During the pre-publish check, WebSphere Commerce also checks that the scheduler is enabled, that the cache triggers and the cache are disabled, and that the summary tables are disabled. If not, a warning message displays, telling you which feature is inappropriately enabled or disabled.

Note: The command line publish does not complete these checks.

Scheduler: As will be discussed in more detail in the following sections, the scheduler runs the publish job. If the scheduler is disabled, the publish process will not be able to run.

Cache and cache triggers: Leaving the cache on during publish may result in cache triggers being invoked during publish, when the database is updated. Cache triggers may generate unnecessary database activity that could result in a database transaction log overflow and affect the publishing performance. To disable the cache triggers, see the WebSphere Commerce online help.

Summary tables: DB2 Leaving the summary tables enabled may result in summary tables being updated during publish, which could result in a database transaction log overflow and affect the publishing performance. To disable the summary tables, see the WebSphere Commerce online help.

Publish assets

The publish assets phase of the publish process is a scheduled job run by the scheduler. When the scheduler runs the publish job, WebSphere Commerce completes the following actions:

- Unpacks the store front files from the store archive
- Loads store data from the XML files in the store archive to the database
- Creates parameters.jsp file
- Unpacks the store configuration files
- Calls the commands to publish business accounts and contracts
- Updates the registry components

Unpacks the store front files from the store archive

Unpacking the Web assets from the store archive to the WebSphere Commerce Server is the first action in the publish assets phase. While unpacking the files from the store archive, WebSphere Commerce does the following:

Unpacks the Web assets and copies them to the following locations on the WebSphere Commerce Server:

- The JSP files, HTML, include files, images and graphics are published to the store directory *(storedir)* under the Stores Web application document root:
  - NT `drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_`*`instancename`*`.ear\wcstores.war\`*`storedir`*
  - 2000 `drive:\Program Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_`*`instancename`*`.ear\wcstores.war\`*`storedir`*
  - AIX `/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*
  - Solaris `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*
  - Linux `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*
  - 400 `/QIBM/UserData/WebASAdv4/WASinstancename/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*
- The resource bundles and properties files are published to the application properties path:
  - NT `drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_`*`instancename`*`.ear\wcstores.war\WEB-INF\classes\`*`storedir`*
  - 2000 `drive:\Program Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_`*`instancename`*`.ear\wcstores.war\WEB-INF\classes\`*`storedir`*
  - AIX `/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/WEB-INF/classes/`*`storedir`*
  - Solaris `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/WEB-INF/classes/`*`storedir`*
  - Linux `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/WEB-INF/classes/`*`storedir`*
  - 400 `/QIBM/UserData/WebASAdv4/WASinstancename/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/WEB-INF/classes/`*`storedir`*

Loads store data from the XML files in the store archive to the database

While loading the store data from the XML files in the store archive to the database, WebSphere Commerce does the following:

Note: Only the XML files of type db-load are loaded into the database. The file type is specified in the sarinfo.xml file. For more information on the sarinfo.xml file, see Appendix C, "sarinfo.xml" on page 307

Validates the XML files in the store archive and concatenates them to a master XML file: WebSphere Commerce validates the XML files using the corresponding DTD files. The DTD files 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
- Linux /opt/WebSphere/CommerceServer/xml/sar
- 400 /qibm/proddata/WebCommerce/xml/sar

If the XML files are not valid, WebSphere Commerce writes an error to the error log. If the errors in the error log exceed the maximum number of errors specified in the DevTools portion of the WebSphere Commerce Configuration File, *instance_name*.xml, (by default MaxErrorsInSarXML=1) publish fails. The WebSphere Commerce Configuration File, *instance_name*.xml is located in the following directory:

- NT drive:\WebSphere\CommerceServer\instances\ *instancename*\xml\ *instance_name*.xml
- 2000 drive:\Program Files\WebSphere\CommerceServer\instances\ *instancename*\xml\ *instance_name*.xml
- AIX /usr/WebSphere/CommerceServer/instances/*instancename* /xml/*instance_name*.xml
- Solaris /opt/WebSphere/CommerceServer/instances/*instancename* /xml/*instance_name*.xml
- Linux /opt/WebSphere/CommerceServer/instances/*instancename* /xml/*instance_name*.xml
- 400 /QIBM/UserData/WebCommerce/instances/*instancename* /xml/*instance_name*.xml

After the XML files are validated, they are then concatenated into one file: *storename*master.xml. The files are concatenated according to the priority specified in the sarinfo.xml file. For more information, see Appendix C, "sarinfo.xml" on page 307. The *storename*master.xml file is located in the following directory:

- NT drive:\WebSphere\CommerceServer\temp\ *instancename*\tools\devtools
- 2000 drive:\Program Files\WebSphere\CommerceServer\temp\ *instancename*\tools\devtools
- AIX /usr/WebSphere/CommerceServer/temp/*instancename*/tools/devtools
- Solaris /opt/WebSphere/CommerceServer/temp/*instancename*/tools/devtools
- Linux /opt/WebSphere/CommerceServer/temp/*instancename*/tools/devtools
- 400 /QIBM/UserData/WebCommerce/instances/*instancename* /temp/tools/devtools

Calls the ID Resolver to resolve IDs: The ID Resolver, which is a Loader package utility, generates unique identifiers for XML elements in the store archive XML files. For example, the ID Resolver replaces the @ alias used in the sample store XML files with a unique value. For an example of internal-alias resolution used in the sample stores, see Appendix B, "Creating your data" on page 305.

Note: The ID Resolver can also resolve identifiers for already published stores, when you republish. For example if you have published the store archive once, and you need to republish the store archive or portions of it, ID Resolver retrieves the unique identifiers from the database and uses those during the republishing process.

For more information on the ID Resolver and the other components of the Loader package, see Chapter 27, "Overview of loading store data" on page 221.

When the Store Services or command line publish calls the ID Resolver, it must specify which ID Resolver method to use. The ID Resolver has several methods, which can be used to process the ID Resolver input; specifically, whether to treat the data as if identifiers exist in the original data (update method) or do not (load method). Mixed method is used when some identifiers exist and others do not.

Note: Mixed method is the recommended method for Store Services. If you are publishing using command line, you may also want to specify the mixed mode, particularly if you don't know if the data being published already exists in the database.

You can specify which method Store Services or the command line publish will use in the WebSphere Commerce Configuration File, `*instance_name*.xml`. By default, Store Services uses the mixed method. For more information on the ID Resolver methods, see "ID Resolve command" on page 225.

Store Services and the command line publish must also specify a customizer file to be used with the ID Resolver. If you do not specify a customizer file in the WebSphere Commerce Configuration File, `*instance_name*.xml`, the publish code will use one of the default customizer files: DBConnectionCustomizer or OracleConnectionCustomizer.

Oracle

The OracleConnectionCustomizer customizer file is located in the following directory:

- NT `drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instancename*.ear\properties`
- 2000 `drive:\Program Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instancename*.ear\properties`
- AIX `/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename*.ear/properties`
- Solaris `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename*.ear/properties`
- Linux `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename*.ear/properties`
- 400 `/QIBM/UserData/WebASAdv4/WASinstancename/installedApps/WC_Enterprise_App_*instancename*.ear/properties`

DB2

The DBConnectionCustomizer file is located in the following ZIP file:

- NT drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instancename*.ear\lib\loader\idresgen.zip
- 2000 drive:\Program Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instancename*.ear\lib\loader\idresgen.zip
- AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename*.ear/lib/loader/idresgen.zip
- Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename*.ear/lib/loader/idresgen.zip
- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instancename*.ear/lib/loader/idresgen.zip
- 400 /QIBM/UserData/WebASAdv4/WASinstancename/installedApps/WC_Enterprise_App_*instancename*.ear/lib/loader/idresgen.zip

Note: If you want to specify your own customizer file, you must add the following to the DevTools section of the *instance_name*.xml file:

- IDResolverCustomizerFile="*myIDResolverCustomizerFile*"

By default the WebSphere Commerce Configuration File, *instance_name*.xml, does not specify a value for this attribute.

The ID Resolver uses the *storename*master.xml and corresponding DTD file, wcs.dtd. After the IDs are resolved, ID Resolver creates the following file, *storenametime_stamp*master.xml file, which contains the unique identifiers. If an error occurs during the ID resolving process, the Loader package creates an error.xml file, *storename*master.error.xml.

Note: The publishing process saves these temporary files automatically when a publishing attempt fails. However, if a publish is successful, these files are deleted by default. You may want to keep these files for troubleshooting purposes, or for the purpose of publishing and working with a store in the WebSphere Test Environment. To save the temporary files, see "Store Service parameters" in the WebSphere Commerce online help.

*storenametime_stamp*master.xml file and *storename*master.error.xml are located in the following directory:

- NT drive:\WebSphere\CommerceServer\temp\instancename\tools\devtools
- 2000 drive:\Program Files\WebSphere\CommerceServer\temp\instancename\tools\devtools
- AIX /usr/WebSphere/CommerceServer/temp/*instancename*/tools/devtools
- Solaris /opt/WebSphere/CommerceServer/temp/*instancename*/tools/devtools

- Linux `/opt/WebSphere/CommerceServer/temp/`*`instancename`*`/tools/devtools`
- 400 `/QIBM/UserData/WebCommerce/instances/`*`instancename`*`/`
  `temp/tools/devtools`

Calls the Loader package to load the resolved master XML file into the database: The Loader package loads the resolved *`storenametime_stamp`*`master.xml` into the database. If an error occurs during the loading process, the Loader package creates an `error.xml` file, *`storenametime_stamp`* `master.error.xml`.

For more information on the Loader package, see Chapter 27, "Overview of loading store data" on page 221.

When the Store Services or command line publish calls the Loader package, it must specify which Loader method to use. Store Services can use any of the following Loader methods:
- SQL import
- Import
- Load

Note: By default Store Services uses the SQL import method.
You can specify which method Store Services or the command line publish will call in the WebSphere Commerce Configuration File, *`instance_name`*`.xml`, using the LoaderMode attribute in the DevTools element.
- SQL import: This method uses Java Database Connectivity (JDBC) to insert and update data, providing the most flexible method of operation but also the slowest for importing large amounts of data into a small number of tables. It allows column-level update. It is recommended that you use SQL import.

  Note: SQL import method is the safest method to use because it will not corrupt your database if the data is invalid. Before you can load using SQL import, the records must meet the database schema constraints. The other Loader methods are faster because they are bulk loaded into the database without much checking. As a result you must be certain of data correctness before using the other methods.
- Import: This method uses DB2 native import functions and allows cell-level update with medium speed and flexibility. This method is not available with Oracle.
- Load: This method uses the native facilities of the RDBMS (DB2 Load or SQLLoad) and is the fastest method for loading large amounts of data into a small number of tables.

For more information on the methods in the load command, see "Load command" on page 232.

Store Services and the command line publish must also specify a customizer file to be used with the Loader. If you do not specify a customizer file in the WebSphere Commerce Configuration File, *`instance_name`*`.xml`, the publish code will use the default customizer file: MassLoadCustomizer.

Note: If you want to specify your own customizer file, you must add the following to the DevTools section of the *`instance_name`*`.xml` file:
- `LoaderCustomizerFile="`*`myLoaderCustomizerFile`*`"`

By default the WebSphere Commerce Configuration File, `*instance_name*.xml`, does not specify a value for this attribute.

Creates parameters.jsp file

The publish process creates the file `parameters.jsp`. This file includes three parameters: storeId, catalogId, and langId. The index.jsp file in the sample stores uses these parameters to launch the store.

In order for the publish process to create the `parameters.jsp` file, the catalog data and Web assets much be published concurrently at least once. If these two items are not published together Store Services will not be able to launch the store.

`parameters.jsp` is located in the following directory:

- NT `drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_`*`instancename`*`.ear\wcstores.war\`*`storedir`*`\include`
- 2000 `drive:\Program Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_`*`instancename`*`.ear\wcstores.war\`*`storedir`*`\include`
- AIX `/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*`/include`
- Solaris `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*`/include`
- Linux `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*`/include`
- 400 `/QIBM/UserData/WebASAdv4/WASinstancename/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*`/include`

Unpack store configuration files

The Business ToolTech and NewFashion sample store archives also including the following files:

- `tools_properties.zip`
- `tools_xml.zip`
- `runtime_xml.zip`

These files are registered in the `sarinfo.xml file` and are used by Store Services to configure stores. During the publish process these files are unpacked and copied to the directories specified in the WebSphere Commerce Configuration File, `*instance_name*.xml`. For more information, see “Publishing store front assets and store configuration files by copying to the WebSphere Commerce Server” on page 278.

Note: These files should not be changed, removed, copied to other stores, or saved in a different directory.

Calls command to publish business accounts and contracts

Some of the store database assets, (contracts and business accounts) cannot be loaded by the Loader package, so publish also calls the corresponding commands to publish those assets to the WebSphere Commerce Server. These commands are as follows:

- AccountImport — Creates a business account from the `businessaccount.xml` file in the store archive.

- ContractImportApprovedVersion — Imports a contract from the `contract.xml` file in the store archive.
- ProductSetPublish — Synchronizes the product set data in the database with the catalog before business accounts and contracts are created. Store Services and the command line publish call the ProductSetPublish command, which then calls the AccountImport and ContractImportApprovedVersion commands.

For more information on publishing business accounts and contracts, see Chapter 29, “Publishing business accounts and contracts” on page 273.

Updates registry components

The last action in the publish process is updating the registry components. Publish updates all of the registries in WebSphere Commerce. For more information on the registries, see the WebSphere Commerce online help.

Error handling

If an error occurs during the publish assets phase of the publish process, you can view the error message either in the publish logs (see “Publish log files”) or through the Publish Summary page in Store Services.

Configure payment

The last step in the publishing process is to configure payment. WebSphere Commerce supports the IBM Payment Manager. If you plan to use Payment Manager as your method of processing payment, you should create a payment XML file as described in Chapter 13, “Payment assets” on page 115. If a payment XML file is included in the store archive being published, WebSphere Commerce will complete the following payment configuration during publish:

- Create the merchant.
- Create the account (for offline cassettes only).
- Create the brands specified in `paymentinfo.xml` (for offline cassettes only).
- Assign user authority.

Error handling

If an error occurs during the configure payment phase of the publish process, you can view the error message in the publish logs (see “Publish log files”).

Publish log files

Any errors encountered during the publish assets phase of the publishing process are written to the following log and trace files:

- `messages.txt`: Contains error messages from the Loader package part of the publishing process. Check this log first when publish fails. Line or column numbers mentioned in these error messages refer to the temporary `master.xml` files: `*storename*master.xml`, or `*storenametime_stamp*master.xml`.
- `trace.txt`: Contains trace information for the Loader package and ID Resolver parts of the publishing process. By default, trace.txt is turned off.
- `ecmsg_*instancename_timestamp*.log`: Logs all running error messages from the WebSphere Commerce Server.
- `wcs.log file`: Contains the output from all applications (including publishing) running in the WebSphere Application Server to the standard console.

The log files are located in the following directory:

- NT `drive:\WebSphere\CommerceServer\instances\*instancename*\logs`
- 2000 `drive:\Program Files\WebSphere\CommerceServer\*instancename*\logs`

- AIX `/usr/WebSphere/CommerceServer/instances/`*`instancename`*`/logs`
- Solaris `/opt/WebSphere/CommerceServer/instances/`*`instancename`*`/logs`
- Linux `/opt/WebSphere/CommerceServer/instances/`*`instancename`*`/logs`
- 400 `/QIBM/UserData/WebCommerce/instances/`*`instancename`*`/logs`

To configure the trace.txt and messages.txt log files (that is, adjust the log level or other options), edit the following file:

- NT `drive:\WebSphere\CommerceServer\xml\loader\WCALoggerConfig.xml`
- 2000 `drive:\Program Files\WebSphere\CommerceServer\xml\loader\WCALoggerConfig.xml`
- AIX `/usr/WebSphere/CommerceServer/xml/loader/WCALoggerConfig.xml`
- Solaris `/opt/WebSphere/CommerceServer/xml/loader/WCALoggerConfig.xml`
- Linux `/opt/WebSphere/CommerceServer/xml/loader/WCALoggerConfig.xml`
- 400 `/QIBM/UserData/WebCommerce/instances/instancename/xml/WCALoggerConfig.xml`

Note: For more information on configuring the `WCALoggerConfig.xml` file, see the *WebSphere Commerce Catalog Manager User's Guide*.

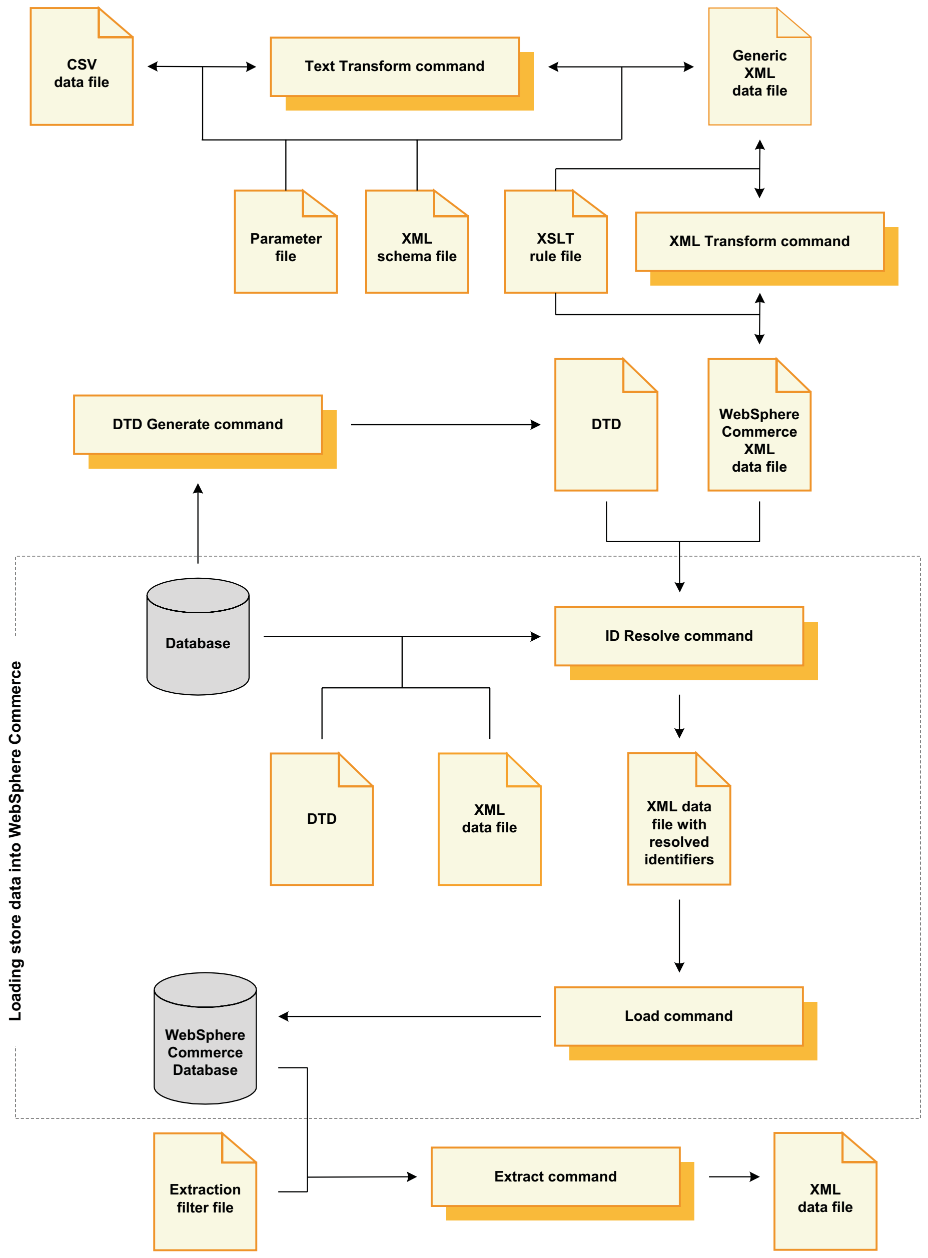
Chapter 27. Overview of loading store data

After creating your store data, you can choose to package it as a store archive and publish it using Store Services or you can load it directly into the WebSphere Commerce Server database using the WebSphere Commerce Catalog Manager Loader package. Refer to Chapter 28, “Loading WebSphere Commerce database asset groups” on page 259 and “Loading database asset groups” on page 267 for information on the loading process for WebSphere Commerce database asset groups.

The Catalog Manager provides six command-line utilities (collectively referred to here as the ″Loader package″) and three related administrative tools that you can use to prepare data for loading as well as to load data into your store. These commands and tools use Extensible Markup Language (XML) data files to manage the information.

Understanding data loading in WebSphere Commerce

The data preparation, loading, and extraction processes that you can perform using the Loader package commands are shown in the following figure.



Note that a dotted line indicates the two processes that are most commonly used to load store data into a WebSphere Commerce Server database: resolving identifiers and loading the data. These processes are the focus of this chapter.

For more information on preparing your data for loading into a WebSphere Commerce Server database, refer to Part 4, “Developing your store data” on page 35.

The following two Loader package command-line utilities are commonly used for loading data into a WebSphere Commerce Server database:

- **ID Resolve command**

  To load XML data into a WebSphere Commerce Server database using the Loader package, the XML elements must map directly to the schema of the targeted WebSphere Commerce Server database. All XML elements that have attributes corresponding to unique or primary keys in the database schema must have unique identifiers; and all non-nullable columns of the database schema must have corresponding attributes defined with non-null values. The ID Resolver can generate unique identifiers for unique or primary key attributes of qualifying XML elements.

  Note: As referred to in this document, an identifier is a value in a single column of a database table that gives each row a unique identity. If you use the ID Resolver to generate identifiers, it obtains a base value from the KEYS or SUBKEYS table and increments the value sequentially to resolve an identifier for each row in the database table.

  For information on this command, refer to "ID Resolve command" on page 225, "Using the Loader package commands and scripts" on page 249, and "Examples of resolving identifiers" on page 250.

- **Load command**

  The Loader uses valid and well-formed XML files as input to load data into the database. Elements of the XML document map to table names in the database; and element attributes map to columns.

  Note: Refer to the World Wide Web Consortium (W3C) XML guidelines for a description of the validity and well-formedness constraints.

  For information on this command, refer to "Load command" on page 232, "Using the Loader package commands and scripts" on page 249, and "Example of loading data" on page 257.

These commands are the primary focus of this chapter.

The following Loader package command-line utilities can also be used to manage your data:

- **DTD Generate command**

  The DTD Generator generates a document type definition (DTD) that describes the tables and columns of the target database into which XML data is to be loaded. The DTD Generator can also generate an XML schema for the database.

  The DTD Generator can create a DTD based on the WebSphere Commerce database schema. If you use the DTDs provided with the sample store archives and you do not modify the database schema, you normally do not need to generate a DTD using the DTD Generator.

  Refer to "DTD Generate command" on page 239 for more information.

- **Extract command**

  The Extractor uses a query against a database to extract selected subsets of data from the database into an XML document.

  You can use this command to extract data from your database into an XML format.

  Refer to "Extract command" on page 242 for more information.

- **Text Transform command**

  The Text Transformer transforms data between a character-delimited variable format and an XML data format.

  If your data cannot be extracted directly from a database in an XML format, for example, you can save your data in a character-delimited variable format then use this command to transform it into an XML format.

  Refer to “Text Transform command” on page 244 for more information.
- **XML Transform command**

  The XML Transformer transforms the data in an XML document to an alternate XML format. It uses Extensible Stylesheet Language (XSL) to define the mapping rules for the transformation.

  You can use this command to convert your XML data into a format that maps directly to the schema of the target WebSphere Commerce database into which you want to load the data.

  Refer to “XML Transform command” on page 245 for more information.

These commands are not the primary focus of this chapter. For detailed information on these commands, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User’s Guide*.

The WebSphere Commerce Catalog Manager also includes the following tools to assist in the administration of its data-management functions:

- **Text Transformation tool**

  The Text Transformation tool helps you process a transformation of data between a character-delimited variable format and an XML data format using the Text Transform command.
- **XSL editor**

  The XSL editor gives you a visual interface for editing XSL files that can be used by the XML Transformer. Using the XSL editor, you establish the association from an element in a source DTD to an element in a target DTD when defining the mapping rules for transforming data between XML formats.
- **Web editor**

  The Web editor enables you to create, modify, and delete data in a database through a Web browser.

These tools are not the primary focus of this chapter. For detailed information on these tools, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User’s Guide*.

Loader package commands for loading store data

ID Resolve command

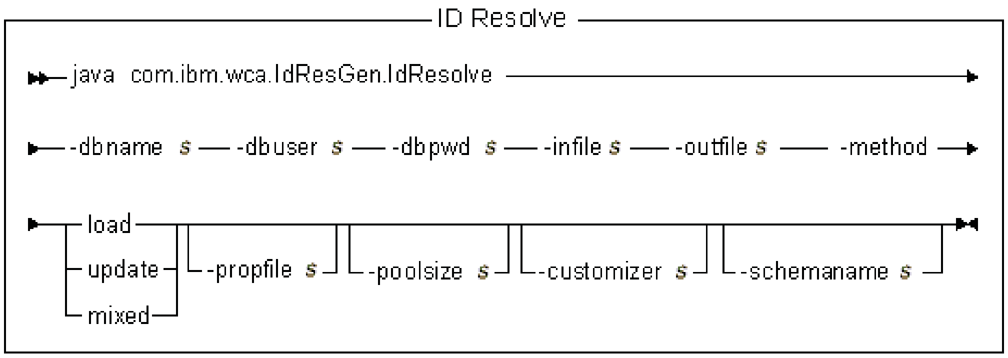
This command generates unique identifiers for XML data elements that require them before the elements can be loaded into a database. If your source XML data already supplies the necessary unique identifiers, you do not have to run the ID Resolver.

The WebSphere Commerce database schema defines primary and foreign keys within its tables that are used to represent various relationships between the tables. For this reason, WebSphere Commerce XML elements must contain corresponding attributes with unique identifiers. Within the WebSphere Commerce Server database, the tables whose identifiers must be resolved are those defined in the KEYS and SUBKEYS tables. These tables are called *primary tables* within WebSphere Commerce. For more information on the KEYS and SUBKEYS tables, see the WebSphere Commerce online help.

Note: If it is necessary to resolve identifiers for a table that is not defined in the KEYS or SUBKEYS table, add the table to the SUBKEYS table before running the ID Resolver.

Because WebSphere Commerce XML elements and attributes are intended to be portable across databases and across database instances, its identifiers are usually represented using internal aliases. Before the data can be loaded into any WebSphere Commerce Server database, these aliases must be resolved into valid numeric identifiers. For more information, refer to Appendix B, "Creating your data" on page 305.

Windows AIX Solaris Linux



Notes:

1. The above diagram is intended primarily as a reference for the command parameters. The command file or script provided for this command and listed under "Using the Loader package commands and scripts" on page 249 acts as a wrapper to the actual Java command and accepts the same parameters; therefore, it is recommended that you use the command file or script rather than invoke the Java command directly.
2. Filenames specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

-dbname
Name of the target database

-dbuser
Name of the user connecting to the database

-dbpwd
Password for the user connecting to the database

-infile Name of the input XML document containing table records

-outfile
Name of the output XML file to be produced; this file can be used as input to the Loader

-method
Method to be used in processing the input file

- Use the load method to process the input file if ***all*** records in the file ***do not exist*** in the database.
- Use the update method to process the input file if ***all*** records in the file ***exist*** in the database.
- Use the mixed method to process the input file if ***only some*** records in the file ***exist*** in the database.

The default method is load.

-propfile
Text file containing Java properties in the form of name=value pairs. This property file sets the way in which the ID Resolver resolves identifiers. It is used to describe which columns of a primary entry should be used as lookups for tables that require the identifier of a primary row. This file defines the column names for foreign-key identifier lookup and the select predicate for main table (such as CATEGORY and PRODUCT) queries. You can omit entries in this file for tables that have a defined unique index that does not include the identifier. This parameter is optional. `IdResolveKeys.properties` is the default file. This property file can be specified as shown in either of the following examples:

```
-propfile d:\WebSphere\CommerceServer\prop\idresprop.properties
-propfile d:\WebSphere\CommerceServer\prop\idresprop
```

If this file exists in the current directory, the same file can be specified as shown in the following example:

```
-propfile idresprop.properties
```

If this file exists in a directory specified in the classpath system-environment variable, the same file can be specified as shown in the following example:

```
-propfile idresprop
```

For more information on creating and specifying a new properties file for use with the ID Resolver, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

-poolsize
Number of identifiers to be reserved. This parameter is optional. The default number is 50.

-customizer

Name of the customizer property file to be used. This parameter is optional. The customizer property file sets the way that the ID Resolver functions. `DB2ConnectionCustomizer.properties` is the default file. The customizer property file can be specified as shown in either of the following examples:

```
-customizer d:\WebSphere\CommerceServer\prop\idres.properties
-customizer d:\WebSphere\CommerceServer\prop\idres
```

If this file exists in the current directory, the same file can be specified as shown in the following example:

```
-customizer idres.properties
```

If this file exists in a directory specified in the classpath system-environment variable, the same file can be specified as shown in the following example:

```
-customizer idres
```

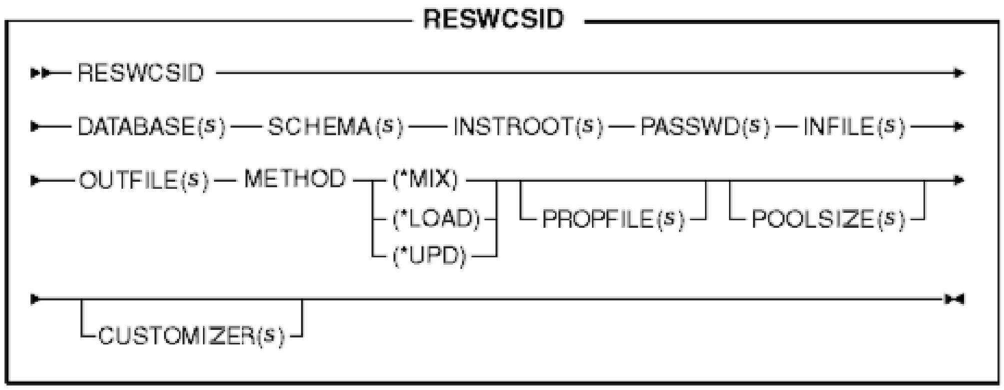
For more information on creating and specifying a new customizer property file, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

-schemaname

Name of the target database schema. This parameter is optional.

If this parameter is not specified when running the command, the command looks for a name=value pair in the customizer property file that specifies the value of SchemaName. If this pair is present in the property file, the command uses the value specified. If neither a command-line nor a property-file specification for this parameter exists, the command defaults to the schema name of the table in the database.

400



Note: Filenames specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

DATABASE

Name of the target database, as displayed in the relational database directory

SCHEMA
Name of the target database schema; this is the same as the instance name

INSTROOT
Full name of the WebSphere Commerce instance root path, such as `/QIBM/UserData/WebCommerce/instances/`*instance_name*

PASSWD
Password for the WebSphere Commerce instance

INFILE
Name of the input XML document containing table records

OUTFILE
Name of the output XML file to be produced; this file can be used as input to the Loader

METHOD
Method to be used in processing the input file

- Use the load method (*LOAD) to process the input file if ***all*** records in the file ***do not exist*** in the database.
- Use the update method (*UPD) to process the input file if ***all*** records in the file ***exist*** in the database.
- Use the mixed method (*MIX) to process the input file if ***only some*** records in the file ***exist*** in the database.

PROPFILE
Text file containing Java properties in the form of name=value pairs. This property file sets the way in which the ID Resolver resolves identifiers. It is used to describe which columns of a primary entry should be used as lookups for tables that require the identifier of a primary row. This file defines the column names for foreign-key identifier lookup and the select predicate for main table (such as CATEGORY and PRODUCT) queries. You can omit entries in this file for tables that have a defined unique index that does not include the identifier. This parameter is optional. `IdResolveKeys.properties`is the default file. This property file can be specified as shown in either of the following examples:

```
PROPFILE(/wc/prop/idresprop.properties)
PROPFILE(/wc/prop/idresprop)
```

If this file exists in the current directory, the same file can be specified as shown in the following example:

```
PROPFILE(idresprop.properties)
```

If this file exists in a directory specified in the classpath system-environment variable, the same file can be specified as shown in the following example:

```
PROPFILE(idresprop)
```

For more information on creating and specifying a new properties file for use with the ID Resolver, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

POOLSIZE
Number of identifiers to be reserved. This parameter is optional. The default number is 50.

CUSTOMIZER

Name of the customizer property file to be used. This parameter is optional. The customizer property file sets the way that the ID Resolver functions. The default file is `ISeries_RESWCSID_Customizer.properties`. The customizer property file can be specified as shown in either of the following examples:

```
CUSTOMIZER(/wc/prop/idres.properties)
CUSTOMIZER(/wc/prop/idres)
```

If this file exists in the current directory, the same file can be specified as shown in the following example:

```
CUSTOMIZER(idres.properties)
```

If this file exists in a directory specified in the classpath system-environment variable, the same file can be specified as shown in the following example:

```
CUSTOMIZER(idres)
```

For more information on creating and specifying a new customizer property file, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

Resolution techniques:
The ID Resolver resolves identifiers using a combination of two or three of the following techniques, depending on whether or not a properties file is used.

- **Internal-alias resolution**

  When using internal-alias ID resolution, an alias is substituted for the unique key (identifier) in the source XML document. This alias is then used elsewhere in the XML file to refer to that element.

  Internal aliases must be used consistently throughout the XML file. For example, if an address-book ID, ADDRBOOK_ID, is aliased to @addrbook_1, all foreign-key references to that ID in the file must use @addrbook_1.

  Note that aliases are transient to the specific XML file. They are not saved; and an alias cannot be used in a separate XML file without introducing the alias again. During publish in Store Services, however, publish concatenates the XML files so that resolution can occur across all of the data.
- **Unique-index resolution**

  The ID Resolver can also analyze the database schema to determine whether or not there is a unique index that fulfills its requirements. The ID Resolver looks for a unique index only when there is no entry in the properties file for the table being analyzed or when there is no properties file. If these conditions are true, a unique-index check is performed. The unique index is considered valid if it exists and does not include the primary key for the table.
- **Properties-file specification**

  The ID Resolver lets you use an alternate Java properties file to describe which columns of a primary entry should be used as lookups for tables that require the identifier of a primary row.

The sample store archives provided with WebSphere Commerce use internal aliases in their XML files. This allows the store archives to be portable across databases. Although the unique-index and properties-file specification techniques also allow for portability across databases, a user can change what the unique columns are at any time and cause problems when these techniques are later used for ID resolution. If a user changes a unique column, for example, the column name must then be changed in the property-file definition. With the internal-alias technique, however, a change in the database does not necessitate a change in the XML or properties files. During publish in Store Services or using the Loader package, the ID Resolver replaces the alias with a unique value. Once the data is loaded, the aliases are transparent to the user. For more information, refer to Appendix B, "Creating your data" on page 305.

The ID Resolver uses the following process:

- If your input XML data has an element from a primary table that already has a hard-coded identifier ("12345" for example), the ID Resolver does not create a new identifier for that element.
- If your input XML data has an element from a primary table that does not have an identifier, the ID Resolver looks in the database to see whether or not there is already a row for this element.

  Looking up the element in the database requires that other columns in the element be used to form a unique key. These other columns can be specified in the properties file; or the ID Resolver can be allowed to determine which columns to use.

  – If a properties file is being used and there is an entry in the properties file for the table being analyzed, the ID Resolver uses the columns specified in the properties file to form the unique key.

  – If there is no properties file being used or there is no entry in the properties file for the table being analyzed, the ID Resolver uses unique-index resolution.

    Unique-index resolution uses any of the specified unique indexes on a table as a means of locating the identifier. For example, MEMBER_ID plus IDENTIFIER is a unique index on the CATALOG table and can therefore be used as a resolution point to the foreign key CATALOG_ID of the CATALOGDSC table.

  The element is deemed to already exist in the database if there is a row with the same unique key; otherwise, it is seen as a new piece of data.
- If the element already exists as a row in the database, its identifier is retrieved and saved so that it can be used later. Otherwise, a new identifier is generated by ID Resolver using an available value in the KEYS or SUBKEYS table.
- If you specified an internal alias for the element ("@store_id_1" for example) in the XML document, that alias is associated with the identifier so that the identifier can be looked up later using the same internal alias.
- Subsequent XML document elements that need to refer to an element from the primary table use either the internal alias if the primary table element had one ("@store_id_1" for example) or the values of the lookup columns if it did not ("@WC2001@100" for example). In either case, the value specified is used to look up the actual identifier and the value is replaced with that identifier.
- When the output XML document is produced, all primary table elements have actual identifiers in them and all elements that refer to those primary table elements refer to them using the actual identifiers, not the internal aliases or lookup column values mentioned above. This is the fully resolved XML document.

Methods for the ID Resolve command:
The ID Resolve command lets you choose the load, update, or mixed method to process the input file.

Load method:
The load method for the ID Resolver is used to generate new identifiers for all new records that are loaded into the database.

Note: If you specify the load method for the ID Resolver, the records in the input file should not already exist in the database. If the load method is used with the ID Resolver and a record in the source XML file already exists in the target database, the Loader will generate an error when you load the data. The ID Resolver will assign a new primary key to the record in the XML file during ID resolution; but when you load the data into the database, an error will be generated. The Loader will not stop at the point of processing the duplicate record; but it will report an error and the duplicate record will not be loaded into the database.

The following example is used to generate identifiers for data elements that are new to the database:

- Windows

  ```
  idresgen -dbname db -dbuser user -dbpwd pwd -infile input.xml
  -outfile output.xml -method load -customizer customizer -schemaname wcsadmin
  ```
- AIX Solaris Linux

  ```
  ./idresgen.sh -dbname db -dbuser user -dbpwd pwd -infile input.xml
  -outfile output.xml -method load -customizer customizer -schemaname wcsadmin
  ```
- 400

  ```
  QWEBCOMM/RESWCSID DATABASE(DATABASE_NAME) SCHEMA(WCSADMIN)
  INSTROOT(/QIBM/UserData/WebCommerce/instances/mser)
  PASSWD(mypassword) INFILE(input.xml) OUTFILE(output.xml)
  METHOD(*LOAD)
  ```

Note: Refer to “Using the Loader package commands and scripts” on page 249 for the location of the appropriate ID Resolve command or script.

Update method:
If you specify the update method for the ID Resolver, the records in the input file should already exist in the database. The ID Resolver locates the identifiers in the database as described on page 230. If a record does not exist in the database, the ID Resolver is not able to resolve the identifier for this record and it indicates that an error has occurred. The following example is used to locate identifiers for data elements that already exist in the database:

- Windows

  ```
  idresgen -dbname db -dbuser user -dbpwd pwd -infile input.xml
  -outfile output.xml -method update -customizer customizer -schemaname wcsadmin
  ```
- AIX Solaris Linux

  ```
  ./idresgen.sh -dbname db -dbuser user -dbpwd pwd -infile input.xml
  -outfile output.xml -method update -customizer customizer -schemaname wcsadmin
  ```
- 400

  ```
  QWEBCOMM/RESWCSID DATABASE(DATABASE_NAME) SCHEMA(WCSADMIN)
  INSTROOT(/QIBM/UserData/WebCommerce/instances/mser) PASSWD(mypassword)
  INFILE(input.xml) OUTFILE(output.xml) METHOD(*UPD)
  ```

Note: Refer to "Using the Loader package commands and scripts" on page 249 for the location of the appropriate ID Resolve command or script.

Mixed method:
If the input data file contains records that already exist in the database as well as some records that are new, the ID Resolver must be run using the mixed method. With this method, the ID Resolver creates new identifiers for records only if the records do not exist in the database. Otherwise, the existing identifier is obtained from the database. The following example is used to generate identifiers for new data and to locate identifiers for data elements that already exist in the database:

- Windows

  ```
  idresgen -dbname db -dbuser user -dbpwd pwd -infile input.xml
  -outfile output.xml -method mixed -customizer customizer -schemaname wcsadmin
  ```

- AIX Solaris Linux

  ```
  ./idresgen.sh -dbname db -dbuser user -dbpwd pwd -infile input.xml
  -outfile output.xml -method mixed -customizer customizer -schemaname wcsadmin
  ```

- 400

  ```
  QWEBCOMM/RESWCSID DATABASE(DATABASE_NAME) SCHEMA(WCSADMIN)
  INSTROOT(/QIBM/UserData/WebCommerce/instances/mser) PASSWD(mypassword)
  INFILE(input.xml) OUTFILE(output.xml) METHOD(*MIX)
  ```

Notes:

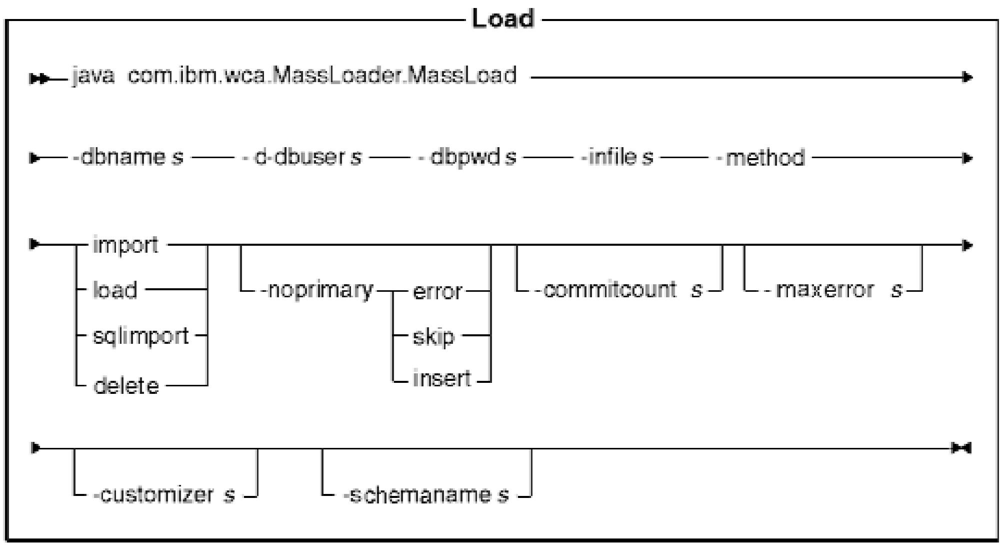
1. Refer to "Using the Loader package commands and scripts" on page 249 for the location of the appropriate ID Resolve command or script.
2. Mixed method is the recommended method for Store Services.

For detailed information on setting up and customizing the files used to run this command, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide.*

Load command

This command loads an XML input file into a target database.

Windows AIX Solaris Linux



Notes:

1. The above diagram is intended primarily as a reference for the command parameters. The command file or script provided for this command and listed under "Using the Loader package commands and scripts" on page 249 acts as a wrapper to the actual Java command and accepts the same parameters; therefore, it is recommended that you use the command file or script rather than invoke the Java command directly.
2. Filenames specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

-dbname
Name of the target database

-dbuser
Name of the user connecting to the database

-dbpwd
Password for the user connecting to the database

-infile Name of the input XML file

-method
Mode of operation for the Loader to use when inserting data into the database

- The load method uses the native loader from the database vendor. You can use the load method for both local and remote Oracle databases; but the load method can only be used for local DB2 databases.
- Although the import method can be used to load data into local or remote databases, it is usually used to load data into remote DB2 databases. This method uses the import or update option if it is available from the database vendor. If you specify this method for a database in which the import or update option is not available, such as Oracle, SQL statements using JDBC are used to update the database.
- The SQL import (sqlimport) method can be used with both local and remote databases.
- The delete method deletes data from the database.

-noprimary
Action the Loader must take when the primary key is missing for a record in the input file

- The error option indicates that it should report the missing primary key as an error and terminate.
- The skip option skips any record in the input file that does not have a primary key.
- The insert option tries to insert or delete the data.

This parameter is optional. The default action is error.

-commitcount
Number of records processed before the database commit occurs when using the SQL import method of operation. This parameter is optional. The default number is 1.

-maxerror
Number of errors after which the Loader will terminate in the SQL import method of operation. This parameter is optional.

-customizer
Name of the customizer property file to be used. This parameter is optional. The customizer property file sets the way that the Loader functions. `MassLoadCustomizer.properties` is the default file. The customizer property file can be specified as shown in the following example:

```
-customizer d:\WebSphere\CommerceServer\prop\ml.properties
```

If this file exists in a directory specified in the classpath system-environment variable, the same file can be specified as shown in the following example:

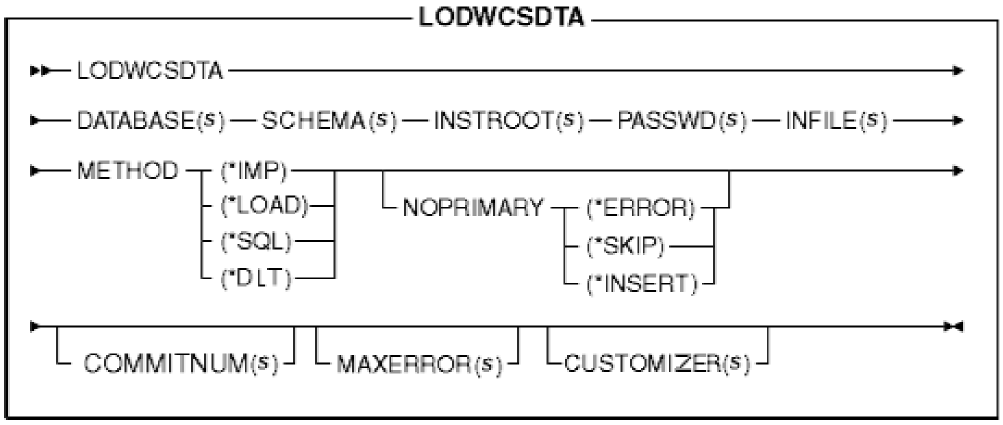
```
-customizer ml
```

For more information on creating and specifying a new customizer property file, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

-schemaname
Name of the target database schema. This parameter is optional.

If this parameter is not specified when running the command, the command looks for a name=value pair in the customizer property file that specifies the value of SchemaName. If this pair is present in the property file, the command uses the value specified. If neither a command-line nor a property-file specification for this parameter exists, the command defaults to the schema name of the table in the database.

400



Note: Filenames specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

DATABASE
Name of the target database as displayed in the relational database directory

SCHEMA
Name of the target database schema; this is the same as the instance name

INSTROOT
: Full name of the WebSphere Commerce instance root path, such as `/QIBM/UserData/WebCommerce/instances/`*instance_name*

PASSWD
: Password for the WebSphere Commerce instance

INFILE
: Name of the input XML file

METHOD
: Mode of operation for the Loader to use when inserting data into the database

- The load method (*LOAD) uses the native loader from the database vendor. You can use the load method (*LOAD) for both local and remote Oracle databases; but the load method (*LOAD) can only be used for local DB2 databases.
- Although the import (*IMP) method can be used to load data into local or remote databases, it is usually used to load data into remote DB2 databases. This method uses the import or update option if it is available from the database vendor. If the import or update option is not available, SQL statements using JDBC are used to update the database.
- The SQL import (*SQL) method can be used with both local and remote databases.
- The delete (*DLT) method deletes data from the database.

NOPRIMARY
: Action that the Loader must take when the primary key is missing for a record in the input file

- The error option (*ERROR) indicates that it should report the missing primary key as an error and terminate.
- The skip option (*SKIP) skips any record in the input file that does not have a primary key.
- The insert option (*INSERT) tries to process (insert or delete) the data.

This parameter is optional. The default action is error.

COMMITNUM
: Number of records processed before the database commit occurs when using the SQL import method of operation. This parameter is optional. The default number is 1.

MAXERROR
: Number of errors after which the Loader will terminate in the SQL import method of operation. This parameter is optional.

CUSTOMIZER
: Name of the customizer property file to be used. This parameter is optional. The customizer property file sets the way that the ID Resolver functions. The default file is `ISeries_LODWCSDTA_Customizer.properties`. The customizer property file can be specified as shown in the following example:

```
CUSTOMIZER(/wc/prop/ml.properties)
```

If this file exists in a directory specified in the classpath system-environment variable, the same file can be specified as shown in the following example:

```
CUSTOMIZER(ml)
```

For more information on creating and specifying a new customizer property file, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

Methods for the Load command:
Before loading data, you should determine which of the three methods of processing will produce the best results.

Load method:
Consider the load method in any of the following situations:

- The source data is clean, and the database does not contain any data

  Note: Clean data is data that does not violate any of the constraints of the tables into which it is being loaded.
- The source data is clean, and the database does not contain the data that is being loaded
- The source data is clean, one or more of the targeted tables do not contain primary keys, and the database does not contain the data that is being loaded
- The database is a local DB2 database
- The database is a local or remote Oracle database
- The database is not being accessed by other users or applications while the load is taking place

400 With the load method, data is loaded into the database. If the data already exists, the command fails as a result of a duplicate-key error and a duplicate-error message displays.

The following restrictions exist on using the load method:

- The load method cannot insert or update data in bit data fields.
- DB2 With the load method, only new records are inserted into the database; existing records are not updated.
- DB2 The load method can only be used for local, not remote, DB2 databases.

Import method:
Windows AIX Solaris Linux With the import method for DB2, data is also loaded into the database. If the data already exists, it is not deleted but is updated with new values. Consider this method in any of the following situations:

- The database management system is DB2
- You do not know whether or not the data is clean
- You have to update large sets of homogeneous data at a column level
- All of the tables into which data is being imported have primary keys

400 With the import method, data is also loaded into the database. If the data already exists, it is not deleted but it is updated with new values. Consider this method in any of the following situations:

- You do not know whether or not the data is clean
- The data already exists in the database
- All of the tables into which data is being imported have primary keys

The following restrictions exist on using the import method:

- The database management system must be DB2 in order to use the import method.
- The import method cannot insert or update data in bit data fields.
- With the import method, the Loader only inserts or updates tables that have primary keys defined on them; the import method cannot insert or update data in tables that do not have a primary key. If the input record only has values for columns that are primary, the record is rejected.

SQL import method:
With the SQL import method, JDBC or SQL statements are used to update or insert data into the database. Data is inserted if it does not already exist, and existing data is updated. Consider this method in any of the following situations:
- You are updating existing data and require column-level updates
- Some of the data is not clean
- The database is not local

Note: If you are using Product Advisor search-space synchronization, you must use the SQL import method for loading data.

Delete method:
The delete method is used to delete data that is in the input XML document from the database. The element must contain the values for the primary key or the unique index for the table. If the data being deleted has data in another table that is dependent on it with ″cascade on delete″ enabled, the dependent data is also deleted.

Comparing the methods:

- **Comparison of the SQL import and load methods**

  The SQL import method checks for data consistency, including foreign references, and allows you to update existing data. The load method does not.
- **Comparison of the import and SQL import methods**

  The import and SQL import methods perform similar functions. The import method is typically faster, but it requires disk space for temporary files.

  The import method can only insert or update tables that have primary keys defined in them; whereas, the SQL import method does not require that tables have primary keys in them.
- **Comparison of methods based on database product used**

  The import and load methods use native utilities that are optimized for DB2, while the SQL import method uses JDBC calls (which are generic to many database products).

Performance considerations:
When using the Loader to load large documents into a database, consider the following items:

- **Java Virtual Machine (JVM) heap size**

  By default, the maximum amount of memory allocated to the JVM heap is 64 MB. If this is not increased, the JVM can eventually run out of memory during the load process. The maximum amount of memory allocated to the Java heap can be varied by using the JVM -mx option in the Java command.
- **Trace logging**

  The trace logger can exhaust the JVM heap when loading a large XML document. Trace information is used mostly for debugging a run if the run fails. If tracing the load process is not necessary, the trace should be turned off. There is a significant performance gain when the trace is turned off. The trace is turned off by modifying the logging configuration XML document. For information on modifying the logging configuration XML document, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.
- **Commit count**

  The default commit count for the Loader when it is operating in SQL import mode is 1. By default, therefore, transactions are committed for every update or insertion into the database. To improve the performance of the Loader for large documents, the commit count should be increased. A value of "100" is suggested; but it can be higher depending on the amount of physical memory on the server, the DBMS transaction log size, and so forth.

  The commit count for the Loader is changed using the -commitcount *count* option for the Load command (where *count* is the number of statements executed before the transaction is committed).
- **Logging configuration**

  Unusually slow progress when loading data could result from one of the following situations:
  - The user invoking the Loader does not have permission to write to the directory or to update the file specified in the logging configuration document.
  - The directory specified as the location of the file in the logging configuration document does not exist.
  - The drive specified as the location of the file in the logging configuration document does not have enough space.

  When you correct any of these problems, you may need to change the specified location of the file by modifying the logging configuration document (`WCALoggerConfig.xml` by default). For information on modifying the logging configuration XML document, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

For detailed information on setting up and customizing the files used to run this command, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

Loader package commands for transforming and extracting data

DTD Generate command

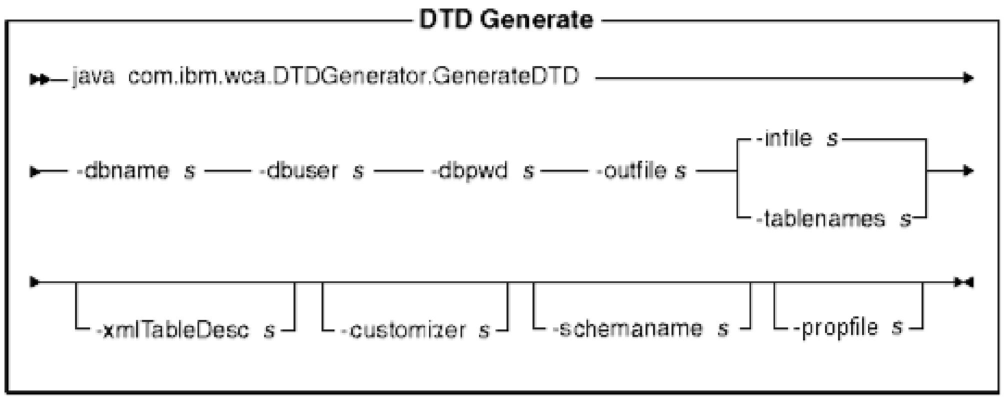
This command creates a DTD for use with the Loader package. This DTD is used throughout the data loading process. Depending on how you invoke the command, the DTD Generator can generate a DTD alone or a DTD along with an XML schema.

The DTD Generator can create a DTD based on the WebSphere Commerce database schema. If you use the DTDs provided with the sample store archives and you do not modify the database schema, you do not need to generate a DTD using the DTD Generator. The DTDs that are provided 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 Linux `/opt/WebSphere/CommerceServer/xml/sar`
- 400 `/QIBM/ProdData/WebCommerce/xml/sar`

It is recommended that you use the DTDs provided. If you customize a database schema, however, you must either edit the DTD provided to match your changes or create a new DTD.

Windows AIX Solaris Linux



Notes:

1. The above diagram is intended primarily as a reference for the command parameters. The command file or script provided for this command and listed under "Using the Loader package commands and scripts" on page 249 acts as a wrapper to the actual Java command and accepts the same parameters; therefore, it is recommended that you use the command file or script rather than invoke the Java command directly.
2. Filenames specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

-dbname
: Name of the target database

-dbuser
: Name of the user connecting to the database

-dbpwd
: Password for the user connecting to the database

-outfile
: Name of the output DTD file (preferably with a `.dtd` extension)

-infile
: Name of an input file containing a database-table name on each line

-tablenames
: Names of tables separated by commas and enclosed in quotation marks ("")

-xmltabledesc
: File path of the XML schema file to be created. This parameter is optional.

-customizer
: Name of the customizer property file to be used. This parameter is optional. The customizer property file sets the way that the DTD Generator functions. `DB2ConnectionCustomizer.properties` is the default file. The customizer property file can be specified as shown in the following example:

```
-customizer d:\WebSphere\CommerceServer\prop\dtdgen.properties
```

If this file exists in a directory specified in the classpath system-environment variable, the same file can be specified as shown in the following example:

```
-customizer dtdgen
```

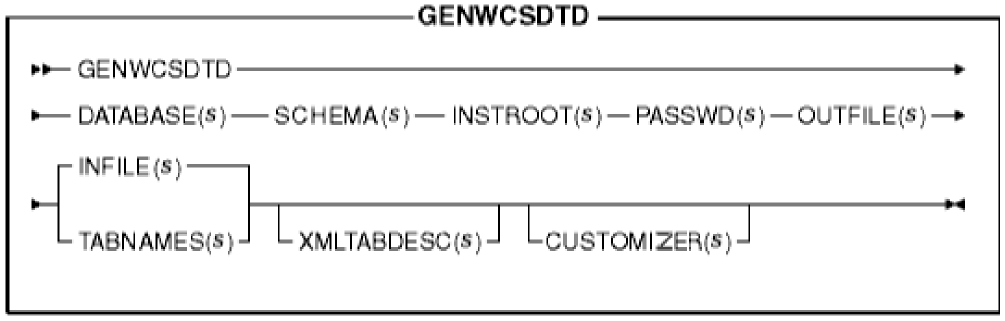
For more information on creating and specifying a new customizer property file, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

-schemaname
: Name of the target database schema. This parameter is optional.

If this parameter is not specified when running the command, the command looks for a name=value pair in the customizer property file that specifies the value of SchemaName. If this pair is present in the property file, the command uses the value specified. If neither a command-line nor a property-file specification for this parameter exists, the command defaults to the name of the database user.

-propfile
: Properties file where help text, default values, and field-description information can be stored for a Web editor form description. This parameter is optional.

400



Note: Filenames specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

DATABASE
: Name of the target database, as displayed in the relational database directory

SCHEMA
: Name of the target database schema; this is the same as the instance name

INSTROOT
: Full name of the WebSphere Commerce instance root path, such as `/QIBM/UserData/WebCommerce/instances/`*instance_name*

PASSWD
: Password for the WebSphere Commerce instance

OUTFILE
: Name of the output DTD file (preferably with a `.dtd` extension)

INFILE
: Name of an input file containing a database-table name on each line

TABNAMES
: Names of tables separated by commas and enclosed in quotation marks ("")

XMLTABDESC
: File path of the XML schema file to be created. This parameter is optional.

CUSTOMIZER
: Name of the customizer property file to be used. This parameter is optional. The customizer property file sets the way that the DTD Generator functions. The default file is `ISeries_GENWCSDTD_Customizer.properties`. The customizer property file can be specified as shown in the following example:

```
CUSTOMIZER(/wc/prop/dtdgen.properties)
```

If this file exists in a directory specified in the classpath system-environment variable, the same file can be specified as shown in the following example:

```
CUSTOMIZER(dtdgen)
```

For more information on creating and specifying a new customizer property file, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

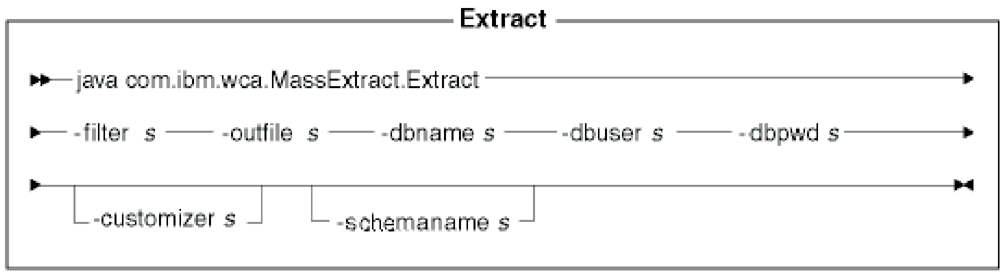
For detailed information on setting up and customizing the files used to run this command, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

Extract command

This command extracts a selected subset of data from a database in the form of an XML file.

To extract data from a database using the Extractor, you must specify the data that you want to extract using an extraction-filter file. The extraction filter that you use depends on the type of data that you want to extract.

Windows AIX Solaris Linux



Notes:

1. The above diagram is intended primarily as a reference for the command parameters. The command file or script provided for this command and listed under "Using the Loader package commands and scripts" on page 249 acts as a wrapper to the actual Java command and accepts the same parameters; therefore, it is recommended that you use the command file or script rather than invoke the Java command directly.
2. Filenames specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

-filter Name of the extraction-filter file

-outfile
Name of the output XML file where the extracted data will be stored

-dbname
Name of the database from which data is being extracted

-dbuser
Database user name for the database from which data is being extracted

-dbpwd
Password associated with the user name for the database from which data is being extracted

-customizer
Name of the customizer property file to be used. The customizer property file sets the way that the Extractor functions.

DB2ConnectionCustomizer.properties is the default file. The customizer property file can be specified as shown in the following example:

```
-customizer d:\WebSphere\CommerceServer\prop\extract.properties
```

If this file exists in a directory specified in the classpath system-environment variable, the same file can be specified as shown in the following example:

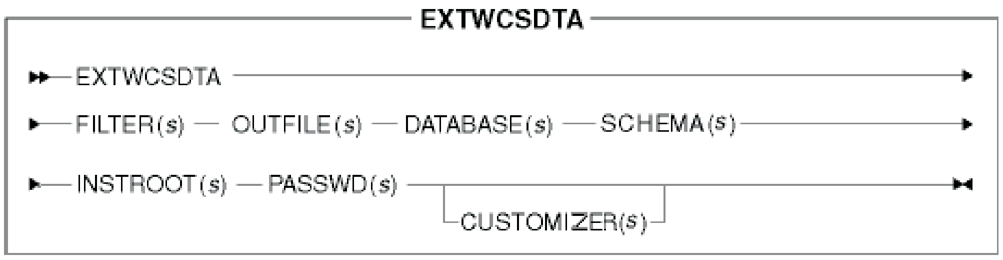
```
-customizer extract
```

For more information on creating and specifying a new customizer property file, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

-schemaname
: Name of the database schema from which data is being extracted. This parameter is optional.

If this parameter is not specified when running the command, the command looks for a name=value pair in the customizer property file that specifies the value of SchemaName. If this pair is present in the property file, the command uses the value specified. If neither a command-line nor a property-file specification for this parameter exists, the command defaults to the schema name of the table in the database.

400



Note: Filenames specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

FILTER
: Name of the extraction-filter file

OUTFILE
: Name of the output XML file where the extracted data will be stored

DATABASE
: Name of the database from which data is being extracted as displayed in the relational database directory

SCHEMA
: Name of the database schema from which data is being extracted; this is the same as the instance name

INSTROOT
: Full name of the WebSphere Commerce instance root path, such as `/QIBM/UserData/WebCommerce/instances/`*instance_name*

PASSWD
Password for the WebSphere Commerce instance

CUSTOMIZER
Name of the customizer property file to be used. The customizer property file sets the way that the Extractor functions. The default file is `ISeries_EXTWCSDTA_Customizer.properties`. The customizer property file can be specified as shown in the following example:

```
CUSTOMIZER(/wc/prop/extract.properties)
```

If this file exists in a directory specified in the classpath system-environment variable, the same file can be specified as shown in the following example:

```
CUSTOMIZER(extract)
```

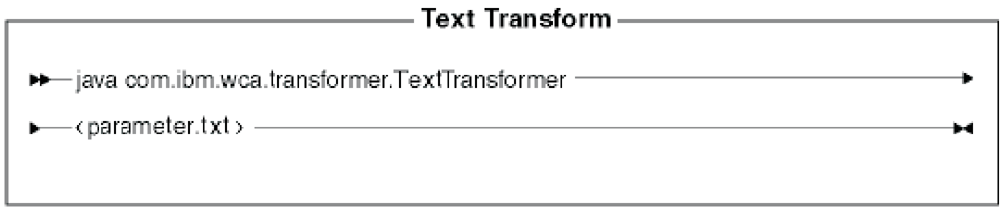
For more information on creating and specifying a new customizer property file, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

For more information on this command, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

Text Transform command

This command transforms data between a character-delimited variable format and an XML format.

Windows AIX Solaris Linux



Note: The above diagram is intended primarily as a reference for the command parameters. The command file or script provided for this command and listed under "Using the Loader package commands and scripts" on page 249 acts as a wrapper to the actual Java command and accepts the same parameters; therefore, it is recommended that you use the command file or script rather than invoke the Java command directly.

Parameter values:
The following values are specified and separated by commas in a parameter file (*parameter.txt*):

input file
Name of the file to be transformed

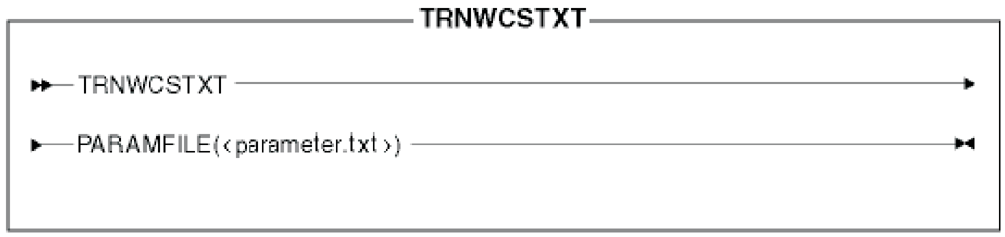
schema file
Name of the XML schema file to be used in the transformation

output file
Name of the output file in which the transformed data will be stored

transformation method
: Method to be used in adding the data to the output file. Specify **Create** if a new file is to be created; or specify **Append** if the output data is to be appended to an existing data file.

This file is also referred to as a "manifest" or "command" file. It can contain multiple lines of four parameters each.

400



Parameter values:
The following values are specified and separated by commas in a parameter file (*parameter.txt*):

input file
: Name of the file to be transformed

schema file
: Name of the XML schema file to be used in the transformation

output file
: Name of the output file in which the transformed data will be stored

transformation method
: Method to be used in adding the data to the output file. Specify **Create** if a new file is to be created; or specify **Append** if the output data is to be appended to an existing data file.

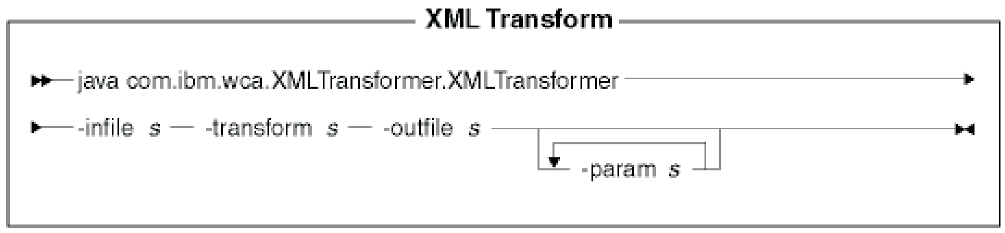
Note: This file is also referred to as a "manifest" or "command" file.

For more information on this command, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

XML Transform command

This command converts an XML file into an alternate XML format.

Windows AIX Solaris Linux



Notes:

1. The above diagram is intended primarily as a reference for the command parameters. The command file or script provided for this command and listed

under "Using the Loader package commands and scripts" on page 249 acts as a wrapper to the actual Java command and accepts the same parameters; therefore, it is recommended that you use the command file or script rather than invoke the Java command directly.

2. Filenames specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

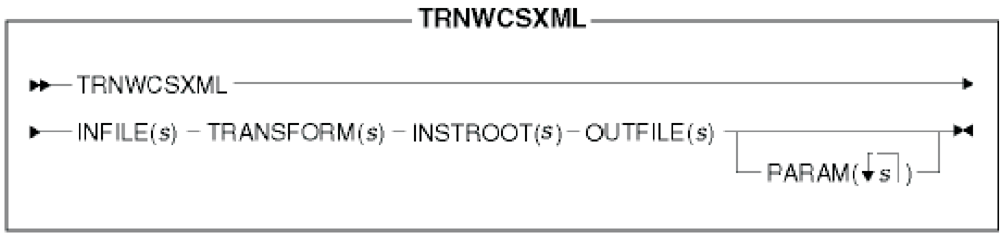
-infile Name of the file to be transformed

-transform
Name of the transform XSL mapping rule file

-outfile
Name of the output XML file in which the transformed data will be stored

-param
Parameter to be passed to the XSL mapping rule file. ***This parameter is optional.*** This parameter can be specified multiple times to pass multiple name=value pairs.

400



Note: Filenames specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

INFILE
Name of the file to be transformed

TRANSFORM
Name of the transform XSL mapping rule file

INSTROOT
Full name of the WebSphere Commerce instance root path, such as `/QIBM/UserData/WebCommerce/instances/`*instance_name*

OUTFILE
Name of the output XML file in which the transformed data will be stored

PARAM
Parameter to be passed to the XSL mapping rule file. ***This parameter is optional.*** The string can contain multiple values to pass multiple name=value pairs.

For more information on this command, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

Tools related to the Loader package commands

Text Transformation tool

The Text Transformation tool helps you to process a transformation of data between a character-delimited variable format and an XML format using the Text Transform command. The following views are provided:

1. The Text Schema Edit View allows you to create and modify the XML schema file to be used in a transformation.
2. The Transformation Command Edit View allows you to create and modify the actual commands used to run the transformation process.
3. The Transformation Command Process View allows you to launch the transformation process.

For more information on this tool, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

XSL editor

The XML Transformer uses XSL to define the rules for transforming an XML file into another XML file. The mapping function in the XSL editor gives you a visual interface with which you can establish the association from an element in a source DTD to an element in a target DTD. Given two DTDs, you can develop XSL rules that determine how an XML file that conforms to the first (source) DTD is transformed into a file that conforms to the second (target) DTD.

For more information on this tool, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

Web editor

The Web editor enables you to create, delete, and change your catalog data through a Web browser. Data-entry forms for viewing and updating information are central to the Web editor. In the simplest case, the forms correspond to tables in the WebSphere Commerce Server database. The Store Developer can choose to use the default forms provided or to customize the available forms.

For more information on this tool, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

Loading store data

This section provides examples of how to load store data into your WebSphere Commerce Server database using the Loader package command-line utilities.

Notes:

1. The examples in this section are performed in a Windows NT environment. For information on running these commands in other environments, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.
2. Although the Loader package command-line utilities support DB2, DB2 for iSeries, and Oracle databases, only the commands and options for DB2 are included in the following examples. If you are using a database other than DB2, make sure that you modify your customizer properties files as described in the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User's Guide*.

Refer to Chapter 28, "Loading WebSphere Commerce database asset groups" on page 259 and "Loading database asset groups" on page 267 for information on the loading process for WebSphere Commerce database asset groups.

Using the Loader package commands and scripts

To run the Loader package commands, use the scripts or commands provided in the following WebSphere Commerce directory:

- NT `drive:\WebSphere\CommerceServer\bin`
- 2000 `drive:\Program Files\WebSphere\CommerceServer\bin`
- AIX `/usr/WebSphere/CommerceServer/bin`
- Solaris Linux `/opt/WebSphere/CommerceServer/bin`
- 400 QWEBCOMM native library

The scripts and commands are as follows:

- Windows

dtdgen.cmd	DTD Generate command
idresgen.cmd	ID Resolve command
massload.cmd	Load command
massextract.cmd	Extract command
txttransform.cmd	Text Transform command
xmltransform.cmd	XML Transform command

- AIX Solaris Linux

dtdgen.sh	DTD Generate shell script
idresgen.sh	ID Resolve shell script
massload.sh	Load shell script
massextract.sh	Extract shell script
txttransform.sh	Text Transform shell script
xmltransform.sh	XML Transform shell script

- 400

GENWCSDTD	DTD Generate command
RESWCSID	ID Resolve command
LODWCSDTA	Load command
EXTWCSDTA	Extract command
TRNWCSTXT	Text Transform command
TRNWCSXML	XML Transform command

Examples of resolving identifiers

The examples of identifier resolution described in this section use the store-asset files from the Business ToolTech sample store.

Because this example is based on loading new data into the WebSphere Commerce Server database, we will use the load method.

If you later need to modify certain elements within the XML document, you can do so using the update method. The update method should run faster than the load method because no new identifiers are allocated with the update method. With the update method, a database query is performed to locate the identifier and an error is reported if the identifier is not found. Refer to the discussion beginning on page 230 for more information on how this process works.

If your input XML file contains elements that already exist in the database as well as elements that do not, use the mixed method. With the mixed method, a database lookup is done first and an identifier is assigned to the element if the record is not found. When in doubt, use the the mixed method. Although the load and update methods provide faster performance than the mixed method, the resolved XML file produced by using the mixed method has a greater likelihood of loading without errors.

For a discussion of how the ID Resolver works, refer to "Methods for the ID Resolve command" on page 231.

Resolving identifiers in XML files with internal aliases

To resolve identifiers using internal aliases before loading the data into your WebSphere Commerce Server database, run the ID Resolve command as shown in the following example.

Note: This example assumes that WebSphere Commerce is located on drive `c`. If WebSphere Commerce is located on a different drive on your system, substitute the letter of that drive for `c` in the examples shown in the remainder of this chapter.

1. Create a working directory.

   For this example, create the following directory:

   ```
   c:\WebSphere\CommerceServer\runtime\test
   ```

   Note: If you do not use `c:\WebSphere\CommerceServer\runtime\test` as your working directory, substitute the name and path of the working directory that you do use for `c:\WebSphere\CommerceServer\runtime\test` in the examples shown in the remainder of this chapter.

2. Make sure that your input XML file as well as any referenced DTD files are in a location where the ID Resolver can find them.

   For this example, do the following:

   a. From the Windows command prompt, enter the following command:

      ```
      copy c:\WebSphere\CommerceServer\samplestores\ToolTech\ToolTech
      _en_US_fr_FR.sar c:\WebSphere\CommerceServer\runtime\test
      ```

      This copies the `ToolTech_ en_US_fr_FR.sar` file into `c:\WebSphere\CommerceServer\runtime\test`.

   b. From a Windows command prompt, enter the following command:

      ```
      cd c:\WebSphere\CommerceServer\runtime\test
      ```

c. Do one of the following:
   - If you have Java installed, enter the following command from the Windows command prompt:

     ```
     jar -xvf ToolTech_en_US_fr_FR.sar
     ```
   - Use any up-to-date unzipper product (such as WinZip or PKZIP) to extract the entire contents of `c:\WebSphere\CommerceServer\samplestores\ToolTech\ToolTech_en_US_fr_FR.sar` into `c:\WebSphere\CommerceServer\runtime\test`.

   This extracts the Business ToolTech sample store XML files into `c:\WebSphere\CommerceServer\runtime\test\data`.

d. From the Windows command prompt, enter the following command:

   ```
   copy c:\WebSphere\CommerceServer\xml\sar\store.dtd
   c:\WebSphere\CommerceServer\runtime\test\data
   ```

   This copies the `store.dtd` file into `c:\WebSphere\CommerceServer\runtime\test\data`.

e. From the Windows command prompt, enter the following command:

   ```
   copy c:\WebSphere\CommerceServer\xml\sar\DBLoadMacros.dtd
   c:\WebSphere\CommerceServer\runtime\test\data
   ```

   This copies the `DBLoadMacros.dtd` file into `c:\WebSphere\CommerceServer\runtime\test\data`.

f. From the Windows command prompt, enter the following command:

   ```
   copy c:\WebSphere\CommerceServer\xml\sar\fulfillment.dtd
   c:\WebSphere\CommerceServer\runtime\test\data
   ```

   This copies the `fulfillment.dtd` file into `c:\WebSphere\CommerceServer\runtime\test\data`.

3. Make sure that the WebSphere Commerce schema is loaded into your database along with the necessary bootstrap data by creating an appropriate WebSphere Commerce Server database instance.

   Note: For information on creating an instance, refer to the WebSphere Commerce installation guide for your operating system.

   The WebSphere Commerce Server database instance that this example uses is called *mall*. Primary and foreign keys will be obtained from the KEYS and SUBKEYS tables of this database; therefore, the ID Resolver will be unable to resolve the identifiers if the database is not loaded properly.

4. In the `store.xml` file, you will find the following element:

   ```
   <storeent
      STOREENT_ID="@storeent_id_1"
      MEMBER_ID="&MEMBER_ID;"
      TYPE="S"
      IDENTIFIER="ToolTech"
      SETCURR="USD"
   />
   ```

   This element of `store.xml` maps to the storeeent table in the database; and its STOREENT_ID MEMBER_ID, TYPE, IDENTIFIER, and SETCURR attributes map to columns in that table. The @storeent_id_1 specification is an internal alias for the value of the STOREENT_ID attribute; and &MEMBER_ID; is an entity parameter. The value of the entity &MEMBER_ID; has to be substituted before it can be loaded using the Loader. The value of &MEMBER_ID; is defined in the DBLoadMacros.dtd macro file; and the value is substituted from that file. When the ID Resolver encounters @storeent_id_1, it looks in its cache of primary tables to see if storeent is present. Because it is a primary table, storeeent is present. The ID Resolver fetches the counter for that table, increments it, and replaces the internal alias with the result. All other such entries in the store.xml file are processed in the same way.

5. Make sure that your path includes the directory containing the appropriate ID Resolve command or script as listed in "Using the Loader package commands and scripts" on page 249.

   For this example, enter the following command from a Windows command prompt:

   ```
   cd c:\WebSphere\CommerceServer\bin
   ```

   where *c:\WebSphere\CommerceServer\bin* should be changed to the name of the directory containing the ID Resolve command idresgen.cmd if it is not located in c:\WebSphere\CommerceServer\bin on your system.

6. From the Windows command prompt, enter the following command:

   ```
   idresgen -dbname mall -dbuser db2admin -dbpwd db2admin
   -infile c:\WebSphere\CommerceServer\runtime\test\data\store.xml
   -outfile c:\WebSphere\CommerceServer\runtime\test\data\store1.xml
   -method load
   ```

   where

   - *mall* should be changed to the name of the target database if you are not using mall
   - the first *db2admin* should be changed to the name of the user connecting to the database if you are not using db2admin
   - the second *db2admin* should be changed to the password of the user connecting to the database if you are not using db2admin

   The first output XML fragment in store1.xml looks like this:

   ```
   <storeent
      STOREENT_ID="10001"
      MEMBER_ID="-2001"
      TYPE="S"
      IDENTIFIER="ToolTech"
      SETCCURR="USD"
   />
   ```

   Note: This is an example. Your output file may contain different values.

   The second XML fragment in store1.xml looks like this:

   ```
   <store
      STORE_ID="10001"
      DIRECTORY="ToolTech"
      FFMCENTER_ID=""
      LANGUAGE_ID="-1"
      STOREGRP_ID="-1"
      ALLOCATIONGOODFOR="43200"
      BOPMPADFACTOR="0"
      DEFAULTBOOFFSET="2592000"
      FFMCSELECTIONFLAGS="0"
      MAXBOOFFSET="7776000"
      REJECTEDORDEXPIRY="259200"
      RTNFFMCTR_ID=""
      PRICEREFFLAGS="0"
      STORETYPE="B2B"
   />
   ```

Note: This is an example. Your output file may contain different values.

The FFMCENTER_ID and the RTNFFMCTR_ID attributes were not resolved. Instead of internal-alias @ffmcenter_id_1 being resolved, it was replaced with empty quotation marks (""). This is an error. You will not be able to load the data properly using the Loader due to a foreign-key constraint violation. The ID Resolver could not resolve this internal alias because the FFMCENTER table had not been processed before using an alias referencing it. To solve this problem, choose one of the following three options:

- Option 1:
  a. Enter the following command to run the ID Resolver against the `fulfillment.xml` file (where the FFMCENTER table is defined):

     ```
     idresgen -dbname mall -dbuser db2admin -dbpwd db2admin
     -infile c:\WebSphere\CommerceServer\runtime\test\data\fulfillment.xml
     -outfile c:\WebSphere\CommerceServer\runtime\test\data\fulfillment1.xml
     -method load
     ```

     where
     - *mall* should be changed to the name of the target database if you are not using mall
     - the first *db2admin* should be changed to the name of the user connecting to the database if you are not using db2admin
     - the second *db2admin* should be changed to the password of the user connecting to the database if you are not using db2admin

     The resolved element in the `fulfillment1.xml` output file looks like this:

     ```
     <fulfillment-asset>
     <ffmcenter
        FFMCENTER_ID="10001"
        MEMBER_ID="-2001"
        NAME="ToolTech Home"
        DEFAULTSHIPOFFSET="0"
        MARKFORDELETE="0"
     />
     </fulfillment-asset>
     ```

     Note: This is an example. Your output file may contain different values.
  b. Get the FFMCENTER_ID key from the resulting output file (`fulfillment1.xml`) and substitute that key for all occurrences of @ffmcenter_id_1 in your working copy of `store.xml` in `c:\WebSphere\CommerceServer\runtime\test\data`.
  c. Enter the following command:

     ```
     idresgen -dbname mall -dbuser db2admin -dbpwd db2admin
     -infile  c:\WebSphere\CommerceServer\runtime\test\data\store.xml
     -outfile c:\WebSphere\CommerceServer\runtime\test\data\store1.xml
     -method load
     ```

     where
     - *mall* should be changed to the name of the target database if you are not using mall
     - the first *db2admin* should be changed to the name of the user connecting to the database if you are not using db2admin
     - the second *db2admin* should be changed to the password of the user connecting to the database if you are not using db2admin

The fully resolved elements in the `store1.xml` output file look like this:

```
<store-asset>
<storeent
    STOREENT_ID="10151"
    MEMBER_ID="-2001"
    TYPE="S"
    IDENTIFIER="ToolTech"
    SETCCURR="USD"
 />
<store
    STORE_ID="10151"
    DIRECTORY="ToolTech"
    FFMCENTER_ID="10001"
    LANGUAGE_ID="-1"
    STOREGRP_ID="-1"
    ALLOCATIONGOODFOR="43200"
    BOPMPADFACTOR="0"
    DEFAULTBOOFFSET="2592000"
    FFMCSELECTIONFLAGS="0"
    MAXBOOFFSET="7776000"
    REJECTEDORDEXPIRY="259200"
    RTNFFMCTR_ID="10001"
    PRICEREFFLAGS="0"
    STORETYPE="B2B"
 />
<vendor
    VENDOR_ID="10001"
    STOREENT_ID="10151"
    VENDORNAME="Tooltech Vendor"
    MARKFORDELETE="0"
 />
<dispentrel
    AUCTIONSTATE="0"
    CATENTRY_ID="0"
    CATENTTYPE_ID="ProductBean"
    DEVICEFMT_ID="-1"
    DISPENTREL_ID="10001"
    MBRGRP_ID="0"
    PAGENAME="CatalogProductDisplay.jsp"
    STOREENT_ID="10151"
    RANK="0"
 />
<dispentrel
   AUCTIONSTATE="0"
    CATENTRY_ID="0"
    CATENTTYPE_ID="ItemBean"
    DEVICEFMT_ID="-1"
    DISPENTREL_ID="10002"
    MBRGRP_ID="0"
    PAGENAME="CatalogItemDisplay.jsp"
    STOREENT_ID="10151"
    RANK="0"
 />
<dispcgprel
    CATGROUP_ID="0"
    DEVICEFMT_ID="-1"
    DISPCGPREL_ID="10001"
    MBRGRP_ID="0"
    PAGENAME="CatalogCategories.jsp"
    STOREENT_ID="10151"
    RANK="0"
 />
<invadjcode
    ADJUSTCODE="PCNT"
    INVADJCODE_ID="10001"
    MARKFORDELETE="0"
    STOREENT_ID="10151"
 />
<invadjcode
    ADJUSTCODE="SPLG"
    INVADJCODE_ID="10002"
    MARKFORDELETE="0"
    STOREENT_ID="10151"
 />
<invadjcode
    ADJUSTCODE="DISC"
    INVADJCODE_ID="10003"
    MARKFORDELETE="0"
    STOREENT_ID="10151"
 />
<rtnreason
    REASONTYPE="C"
    RTNREASON_ID="10001"
    STOREENT_ID="10151"
    MARKFORDELETE="0"
    CODE="WPR"
 />
<rtnreason
    REASONTYPE="B"
    RTNREASON_ID="10002"
    STOREENT_ID="10151"
    MARKFORDELETE="0"
    CODE="DEF"
 />
<rtnreason
    REASONTYPE="M"
    RTNREASON_ID="10003"
    STOREENT_ID="10151"
    MARKFORDELETE="0"
    CODE="ERR"
 />
<rtnreason
    REASONTYPE="M"
    RTNREASON_ID="10004"
    STOREENT_ID="10151"
    MARKFORDELETE="0"
    CODE="WPS"
 />
</store-asset>
```

Note: This is an example. Your output file may contain different values.

- Option 2:
  a. Merge the `fulfillment.xml` and `store.xml` files by adding any content that is unique in `fulfillment.xml` (including the reference to `fulfillment.dtd`) to `store.xml`, making sure that the ffmcenter element shown below precedes the store element.

  ```
  <ffmcenter
     FFMCENTER_ID="@ffmcenter_id_1"
     MEMBER_ID="&MEMBER_ID;"
     NAME="ToolTech Home"
     DEFAULTBOOFFSET="0"
     MARKFORDELETE="0"
  />
  ```

  b. Run the ID Resolver against the merged file.
- Option 3: Load the store-assets data group using the process described in "Loading database asset groups" on page 267.

Specifying a properties file with the ID Resolver

You can modify the way in which the ID Resolver resolves identifiers by using the -propfile parameter. The default properties file is `IdResolveKeys.properties`; but you can modify it or specify your own file when invoking the ID Resolve command.

- Windows AIX Solaris Linux `IdResolveKeys.properties` is located in the following directory:
  - NT *drive*:\WebSphere\CommerceServer\properties
  - 2000 *drive*:\Program Files\WebSphere\CommerceServer\properties
  - AIX /usr/WebSphere/CommerceServer/properties
  - Solaris Linux /opt/WebSphere/CommerceServer/properties

  If you do not place this file in the current directory when you run the ID Resolver, you can place it in a directory defined in the classpath environment variable. You can also specify the full path to this file.

- 400 To change `IdResolveKeys.properties`, copy it from the `/QIBM/ProdData/WebCommerce/properties` directory, save it to the `/`*`instroot`*`/xml` directory, then make any necessary changes to the new file.

  Note: The above directory is in the classpath used by the RESWCSID command.

The property-file specification takes precedence over the use of internal aliases.

Here is a sample XML fragment from the `store.xml` file:

```
<store
   STORE_ID="@storeent_id_1"
   DIRECTORY="ToolTech"
   FFMCENTER_ID="@ffmcenter_id_1"
   LANGUAGE_ID="&en_US;"
   STOREGRP_ID="-1"
   ALLOCATIONGOODFOR="43200"
   BOPMPADFACTORr="0"
   DEFAULTBOOFFSET="2592000"
   FFMCSELECTIONFLAGS="0"
   MAXBOOFFSET="7776000"
   REJECTEDORDEXPIRY="259200"
   RTNFFMCTR_ID="@ffmcenter_id_1"
   PRICEREFFLAGS="0"
   STORETYPE="B2B"
/>
```

If you run the ID Resolver with the -propfile specified as `c:\WebSphere\CommerceServer\runtime\test\data\myPropFile` and the specified file, `myPropFile.properties`, contains the following entries:

```
NAMEDELIMETER=@
SELECTDELIMETER=:
FFMCENTER=@FFMCENTER_ID@MEMBER_ID:10051 -2001
```

the ID Resolver queries the database for the FFMCENTER table with a where clause of 10051 and -2001 when the store element is processed. The index that is returned for this value is then used to resolve the identifier for FFMCENTER_ID.

For more information on using this command, refer to “ID Resolve command” on page 225.

Example of loading data

When you have resolved the identifiers in the XML file if necessary, you are ready to load the data into the WebSphere Commerce Server database.

Note: If you have properly resolved the identifiers in your XML data, your source XML file should not contain any of the following:

- words preceded by an at (@) symbol
- words preceded by an ampersand (&) symbol
- identifiers with empty quotation marks ("")

The presence of any of these is an indication that your XML file is not ready to be loaded.

The example of loading data described in this section uses the `fulfillment1.xml` file that was resolved in "Examples of resolving identifiers" on page 250.

To load data into your WebSphere Commerce Server database, run the Load command as shown in the following example:

1. Create a working directory.

   For this example, use the directory called `c:\WebSphere\CommerceServer\runtime\test\data` that you created in "Examples of resolving identifiers" on page 250.
2. Make sure that your input XML file is in a location where the Loader can find it.

   For this example, make sure that the `fulfillment1.xml` output file that you created in "Examples of resolving identifiers" on page 250 is in `c:\WebSphere\CommerceServer\runtime\test\data`.
3. Make sure that you back up your WebSphere Commerce Server database so that you can restore the database from the backup if an unrecoverable error occurs.

   Note: See the backup and recovery documentation provided with your database product for information on backing up your database.
4. Make sure that your path includes the directory containing the appropriate Load command or script as listed in "Using the Loader package commands and scripts" on page 249.

   For this example, enter the following command from a Windows command prompt:

   ```
   cd c:\WebSphere\CommerceServer\bin
   ```

   where *`c:\WebSphere\CommerceServer\bin`* should be changed to the name of the directory containing the Load command `massload.cmd` if it is not located in `c:\WebSphere\CommerceServer\bin` on your system.
5. Run the Load command against your resolved XML file to load your data into the target database.

   For this example, enter the following command from a Windows command prompt:

   ```
   massload -dbname mall -dbuser db2admin -dbpwd db2admin -infile
   c:\WebSphere\CommerceServer\runtime\test\data\fulfillment1.xml
   -method sqlimport -commitcount 50
   ```

where

- *mall* should be changed to the name of the target database if you are not using mall
- the first *db2admin* should be changed to the name of the user connecting to the database if you are not using db2admin
- the second *db2admin* should be changed to the password of the user connecting to the database if you are not using db2admin

Even though there are less than 50 elements to be loaded, this example specifies a value of 50 for -commitcount. This is for performance reasons. By default, the commit count is 1. Using this default causes a commit operation for each record written to the database. Setting the number to 50 in the above example ensures that database I/O occurs only once if loading is successful and that nothing is written to the database if errors occur. If you have a large amount of data to load, however, it is recommended that you do not set the commit-count value as large as the number of elements for the following reasons:

- A high commit-count value causes high memory consumption.
- When the commit-count value is smaller than the number of elements, at least some data is written to the database. Depending on the value of -maxerror, a smaller value for -commitcount ensures that some data is written to the database before the maximum number of errors is exceeded and the tool terminates. The default value for -maxerror is 1.

The default for the -noprimary option is error so that the tool reports errors and terminates when primary keys are missing.

Because these examples do not load the store assets in the order used by Store Services and described in “Database asset loading sequence” on page 259, the `store1.xml` file created in “Examples of resolving identifiers” on page 250 may violate the integrity constraints of some tables. If you tried to load `store1.xml` without modification using the load method, a constraint violation would cause the database to enter pending state. For simplicity, therefore, this example of using the Load command is based on the resolved version of the `fulfillment.xml` file, whose only foreign key is that of the MEMBER_ID defined in the sample store. This example loads the resolved `fulfillment1.xml` file that was output in “Examples of resolving identifiers” on page 250 and uses the SQL import method. When you are not sure that the contents of your XML file are clean, use the SQL import method as shown in this example with the -commitcount and -maxerror parameters set appropriately so that any database constraint violations are reported without altering the database and jeopardizing database integrity.

When you run this command, a trace text file (`trace.txt`) is created in the execution subdirectory (*c:\WebSphere\CommerceServer\bin* in the above example) by default. If your `WCALoggerConfig.xml` logging configuration file has been altered to place `trace.txt` in a different location, go to that location to inspect the file. For more information on customizing `WCALoggerConfig.xml`, refer to the most recent version of the *IBM WebSphere Commerce 5.4 Catalog Manager User’s Guide*.

The `trace.txt` file contains a listing of the actions performed by the command and their results. If you use the SQL import method with the command as shown in the above example, the end of `trace.txt` will contain an entry indicating the number of records committed.

For more information on using this command, refer to “Load command” on page 232.

Chapter 28. Loading WebSphere Commerce database asset groups

If you do not want to create all the database assets and package them into a store archive file before publication, then you can load database asset groups using the WebSphere Commerce Loader package.

The first part of this chapter explains WebSphere Commerce database asset groups, and how a grouping is determined. The second part describes the loading process for these database asset groups into the WebSphere Commerce database. Before reading this section, you should thoroughly review the information in Chapter 27, "Overview of loading store data" on page 221, which helps you understand what you need to know to load database asset groups with the Loader package.

Database asset groups

Database assets are divided into groups to simplify the creation and loading processes. These *database asset groups* comprise a logically related set of tables. The order in which a database asset group is organized is important to loading, since before loading the relationship between the data, the data must exist.

To load the entire set of database assets for a store, you need to follow the "Database asset loading sequence". To load a single group of database assets, you need to ensure that this group is logically complete. For example, when publishing a store archive, you can choose to omit the catalog database assets, which can then be published at a later time. In this case, any database assets dependent on the catalog (inventory, price lists, and some shipping and taxation data) also remain unpublished. To publish the omitted data, ensure that the catalog database assets are logically complete: that is, the base items, catalog entries, attributes, and so on must be provided. You must also publish the dependent database assets which must be logically complete amongst itself. In other words, each SKU must have the appropriate inventory, price, shipping, and taxes defined. In this case, related catalog data which is logically complete is collectively called the *catalog database asset group*.

WebSphere Commerce database assets described in the previous chapters of this guide can be arranged into groups. A group is a logically complete set of data, which can be loaded individually. Each database asset group consists of WebSphere Commerce database tables and has external dependencies as described in Appendix E, "Database asset groups" on page 317. The table list is based on the WebSphere Commerce sample stores, however the list is applicable to any generic store. Remember that the list of tables for each database asset group is not exhaustive, but provided as a general guideline. You may need to include or exclude some tables depending on your store's specific needs.

Database asset loading sequence

There is a certain order to follow to successfully load database asset groups. Each group is considered structurally complete and independent from the other database asset groups. There are, however, foreign key relationships within a database asset group. Such relationships (with the data from other groups) are called the *external dependencies* of a database asset group.

The external dependencies of a database asset group must be met before loading the group into the WebSphere Commerce database. Any group defined as the external dependency of a given database asset group must be loaded first. You can find the list of external dependencies and related tables in "Database asset groups dependencies" on page 317.

Note: A WebSphere Commerce store requires a store owner. You can use the default organization, available as the default owner. ▶Business To load this group, create a new organization instead of using the default one.

Load the database asset groups in the following order:

1. Database asset groups dependent on bootstrap data only.
   a. Load the **organization** database assets first.
2. Database asset group dependent on fulfillment owner.
   a. **Fulfillment** database assets. Except for the organization database asset group, several other database asset groups have a direct or indirect external dependency on the data defined in this group.
3. Database asset groups dependent on store owner organization.
   a. **Access control** database assets are dependent on the store owner organization (`ORGENTITY_ID`). None of the other database asset groups have a dependency on the data defined in this group, which means that access control database assets can be loaded at any time. However, the access control owner must be the same as the store owner.
   b. **Store** database assets are dependent on the store owner organization (`ORGENTITY_ID`).

   The store can refer to a fulfillment center. The store owner organization can also be the fulfillment center's owner organization.
4. Database asset groups dependent on the store database assets. The following groups can be loaded in any order:
   a. **Campaign** database assets.
   b. **Command** database assets.
   c. **Currency** database assets.
   d. **Policy** database assets.
   e. **Shipping** database assets.
   f. **Tax** database assets.
5. Other database asset groups.
   a. **Catalog** database assets are dependent on the shipping and tax database asset groups.
   b. **Store default** database assets have external dependencies on the shipping database asset group. If shipping database assets do not exist, then this group does not need to be populated.
   c. **Contract** database assets are dependent on the organization assets. The contract database assets are not loaded directly. Refer to "Publishing contract assets" on page 275 for more information. You should load the contract assets after the other database asset groups.

Refer to Appendix E, "Database asset groups" on page 317 to see the contents of the database asset groups as formed by the WebSphere Commerce sample stores.

Loading a store

To assist you in loading database assets, sample packages are available from the WebSphere Commerce Web site. These packages are based on the WebSphere Commerce sample stores and contain the files found in the steps below. You can download the packages from:

- Business `http://www.ibm.com/software/webservers/commerce/wc_be/downloads.html`
- Professional `http://www.ibm.com/software/webservers/commerce/wc_pe/downloads.html`

To load XML data for an entire store into the WebSphere Commerce database, do the following:

1. Review the following information:
   a. Appendix B, "Creating your data" on page 305
   b. Appendix E, "Database asset groups" on page 317, as you need to know which WebSphere Commerce database asset files and database tables are affected.
   c. Chapter 27, "Overview of loading store data" on page 221, which provides the background information for the Loader package.
2. Plan your loading process for a complete set of store database assets. Whether you want to load a single database asset group as instructed in "Loading database asset groups" on page 267 or an entire store, the basic process remains the same. In the next steps, you will use or create the following files for your loading process:
   a. one or more database asset files for each group. When you load the complete store, you need all of your created database asset files. For example, you will need a *database asset*`.xml` file (as in `campaign.xml`, `catalog.xml` or `currency.xml`), and separate, locale specific *database asset*`.xml` files for locale your store supports. Examples of such files are shipped with the WebSphere Commerce sample stores in the following directory:
      - NT *drive*`:\WebSphere\CommerceServer\samplestores\`*sample store name*`\data\`
      - 2000 *drive*`:\Program Files\WebSphere\CommerceServer\samplestores\`*sample store name*`\data\`
      - AIX `/usr/WebSphere/CommerceServer/samplestores/`*sample store name*`/data/`
      - Solaris Linux `/opt/WebSphere/CommerceServer/samplestores/`*sample store name*`/data/`
      - 400 `/QIBM/ProdData/WebCommerce/samplestores/`*sample store name*`/data/`

      Note that not all database asset groups require locale specific information.
   b. a new XML file which consolidates all the XML database asset files for your store, contains the XML entity references, and contains the root element for the entire store. This is referred to as the *main database asset group XML file*. You can find this file in the sample package, called `store-data-assets.xml`.

c. a new DTD file which defines all the data types required by the XML files from a database asset group, referred to as the *main database asset group DTD file*. You can find this file in the sample package, called `store-data-assets.dtd`.
d. a second DTD file which defines the external dependencies. You may need to include this file in the *main database asset group DTD file*. You can find this file in the sample package, called `ForeignKeys.dtd`.
e. a third DTD file containing the definition of all WebSphere Commerce tables. The `wcs.dtd` file already exists in WebSphere Commerce, located in the following directory:
   - NT `*drive*:\WebSphere\CommerceServer\schema\xml\`
   - 2000 `*drive*:\Program Files\WebSphere\CommerceServer\schema\xml\`
   - AIX `/usr/WebSphere/CommerceServer/schema/xml/`
   - Solaris Linux `/opt/WebSphere/CommerceServer/schema/xml/`
   - 400 `/QIBM/ProdData/WebCommerce/schema/xml/`

   You may need to include this file in the *main database asset group DTD file*. If you have not customized the WebSphere Commerce schema, then you can use this file without modification.
3. Create the database asset XML files as instructed in previous chapters from this guide. If you completed the tasks in the asset chapters, then these XML files already exist. The database asset files should not contain any DTD declarations or page directives at the start of the file since this may cause conflicts when the files are concatenated. Also, for simplicity you may decide not to create any root elements. The only file that must have a root element is the main database asset group XML file.

   Note: If you have database asset files for more than one language, then each file must begin with `<?xml encoding =` *`locale specific encoding`*`>`. For example, English database asset files should specify `<?xml encoding = "UTF-8"?>`, but French files should specify `<?xml encoding = "ISO-8859-1"?>`.
4. Create the main database asset group XML file for the entire set of store data. This file contains reference entities to include various database asset XML files for your store. By using external reference entities, you can concatenate the XML files to simplify the ID Resolve command and the load process. Also, internal aliases used within each XML file can be external to another XML database asset file within a group or across other groups when loading more than one group at a time. An XML parser would substitute the contents of the file referenced by the external reference entity in place of the external reference.

   Using the following example for loading the entire set of store data as your guide, you can create your database asset group file based on this extract:

```
<?xml version="1.0"?>
<!DOCTYPE import SYSTEM "all-store-assets.dtd">
<import>
<!Fulfillment data group -->
&fulfillment.xml;

<!-- Store data group -->
&store.xml;
&en_US_store.xml;
&fr_FR_store.xml;
```

```
<!-- Tax data group -->
&tax.xml;
&en_US_tax.xml;
&fr_FR_tax.xml;
&taxfulfill.xml;

<!-- Shipping data group -->
&shipping.xml;
&en_US_shipping.xml;
&fr_FR_shipping.xml;
&shipfulfill.xml;

<!-- Catalog data group -->
&catalog.xml;
&en_US_catalog.xml;
&fr_FR_catalog.xml;
&storecatalog.xml;
&storefulfill.xml;
&offering.xml;
&store-catalog-tax.xml;
&store-catalog-shipping.xml;

<!-- Currency data group -->
&currency.xml;
&en_US_currency.xml;
&fr_FR_currency.xml;

<!-- Campaign data group -->
&campaign.xml;
&en_US_campaign.xml;
&fr_FR_campaign.xml;

<!-- Business policy data group -->
&businesspolicy.xml;
&en_US_businesspolicy.xml;
&fr_FR_businesspolicy.xml;

<!-- Access control data group -->
&accesscontrol.xml;
&en_US_accesscontrol.xml;
&fr_FR_accesscontrol.xml;

<!-- Other data groups -->
&command.xml;
&store-default.xml;
</import>
```

where

- `import` is the root element of the XML document. The root element has already been defined in the `wcs.dtd` file, provided with WebSphere Commerce, and includes the definitions for all the WebSphere Commerce database tables. However, if you customized the WebSphere Commerce schema, you may need to use a different root element. You can generate a new DTD file that reflects the customized schema or you can update the existing `wcs.dtd` file.
- `all-store-assets.dtd` refers to the name of the main database asset group DTD file you will create in the next step.
- the commented text separates the different database asset groups for your store.
- `&`*`database asset`*`.xml;` is an XML entity reference to the database asset XML file. The path and location are defined in the database asset group DTD file. This name will change to match the database assets files already created for each group.

- `&`*`locale_database asset`*`.xml;` is needed for each language your store supports. If your store is unilingual, then only reference one file. If your store supports more than one language, then you require a reference for each language. The above extract assumes that your store supports the English and French languages.

5. Create a main database asset group DTD file that defines the above entities and the other DTD files required by the database assets.

   Using the following example for the entire set of store database assets as your guide, you can create your main database asset group DTD file:

```
<!ENTITY % wcs.dtd SYSTEM "absolute path for WebSphere Commerce wcs.dtd file">
%wcs.dtd;

<!ENTITY % NonStoreForeignKeys.dtd SYSTEM "NonStoreForeignKeys.dtd">
%NonStoreForeignKeys.dtd;
<!ENTITY fulfillment.xml SYSTEM "data/fulfillment.xml">
<!ENTITY en_US_fulfillment.xml SYSTEM "data/en_US/fulfillment.xml">
<!ENTITY fr_FR_fulfillment.xml SYSTEM "data/fr_FR/fulfillment.xml">

<!ENTITY store.xml SYSTEM "data/store.xml">
<!ENTITY en_US_store.xml SYSTEM "data/en_US/store.xml">
<!ENTITY fr_FR_store.xml SYSTEM "data/fr_FR/store.xml">

<!ENTITY tax.xml SYSTEM "data/tax.xml">
<!ENTITY en_US_tax.xml SYSTEM "data/en_US/tax.xml">
<!ENTITY fr_FR_tax.xml SYSTEM "data/fr_FR/tax.xml">
<!ENTITY taxfulfill.xml SYSTEM "data/taxfulfill.xml">

<!ENTITY shipping.xml SYSTEM "data/shipping.xml">
<!ENTITY en_US_shipping.xml SYSTEM "data/en_US/shipping.xml">
<!ENTITY fr_FR_shipping.xml SYSTEM "data/fr_FR/shipping.xml">
<!ENTITY shipfulfill.xml SYSTEM "data/shipfulfill.xml">

<!ENTITY catalog.xml SYSTEM "data/catalog.xml">
<!ENTITY en_US_catalog.xml SYSTEM "data/en_US/catalog.xml">
<!ENTITY fr_FR_catalog.xml SYSTEM "data/fr_FR/catalog.xml">
<!ENTITY store-catalog.xml SYSTEM "data/store-catalog.xml">
<!ENTITY storefulfill.xml SYSTEM "data/storefulfill.xml">
<!ENTITY offering.xml SYSTEM "data/offering.xml">
<!ENTITY store-catalog-tax.xml SYSTEM "data/store-catalog-tax.xml">
<!ENTITY store-catalog-shipping.xml SYSTEM "data/store-catalog-shipping.xml">

<!ENTITY currency.xml SYSTEM "data/currency.xml">
<!ENTITY en_US_currency.xml SYSTEM "data/en_US/currency.xml">
<!ENTITY fr_FR_currency.xml SYSTEM "data/fr_FR/currency.xml">

<!ENTITY campaign.xml SYSTEM "data/campaign.xml">
<!ENTITY en_US_campaign.xml SYSTEM "data/en_US/campaign.xml">
<!ENTITY fr_FR_campaign.xml SYSTEM "data/fr_FR/campaign.xml">

<!ENTITY businesspolicy.xml SYSTEM "data/businesspolicy.xml">
<!ENTITY en_US_businesspolicy.xml SYSTEM "data/en_US/businesspolicy.xml">
<!ENTITY fr_FR_businesspolicy.xml SYSTEM "data/fr_FR/businesspolicy.xml">

<!ENTITY accesscontrol.xml SYSTEM "data/accesscontrol.xml">
<!ENTITY en_US_accesscontrol.xml SYSTEM "data/en_US/accesscontrol.xml">
<!ENTITY fr_FR_accesscontrol.xml SYSTEM "data/fr_FR/accesscontrol.xml">

<!ENTITY command.xml SYSTEM "data/command.xml">
<!ENTITY store-defaults.xml SYSTEM "data/store-defaults.xml">
```

   where

- `wcs.dtd` refers to the DTD file containing data defined outside its database asset group. This file, provided with WebSphere Commerce, also defines the root element used in the database asset group XML file.
- `NonStoreForeignKeys.dtd` refers to the DTD file which defines elements other than the root element. This file contains all the XML entity reference declarations and definitions for the external dependencies outside the database asset group. As such, the XML files have references to foreign key values that are not created as part of the database asset group and must already be loaded into the database before this group.

  Note: Ensure that the path is correctly identified. In this example, the file is in the same directory as the main database asset group DTD file.
- `store.xml`, `en_US_store.xml`, and `fr_FR_store.xml` are the external reference entities used in the main database asset group XML file, assuming your store supports English and French. To use the reference, follow the entity reference convention: *&alias_name;*.
- *database asset*`.xml` refers to the name of the XML files from which the database assets are loaded. This name will change to match the database assets files already created for each group. Note that the locale-specific XML files are under the following directory:
  - NT *drive*`:\WebSphere\CommerceServer\samplestores\` *sample store name*`\data\`*locale*`\`
  - 2000 *drive*`:\Program Files\WebSphere\CommerceServer\samplestores\` *sample store name*`\data\`*locale*`\`
  - AIX `/usr/WebSphere/CommerceServer/samplestores/` *sample store name*`/data/`*locale*`/`
  - Solaris Linux `/opt/WebSphere/CommerceServer/samplestores/` *sample store name*`/data/`*locale*`/`
  - 400 `/QIBM/ProdData/WebCommerce/samplestores/` *sample store name*`/data/`*locale*`/`
- the *path_database asset*`.xml` files are needed for each language your store supports, located under the above directories. If your store is unilingual, then you would only reference one file. If your store supports more than one language, then you would require a locale-specific file for each language. The above extract assumes that your store supports the English and French languages.

6. Each database asset group requires information defined outside its domain or its set of data, as each group may have external dependencies. You can provide this data in a DTD file. For example, the store database asset group has the following external dependencies:

```
  bootstrap.LANGUAGE.LANGUAGE_ID, bootstrap.MEMBER.MEMBER_ID,
bootstrap.SETCURR.SETCURR_ID, fulfillment.FFMCENTER.FFMCENTER_ID
```

   When loading a database asset group or the entire set of store assets, the external dependencies must be defined from the WebSphere Commerce database. To use this data, follow the corresponding XML entity reference. For example, to use the data defined by the `ffmcenter_id` entity, you would write `&ffmcenter_id;` in your XML file. Using the following example for store database assets as your guide, you can create your DTD file based on this extract, called `ForeignKeys.dtd`:

```
<!ENTITY en_US "-1">
<!ENTITY fr_FR "-2">
<!ENTITY de_DE "-3">
<!ENTITY it_IT "-4">
<!ENTITY es_ES "-5">
<!ENTITY pt_BR "-6">
<!ENTITY zh_CN "-7">
<!ENTITY zh_TW "-8">
<!ENTITY ko_KR "-9">
<!ENTITY ja_JP "-10">
<!ENTITY MEMBER_ID "-2000">
<!ENTITY ffmcenter_id "10001">
```

where

- `MEMBER_ID` is the internal reference number that identifies the owner of the store.
- `ffmcenter` is the reference number for your store's fulfillment center. Since your store can use more than one fulfillment center, more than one can be defined in the `NonStoreForeignKeys.dtd` file.
- *`locale`* is the WebSphere Commerce reference number for each locale (identified by country or region and language). The values are located in the LANGUAGE database table.

Note: If you are splitting an existing store archive into database asset groups, ensure that all references to, as an example, alias `@ffmcenter_id` are replaced with the corresponding entity reference: `&ffmcenter_id;`.

7. Once all the data files have been created, run the IDResolve command against the main database asset group XML file to resolve the data as described in "ID Resolve command" on page 225.
8. Run the Load command on the resolved data file as described in "Load command" on page 232. To verify your loading process, refer to the log files:

   - DB2 `idresgen.db2.log` and `massload.db2.log`
   - Oracle `idresgen.oracle.log` and `massload.oracle.log`

   The log files are located under:

   - NT *`drive`*`:\WebSphere\CommerceServer\logs\`
   - 2000 *`drive`*`:\Program Files\WebSphere\CommerceServer\logs\`
   - AIX `/usr/WebSphere/CommerceServer/logs/`
   - Solaris Linux `/opt/WebSphere/CommerceServer/logs/`
   - 400 `/QIBM/ProdData/WebCommerce/logs/`
9. Business Run the AccountImport command as described in "Publishing business account assets" on page 274.
10. If applicable, publish contracts as described in "Publishing contract assets" on page 275.
11. Complete the tasks in "Publishing store front assets and store configuration files by copying to the WebSphere Commerce Server" on page 278

Loading database asset groups

To assist you in loading database assets, sample packages are available from the WebSphere Commerce Web site. These packages are based on the WebSphere Commerce sample stores and contain the files found in the steps below. You can download the packages from:

- Business `http://www.ibm.com/software/webservers/commerce/wc_be/downloads.html`
- Professional `http://www.ibm.com/software/webservers/commerce/wc_pe/downloads.html`

To load XML data for a single database asset group into the WebSphere Commerce database, do the following:

1. Review the following information:
   a. Appendix B, "Creating your data" on page 305
   b. Appendix E, "Database asset groups" on page 317, as you need to know which WebSphere Commerce asset files and database tables are affected.
   c. Chapter 27, "Overview of loading store data" on page 221, which provides the background information for the Loader package.
2. Plan your loading process and decide which database asset group you you will load. Whether you want to load the entire set of store database assets as instructed in "Loading a store" on page 261 or a single database asset group, the basic process remains the same. In the next steps, you will use or create the following files for your loading process:
   a. one or more database asset files, depending on which group you choose. For example, if you load the store database group assets, you will need a `store.xml` file and a separate `store.xml` file for each locale your store supports. Examples of such files are shipped with the WebSphere Commerce sample stores in the following directory:
      - NT `drive:\WebSphere\CommerceServer\samplestores\sample store name\data`
      - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores\sample store name\data`
      - AIX `/usr/WebSphere/CommerceServer/samplestores/sample store name/data`
      - Solaris Linux `/opt/WebSphere/CommerceServer/samplestores/sample store name/data`
      - 400 `/QIBM/ProdData/WebCommerce/samplestores/sample store name/data`

      Note that not all database asset groups require locale specific information.
   b. a new XML file which consolidates all the XML database asset files, contains the XML entity references, and contains the root element for the database assets. This is referred to as the *main database asset group XML file*. You can find this file in the sample package, called `store-all-assets.xml`.
   c. a new DTD file which defines all the data types required by the XML files from a database asset group, referred to as the *main database asset group DTD file*. You can find this file in the sample package, called `store-all-assets.dtd`.

d. a second DTD file which defines the external dependencies. You may need to include this file in the *main database asset group DTD file*. You can find this file in the sample package, called `Non`*`database asset group`*`ForeignKeys.dtd`.

e. a third DTD file containing the definition of all the WebSphere Commerce tables. The `wcs.dtd` file already exists in WebSphere Commerce, located in the following directory:

   - NT *`drive`*`:\WebSphere\CommerceServer\schema\xml\`
   - 2000 *`drive`*`:\Program Files\WebSphere\CommerceServer\schema\xml\`
   - AIX `/usr/WebSphere/CommerceServer/schema/xml/`
   - Solaris Linux `/opt/WebSphere/CommerceServer/schema/xml/`
   - 400 `/QIBM/ProdData/WebCommerce/schema/xml/`

   You may need to include this file in the *main database asset group DTD file*. If you have not customized the WebSphere Commerce schema, then you can use this file without modification.

3. Create the database asset XML files for the group you will load as instructed in previous chapters from this guide. If you completed the tasks in the asset chapters, then these XML files already exist. The database asset files should not contain any DTD declaration or page directives at the start of the file since this may cause conflicts when the files are concatenated. Also, for simplicity you may decide not to create any root elements. The only file that must have a root element is the main database asset group XML file.

   Note: If you have database asset files for more than one language, then each file must begin with `<?xml encoding =` *`locale specific encoding`*`>`. For example, English database asset files should specify `<?xml encoding = "UTF-8"?>`, but French files should specify `<?xml encoding = "ISO-8859-1"?>`.

4. Create the main database asset group XML file for each group you want to load. This file contains reference entities to include various XML files in one database asset group, or more. By using external reference entities, you can concatenate the XML files to simplify the ID Resolve command and the load process. Also, the internal aliases used within each XML file can be external to another XML data file within a group or across groups when loading more than one group at a time. An XML parser would substitute the contents of the file referenced by the external reference entity in place of the external reference.

   Using the following example for loading the single store database asset group as your guide, you can create your database asset group XML file based on this extract:

   ```
   <?xml version="1.0"?>
   <!DOCTYPE import SYSTEM "store-assets.dtd">
   <import>
   &store.xml;
   &en_US_store.xml;
   &fr_FR_store.xml;
   </import>
   ```

   where

   - `import` is the root element of the XML document. The root element has already been defined in the `wcs.dtd` file, provided with WebSphere Commerce, and includes the definitions for all the WebSphere Commerce database tables. However, if you customized the WebSphere Commerce schema, you may need to use a different root element. You can generate a new DTD file that reflects the customized schema or you can update the `wcs.dtd` file.

- `store-assets.dtd` refers to the name of the main database asset group DTD file you will create in the next step.
- `&store.xml;` is an XML entity reference to the database asset group XML file. The path and location are defined in the database asset group DTD file. This name will change to match the assets files already created for each group.
- *`locale`*`_store.xml;` is needed for each language your store supports. If your store is unilingual, then only reference one file. If your store supports more than one language, then you would require a reference for each language. The above extract assumes that your store supports the English and French languages.

5. Create a main database asset group DTD file that defines the above entities and the other DTD files required by the group.

   Using the following example for loading the single store database asset group as your guide, you can create your main database asset group DTD file for any data group:

```
<!ENTITY % wcs.dtd SYSTEM "absolute path for WebSphere Commerce wcs.dtd file">
%wcs.dtd;

<!ENTITY % NonStoreForeignKeys.dtd SYSTEM "NonStoreForeignKeys.dtd">
%NonStoreForeignKeys.dtd;
<!ENTITY store.xml SYSTEM "store.xml">
<!ENTITY en_US_store.xml SYSTEM "en_US/store.xml">
<!ENTITY fr_FR_store.xml SYSTEM "fr_FR/store.xml">
```

   where

   - `wcs.dtd` refers to the DTD file containing data defined outside its database asset group. This file, provided with WebSphere Commerce, also resolves and defines the root element used in the database asset group XML file.
   - `NonStoreForeignKeys.dtd` refers to the DTD file which defines elements other than the root element. This file contains all the XML entity reference declarations and definitions for the external dependencies outside the database asset group. As such, the XML files have references to foreign key values that are not created as part of the database asset group and must already be loaded into the database before this group.

     Note: Ensure that the path is correctly identified. In this example, the file is in the same directory as the database asset group DTD file.
   - `store.xml`, `en_US_store.xml`, and `fr_FR_store.xml` are the external reference entities used in the database asset group XML file. To use the reference, follow the entity reference convention: *&alias_name;*.
   - `store.xml` refers to the data file for the group from which the database assets are loaded. This name will change to match the database assets files already created for each group. Note that the locale-specific XML files are under the following directory:
     – NT *`drive`*`:\WebSphere\CommerceServer\samplestores\`*`sample store name`*`\data\`*`locale`*`\`

– 2000 *drive*:\Program Files\WebSphere\CommerceServer\samplestores\ *sample store name*\data*locale*\

– AIX /usr/WebSphere/CommerceServer/samplestores/ *sample store name*/data/*locale*/

– Solaris Linux /opt/WebSphere/CommerceServer/samplestores/ *sample store name*/data/*locale*/

– 400 /QIBM/ProdData/WebCommerce/samplestores/ *sample store name*/data/*locale*/

- the *path*_store.xml are needed for each language your store supports, located under the above directories. If your store is unilingual, then you would only reference one file. If your store supports more than one language, then you would require a locale-specific file for each language. The above extract assumes your store supports the English and French languages.

6. Each database asset group requires information defined outside its domain or its set of data, as each group may have external dependencies. You can provide this data in a DTD file. For example, the store database asset group has the following external dependencies:

```
  bootstrap.LANGUAGE.LANGUAGE_ID, bootstrap.MEMBER.MEMBER_ID,
bootstrap.SETCURR.SETCURR_ID, fulfillment.FFMCENTER.FFMCENTER_ID
```

When loading a data group or the entire set of store data, the following external dependencies must be defined from the WebSphere Commerce database. To use this data, follow the corresponding XML entity reference. For example, to use the data defined by the ffmcenter_id entity, you would write &ffmcenter_id; in your XML file. Using the following example for the store database asset group as your guide, you can create your DTD file based on this extract, called Non*database asset group*ForeignKeys.dtd:

```
<!ENTITY en_US "-1">
<!ENTITY fr_FR "-2">
<!ENTITY de_DE "-3">
<!ENTITY it_IT "-4">
<!ENTITY es_ES "-5">
<!ENTITY pt_BR "-6">
<!ENTITY zh_CN "-7">
<!ENTITY zh_TW "-8">
<!ENTITY ko_KR "-9">
<!ENTITY ja_JP "-10">
<!ENTITY MEMBER_ID "-2000">
<!ENTITY ffmcenter_id "10001">
```

where

- MEMBER_ID is the internal reference number that identifies the owner of the store.
- ffmcenter is the reference number for your store's fulfillment center. Since your store can use more than one fulfillment center, more than one can be defined in the NonStoreForeignKeys.dtd file.
- *locale* is the WebSphere Commerce reference number for each locale (identified by country or region and language). The values are located in the LANGUAGE database table.

Note: If you are splitting an existing store archive into database asset groups, ensure that all references to, as an example, alias `@ffmcenter_id` are replaced with the corresponding entity reference: `&ffmcenter_id;`.

7. Once all the data files have been created, run the IDResolve command against the database asset group XML file to resolve the data as described in "ID Resolve command" on page 225.
8. Run the Load command on the resolved data file as described in "Load command" on page 232. To verify your loading process, refer to the log files:
   - DB2 `idresgen.db2.log` and `massload.db2.log`
   - Oracle `idresgen.oracle.log` and `massload.oracle.log`

   The log files are located under:
   - NT *drive*`:\WebSphere\CommerceServer\logs\`
   - 2000 *drive*`:\Program Files\WebSphere\CommerceServer\logs\`
   - AIX `/usr/WebSphere/CommerceServer/logs/`
   - Solaris Linux `/opt/WebSphere/CommerceServer/logs/`
   - 400 `/QIBM/ProdData/WebCommerce/logs/`
9. Business Run the AccountImport command as described in "Publishing business account assets" on page 274.
10. If applicable, publish contracts as described in "Publishing contract assets" on page 275.
11. Complete the tasks in "Publishing store front assets and store configuration files by copying to the WebSphere Commerce Server" on page 278

Chapter 29. Publishing business accounts and contracts

Some of the store database assets, (business accounts, and contracts) cannot be loaded by the Loader package. You can publish these database assets by using Store Services or from the command line, as part of the publishing a complete store option, described in Chapter 26, "Publishing a complete store" on page 209, or you can publish business accounts and contracts using their corresponding commands. These commands are as follows:

- AccountImport— Creates business accounts from the `businessaccount.xml` file in the store archive.
- ContractImportApprovedVersion—Creates a contract from the `contract.xml` file. If the contract is in active state, the command creates and deploys the contract. Even if the `contract.xml file` contains more than one contract the command only needs to be called once.
- ProductSetPublish — Synchronizes the product set data in the product set database tables with the catalog before business accounts and contracts are created. Store Services and the command line publish call the ProductSetPublish command which then calls the AccountImport and ContractImportApprovedVersion commands.

Note: For more information on these commands, see the WebSphere Commerce online help.

Business account assets are included in the form of XML files in some of the sample store archives provided with WebSphere Commerce. However, it is recommended that you create business account assets using the tools provided, rather than creating XML files for these assets. For more information on creating these assets using the tools provided, see the WebSphere Commerce online help. The instructions for publishing business accounts are included in the following sections, in case you choose to publish the corresponding XML files provided with the sample store archives, or create your own.

Note: If you are not using Store Services to publish the business accounts or contracts, the store and catalog assets must be published before you can publish business accounts, and contracts. In particular you need the store and catalog identifiers, as well as the ID for the organization that owns the store, as well as the IDs for any buyer organizations associated with the contract. If the terms and conditions of your contract do not specify a particular catalog, you do not need to publish a catalog before publishing a business account or contract.
If you publish these assets using Store Services or the command line publish, ensure that you select the catalog option, or that your store already has a published catalog. If you publish these assets using the corresponding commands, ensure that you have already loaded the assets listed above into the database.

Publishing business accounts and contracts using Store Services or the command line

You can publish business accounts and contracts using Store Services, or using the publish utility from the command line. In order to publish the business accounts and contracts using either Store Services or the command line, the assets must be packaged in the store archive format. For more information on packaging the store front assets as a store archive, see Part 6, "Packaging your store" on page 199.

Using Store Services or the command line publish, you can choose to publish all types of assets in the store archive (including store front assets, store data assets and resource bundles) or you can choose to publish just one type of the assets. For detailed step-by-step instructions on publishing assets using Store Services or the command line, see the WebSphere Commerce online help.

Publishing business accounts and contracts using commands

If you prefer not to package your assets as a store archive, you can still publish the business accounts and contracts using the corresponding commands:

- AccountImport— Creates business accounts from the `businessaccount.xml` file in the store archive.
- ContractImportApprovedVersion— Imports an approved or active contract into WebSphere Commerce Server from an XML file. Before importing the contract, the command ensures that the contract being imported contains the necessary terms and conditions and is a valid contract.
- ProductSetPublish — Synchronizes the product set data in the product set database tables with the catalog before business accounts and contracts are created. Store Services and the command line publish call the ProductSetPublish command which then calls the AccountImport and ContractImportApprovedVersion commands.

Publishing business account assets

To publish the business account assets, do the following:

1. Using the Administration Console, update the view registry, or restart the WebSphere Commerce instance. For more information see the WebSphere Commerce online help.
2. Copy `businessaccount.xml` to the following directory:
   - NT `drive:\WebSphere\CommerceServer\xml\trading`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\xml\trading`
   - AIX `/usr/WebSphere/CommereServer/xml/trading`
   - Solaris `/opt/WebSphere/CommerceServer/xml/trading`
   - Linux `/opt/WebSphere/CommerceServer/xml/trading`
   - 400 `/QIBM/UserData/WebCommerce/instances/instancename/xml/trading`
3. Open `businessaccount.xml` and make the following changes:
   - Replace all occurrences of &STORE_IDENTIFIER; with the store identifier for your store.
   - Replace all occurrences of &MEMBER_IDENTIFIER; with the member distinguished name for your store.

Note: If you are working with a `businessaccount.xml` that is part of a store archive created using Store Services this step will already be completed.

4. Save and close the file.
5. Open the Administration Console. Login as an administrator.
6. In a browser, type the following:
   - `https://`*`hostname`*`:8000/webapp/wcs/stores/servlet/AccountImport?fileName=businessaccount.xml&URL=`*`URL to redirct to upon successful completion`*

Note: For more information on the command syntax and parameters, see the WebSphere Commerce online help.

Publishing contract assets

To publish the contract assets, do the following:

1. Copy `contract.xml` to the following directory:

   Note: This path is configurable. The following path is the default.
   - NT `drive:\WebSphere\CommerceServer\xml\trading`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\xml\trading`
   - AIX `/usr/WebSphere/CommerceServer/xml/trading`
   - Solaris `/opt/WebSphere/CommerceServer/xml/trading`
   - Linux `/opt/WebSphere/CommerceServer/xml/trading`
   - 400 `/QIBM/UserData/WebCommerce/instances/`*`instancename`*`/xml/trading`
2. Open `contract.xml` and make the following changes:
   - Replace all occurrences of &STORE_IDENTIFIER; with the store identifier for your store.
   - Replace all occurrences of &MEMBER_IDENTIFIER; with the member distinguished name for your store.
3. Save and close the file.
4. Open the Administration Console. Login as an administrator.
5. In a browser, type the following:
   - `https://`*`hostname:portnumber`*`/webapp/wcs/tools/servlet/ContractImportApprovedVersion?fileName=contract.xml&targetStoreId=store_id&URL=ContractDisplay`
6. If your store contains multiple `contract.xml` files (for example, locale specific contract files), repeat steps 1 through 5 for each `contract.xml` file.

Chapter 30. Publishing store front assets and store configuration files

Publishing the store front assets, the HTML and JSP files, properties files or resource bundles, and images and graphics that create your store pages, is part of the process of creating a functional store. You can publish your store front assets using Store Services or from the command line, as part of the publishing a complete store option, described in Chapter 26, “Publishing a complete store” on page 209, or you can publish the store front assets by simply copying the assets to a specified location on the WebSphere Commerce Server.

If you publish JSP files contained in the sample stores Business ToolTech and NewFashion, and you plan to configure the store to use the collaboration features, you will also need to publish the store configuration files that are part of that store archive. Both stores contain the following store configuration files:

- `tools_properties.zip`
- `tools_xml.zip`
- `runtime_xml.zip`

If you publish a complete store using Store Services or from the command line (selecting all publishing options), the store configuration files are also published. However, if you choose to publish the store front assets by copying them to the WebSphere Commerce Server, you will also have to copy the store configuration files to the WebSphere Commerce Server.

Publishing store front assets and store configuration files using Store Services or the command line

You can publish the store front assets and store configuration files using Store Services, or using the publish utility from the command line.

Note: In order to publish the store configuration files using Store Services or the publish utility, you must publish all store assets, including Web assets and data assets. You cannot choose to publish just the store configuration files.

In order to publish the store front assets and store configuration files using either Store Services or the command line, the store front assets and store configuration files must be packaged in the store archive format. For more information on packaging the store front assets as a store archive, see Part 6, “Packaging your store” on page 199.

Using Store Services or the command line publish, you can choose to publish all types of assets in the store archive (including store front assets, store data assets and resource bundles) or you can choose to publish just one type of the assets. For detailed step-by-step instructions on publishing assets using Store Services or the command line, see the WebSphere Commerce online help.

Publishing store front assets and store configuration files by copying to the WebSphere Commerce Server

If you prefer not to package your assets as a store archive, you can still publish the store front assets by copying them directly to the WebSphere Commerce Server. The Web assets (HTML, JSP files, images, and graphics) must be copied to the Web application document root. The resource bundles or properties files must be copied to the application's properties path.

To copy the store front assets and the store configuration files to the WebSphere Commerce Server, do the following:

1. Copy the JSP files, HTML, include files, images and graphics to the store directory (*storedir*) in the Stores Web application document root:
   - NT `drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_`*`instancename`*`.ear\wcstores.war\`*`storedir`*
   - 2000 `drive:\Program Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_`*`instancename`*`.ear\wcstores.war\`*`storedir`*
   - AIX `/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*
   - Solaris `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*
   - Linux `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*
   - 400 `/QIBM/UserData/WebASAdv4/WASinstancename/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*

   where *storedir* is the value of the DIRECTORY column from the STORE database table. If this value does not exist, you can add a value by doing the following: `select directory from store=`*`add relative directory name for publishing file assets`*
2. Copy the resource bundles and properties files to the application properties path:
   - NT `drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_`*`instancename`*`.ear\wcstores.war\WEB-INF\classes\`*`storedir`*
   - 2000 `drive:\Program Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_`*`instancename`*`.ear\wcstores.war\WEB-INF\classes\`*`storedir`*
   - AIX `/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/WEB-INF/classes/`*`storedir`*
   - Solaris `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/WEB-INF/classes/`*`storedir`*
   - Linux `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/WEB-INF/classes/`*`storedir`*

- 400 `/QIBM/UserData/WebASAdv4/WASinstancename/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/WEB-INF/classes/`*`storedir`*

3. Copy the store configuration files to the locations defined in the WebSphere Commerce Configuration File, `instance_name.xml`. This file is located in the following directory:
   - NT `drive:\WebSphere\CommerceServer\instances\`*`instancename`*`\xml`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\instances\`*`instancename`*`\xml`
   - AIX `/usr/WebSphere/CommerceServer/instances/`*`instancename`*`/xml`
   - Solaris `/opt/WebSphere/CommerceServer/instances/`*`instancename`*`/xml`
   - Linux `/opt/WebSphere/CommerceServer/instances/`*`instancename`*`/xml`
   - 400 `/QIBM/UserData/WebCommerce/instances/`*`instancename`*`/xml`

   The store configuration files are copied to the following locations:
   - `runtime_xml.zip` is copied to the StoresXMLPath. This path is defined in the WebSphere Commerce Configuration File, `instance_name.xml`.
   - `tools_properties.zip` is copied to the ToolsStoresPropertiesPath. This path is defined in the WebSphere Commerce Configuration File, `instance_name.xml`.
   - `tools_xml.zip` is copied to the ToolsStoresXMLPath. This path is defined in the WebSphere Commerce Configuration File, `instance_name.xml`.
4. Launch the store using one of the following methods:
   - Use the StoreCatalogDisplay command:
     `StoreCatalogDisplay?storeId=`*`storeId`*`&catalogId=`*`catalogId`*`&langId=`*`langId`*

     where
     - `storeId` is the value located in the STORE_ID column of the STORE database table,
     - `catalogId` is the value located in the CATALOG_ID column of the CATALOG database table,
     - `langId` is the value of the LANGUAGE_ID column of the LANGUAGE database table for a given locale. For a list of default WebSphere Commerce values, refer to the LANGUAGE database table.
   - If your store is based on a WebSphere Commerce sample store, assemble the store's URL by editing the `parameters.jsp` file under:
     - NT *`drive`*`:\WebSphere\AppServer\installedApps\WC_Enterprise_App_`*`instancename`*`.ear\wcstores.war\`*`storedir`*
     - 2000 *`drive`*`:\Program Files\WebSphere\AppServer\installedApps\WC_Enterprise_App_`*`instancename`*`.ear\wcstores.war\`*`storedir`*
     - AIX `/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_`*`instancename`*`.ear/wcstores.war/`*`storedir`*

– Solaris Linux `/opt/WebSphere/AppServer/installedApps /WC_Enterprise_App_`*`instancename`*`.ear/ wcstores.war/`*`storedir`*

– 400 `/QIBM/ProdData/WebCommerce/AppServer/ installedApps/WC_Enterprise_App_`*`instancename`*`.ear/ wcstores.war/`*`storedir`*

Add the correct values for the following parameters:

– `hostname` is the fully-qualified name of your WebSphere Commerce machine,

– `storeId` is the value located in the STORE_ID column of the STORE database table,

– `catalogId` is the value located in the CATALOG_ID column of the CATALOG database table,

– `langId` is the value of the LANGUAGE_ID column of the LANGUAGE database table for a given locale. For a list of default WebSphere Commerce values, refer to the LANGUAGE database table.

To view your store in a browser, launch the following URL: `http://`*`host name`*`/webapp/wcs/stores/servlet/`*`storedir`*`/index.jsp`

Part 8. Adding WebSphere Commerce features to your store

In order to add certain features available in WebSphere Commerce to your store, you need to complete some manual steps. The chapters in this section discuss adding the following features to your store:

- Chapter 31, “Adding customer care to your store” on page 283
- Chapter 32, “Adding e-Marketing Spots to your store” on page 295

Chapter 31. Adding customer care to your store

The customer care feature in WebSphere Commerce provides real-time customer service support by way of a synchronous text interface using the Lotus® Sametime™ server. When customer care is enabled in your store, a customer may enter the store, click on a link and connect to a Customer Service Representative (CSR). Then, the customer can can communicate with a CSR over the Internet.

Note: This chapter covers how to enable customer care in your store. However, before you can enable customer care in your store, you must first install a Sametime server and configure it to work with WebSphere Commerce. For more information, see the *WebSphere Commerce Additional Software guide*. You must also register CSRs in the Administration Console to enable them to use customer care. For more information on this task, as well as the overall concepts of customer care and how a CSR uses customer care, see the WebSphere Commerce online help.

Note:

You can enable customer care in your store quickly and easily using Store Services, if you create your store based on one of the following sample stores: ▶Business ToolTech and NewFashion. After publishing the store using Store Services, select the Stores view, then select the store, then select **Configure** and enable the customer care features. For more detailed instructions, see the WebSphere Commerce online help.

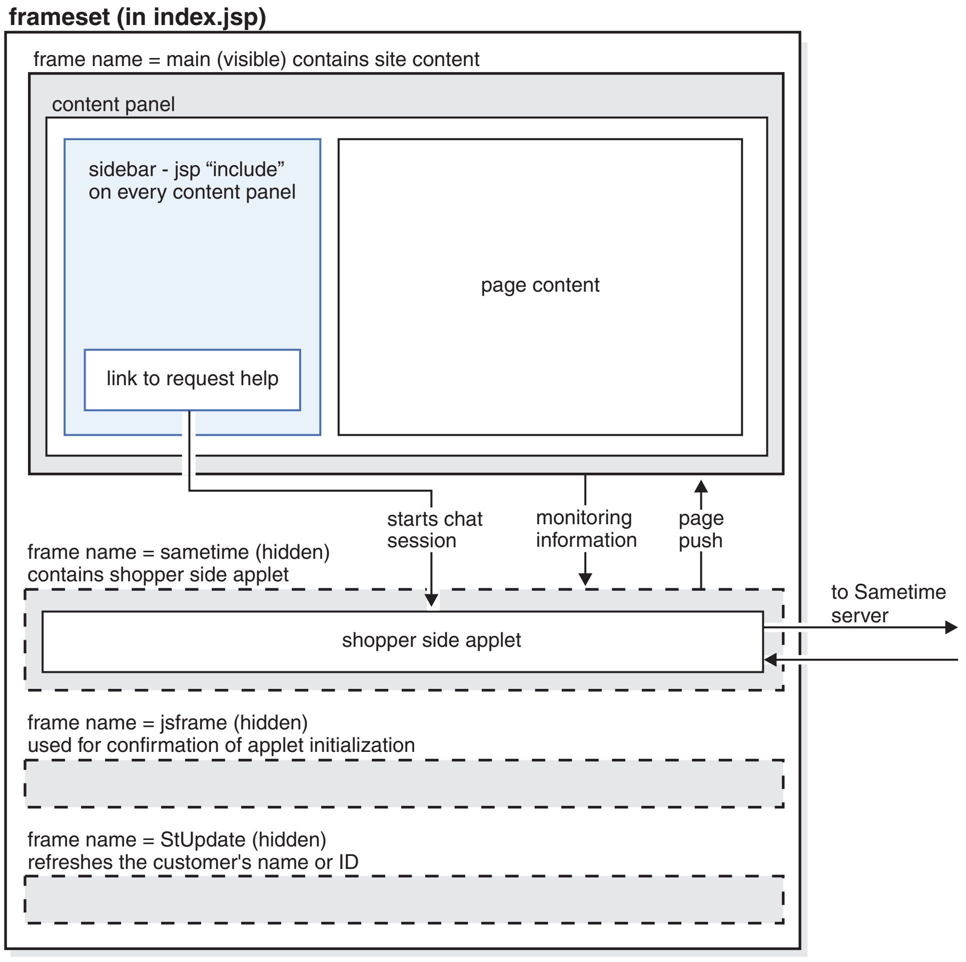
However, if you do not create your store using a sample store as a base, you will have to do some work to enable customer care in your store. The remainder of this chapter discusses the concepts and steps necessary to enable customer care in a store not based on one of the samples.

Note: The sample stores ▶Business ToolTech and NewFashion demonstrate how customer care should be implemented, and provide the code that you can use in your store to enable customer care. This chapter will refer to examples from these two stores to illustrate how to enable customer care in your store. Ensure that you have the latest version of the sample stores (available from the WebSphere Commerce product Web site) when reading this chapter.

Understanding customer care in a store

When a customer selects the customer care link, for example, **Live Chat with Customer Assistant**, in a store enabled with customer care, an applet containing the chat window is launched. This applet is run within a hidden frameset that does not interfere with the look and feel of the site. When the applet is launched it connects to a Lotus Sametime server.

The following diagram illustrates the composition of the frameset.


frameset (in index.jsp)
frame name = main (visible) contains site content
content panel
sidebar - jsp “include” on every content panel
link to request help
page content
starts chat session
monitoring information
page push
frame name = sametime (hidden) contains shopper side applet
shopper side applet
to Sametime server
frame name = jsframe (hidden) used for confirmation of applet initialization
frame name = StUpdate (hidden) refreshes the customer's name or ID

The frameset includes four frames:

- Main: The frame that contains the content for your store, including the files that create the store pages, that is the files that create the body of the page, the header and footer files, and the sidebar files. The contents of this frame are visible to the visitors to your store. Note that the main frame contains the following connections to the Sametime frame: a link to customer care and monitoring information. Monitoring information is discussed in more detail in “Monitoring customers using customer care” on page 286.
- Sametime: The frame that contains the customer care applet. This frame is not visible to your store’s visitors. However, when a customer clicks on the link to launch the applet, the customer will see the customer care window. This frame also pushes information to the main frame, through the page push feature.
- jsframe: The frame that confirms that the applet has been loaded properly. The contents of this frame do not display to customers.
- StUpdate: The frame refreshes the customer’s name or ID.

Using the frameset

Launching the customer care applet in a frameset separates the applet code from the code in the store pages. As the diagram above illustrates, the store pages are contained in the main frame of the frameset, while the applet code is contained in the Sametime frame. By separating the applet code from the store pages you reduce network traffic, as the applet is only downloaded once, when the frameset is first launched. If the customer care applet was not part of the frameset, it would have to be in every store page, and would be downloaded every time a new store page was accessed.

Using a frameset also allows you to maintain the connection with the Sametime server. If the applet was part of each page and not the frameset, a new Sametime session would be created each time a customer accessed a new page. Since the customer care applet logs on to the Sametime server anonymously, creating a new session each time a customer accessed a new page would not allow you to trace the customers activities through the store. Using the frameset, the customer's original Sametime session is maintained, and the customer's activities are sent back to the Sametime server as attributes change.

Issues with using framesets

Although using a frameset is the recommended method to implement Customer Care in your store, you should be aware of the following issues with using framesets:

- Single point of entry: Customers can only use customer care if they browse your store within the framework. Likewise, CSRs can only monitor customer movement through the frameset. To ensure that customers are browsing the store via the frameset, they must access the site through a single entry point, for example through the store's home page (in the case of the sample stores, `index.jsp`). If a customer accesses your store through another page (for example, a catalog page), they will not be in the frameset.
- Bookmarking: When using the frameset customers will only be able to bookmark the main URL for the site, not individual pages.
- Refreshing: When a customer is in the frameset and clicks refresh, they will be taken back to the main frame address, as coded in the frameset, for example, `index.jsp`.
- Resizing browser window: If a customer resizes the browser window while in the frameset, the browser may automatically reload the entry address. If the entry address is reloaded, the connection to the Sametime Server may be terminated. Different browsers behave differently in this situation.
- Security: When a customer is browsing a site through a frameset, each individual frame, as well as the frameset (the URL in the location bar) maintains its own connection, either unsecure (`http`, by default port 80) or secure (`https`, by default port 443). If a customer is browsing the store via an unsecure connection all frames within the frameset are in `HTTP`. In this scenario there are no issues with SSL. However, if the customer browses to a secure page (for example, the registration page), the `main` frame within the frameset will switch to `HTTPS`, while the rest of the frames remain unsecure (`http`). In this situation, a customer will not be able to launch the customer care applet. The browser will not authorize launching the applet, because the applet (secure, port 443) appears to be coming from a different server than the URL in the location bar of the browser (HTTP, port 80). The applet cannot be launched until the the entire frameset becomes secure again, that is, until the URL in the location bar points to `HTTPS`. To make the entire frameset secure again, you have the following options:
  - Redirect the entire frameset to a secure connection. However, when doing so, you will end any chat currently in progress, as the applet switches to a new Sametime connection.
  - Redirect the entire frameset to a secure connection upon entry to the site. This imposes a slight performance penalty, but adds security and browsing privacy to the customer's session.

One method of redirecting the frameset to a secure connection, is to add the following code to the index.html file:

-
  ```
  <html>
  <head>
  <META HTTP-EQUIV=Refresh CONTENT="0;URL=https://hostname/webapp/wcs/stores/
  servlet/NewFashion/index.jsp>
  </head>
  <html>
  </head>
  ```

Monitoring customers using customer care

Customer care allows you to monitor the customers who are corresponding with the CSRs in your store by

- Obtaining the customer's name or ID
- Determining which page the customer is browsing
- Tracking the items in the shopping cart

Customized code is added to the store pages in order to obtain this information. The following sections discuss how each of these monitoring features are implemented in the sample stores.

Obtaining the customer's name or ID

Once the customer care applet is launched and the CSR is logged on, the CSR is able to identify who is using the applet by name or by shopper ID. The sample stores include specialized code that work with the customer care applet to determine the customer's name or shopper ID. This code determines whether the customer is a guest customer, a guest customer with items in a shopping cart or a registered customer, then assigns a name or ID to the customer, and passes this name back to the customer care applet. These names then display to the CSR. For example, if the customer is a guest customer, who hasn't placed anything in the shopping cart, the customer is assigned a generated ID, with shopper ID -1002. If the customer is a guest with items in the shopping cart, the shopper ID will display, and if the customer is registered, their first name and last names display.

The sample stores obtain the customer's name or ID by adding the following code to the store's header file which refreshes the StUpdate frame. This code is included in header.jsp in NewFashion and in NavHeader.jsp in Business ToolTech.

Note: Each time a customer browses a new page in the store, the customer's name or ID is refreshed.

```
<script language="javascript">
    if (typeof top.updateStInfo == 'function')
       top.updateStInfo();
</script>
```

The preceding code refreshes the following in the StUpdate frame.

```
//set Customer Name for LiveHelp if user is registered.

if (userRegistrationDataBean.findUser()) {
 if (userRegistrationDataBean.getLastName() != null & &
userRegistrationDataBean.getLastName().length() > 0) {
      if(cmdcontext != null) {
       Long uid = cmdcontext.getUserId();
       String customerName ="";
       if (locale.toString().equals("ja_JP")||locale.toString().equals("ko_KR")
||locale.toString().equals("zh_CN")||locale.toString().equals("zh_TW"))
               {
                customerName = "" + userRegistrationDataBean.getLastName() + " "
```

```
 + userRegistrationDataBean.getFirstName();
               }
               else {
           customerName = "" + userRegistrationDataBean.getFirstName() + " "
+ userRegistrationDataBean.getLastName();
          }
        }
     else {
         customerName=userRegistrationDataBean.getUserId();
           if (customerName.equals("-1002"))
           customerName="";
              customer_name=customer_name.trim();
       }
```

In the sample store's Logout page, more custom code is included, which sets the customer name to a generated ID and resets the number of items in the shopping cart to zero. The Logout page in NewFashion is LoginForm.jsp. In Business ToolTech it is Logoff.jsp. The custom code is as follows:

```
<HTML>
<HEAD>
<SCRIPT language="javascript">
 if (typeof parent.setCustomerName == 'function')
     parent.setCustomerName (parent.WCSGUESTID, '')
   if (typeof parent.setShoppingCartItems == 'function')
     parent.setShoppingCartItems(0);
</SCRIPT>
</HEAD>
</HTML>
```

Determining which page the customer is browsing

Customer care also allows CSRs to determine what page the customers in the store are currently browsing. The sample stores determine what pages the customers are in, by adding the following code to the header file (header.jsp in NewFashion and in NavHeader.jsp in Business ToolTech):

```
<%
//Determine Page Type for LiveHelp
String headerType = (String) request.getAttribute("liveHelpPageType");
if (headerType==null)
 headerType = "";

%>

<script language="javascript">
<%
    String pname = request.getRequestURI();
    int indpn = pname.lastIndexOf('/');
    indpn = pname.lastIndexOf('/', indpn-1);
    if(indpn != -1)
        pname = pname.substring(indpn+1);

    //Determine if this is a personal page or not
    if (headerType.equals("personal") ) {
%>
   if (typeof parent.setPageParams == 'function')
    parent.setPageParams('PERSONAL_URL', '<%=pname%>');
<% } else { %>
 if (typeof parent.setPageParams == 'function')
  parent.setPageParams(location.href, '<%=pname%>');
<% } %>
</script>
```

If a page does not use the header file, the file StHeader1.jsp is included in the page. StHeader1.jsp contains the same code that is added to the store's header file.

In order to maintain customer privacy, CSRs should not have access to certain pages. For example, a CSR might not have access to a campaign page, a page that includes a price determined by a contract, or a page that includes the user ID, for example, the address book page. These pages are marked as personal. In the sample stores the following pages are marked as personal:

- NewFashion
  - `AddressBookForm.jsp`
  - `AllocationCheck.jsp`
  - `edit_registration.jsp`
  - `emptyshoppingcart.jsp`
  - `interestItemDisplay.jsp`
  - `myaccount.jsp`
  - `orderItemDisplay.jsp`
  - `OrderDisplayPending.jsp`
  - `ResultList.jsp`
  - `shoppingcart.jsp`
  - `TrackOrderStatus.jsp`
- Business ToolTech
  - `Address.jsp`
  - `Addressbook.jsp`
  - `AddToExistReqList.jsp`
  - `AdvancedSearch.jsp`
  - `AllocationCheck.jsp`
  - `CatalogMainDisplay.jsp`
  - `CatalogItemDisplay.jsp`
  - `CatalogTopCategoriesDisplay.jsp`
  - `Confirmation.jsp`
  - `OrderDisplayPending.jsp`
  - `OrderItemDisplay.jsp`
  - `OrderDetail.jsp`
  - `QuickOrder.jsp`
  - `RequisitionListCreate.jsp`
  - `RequisitionListDetailDisplay.jsp`
  - `RequsitionListDisplay.jsp`
  - `RequsitionListUpdate.jsp`
  - `Result List.jsp`
  - `Shipping.jsp`
  - `shoppingcart.jsp`
  - `TrackOrderStatus.jsp`
  - `UserAccount.jsp`
  - `UserRegistrationUpdate.jsp`

In order to mark pages as personal, that is, not available to the CSR, the sample stores include the following code in the page, just before the header is included.

```
<%
// Set header type needed for this JSP for Customer Care.  This must
// be set before Header.jsp
```

```
request.setAttribute("liveHelpPageType", "personal");
%>

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

Note: Although a CSR cannot see the content of a page marked personal, the CSR can see the URL of that page.

Tracking the number of items in the shopping cart

Customer Care also allows CSRs to track how many items a customer has in their shopping cart at any time. The sample stores obtain the number of items in the shopping cart in the following locations:

- Shopping cart page (NewFashion: `shoppingcart.jsp`, Business ToolTech:`ShoppingCart.jsp`)
- Empty shopping cart page (NewFashion: `emptyshopcart.jsp`, Business ToolTech:`EmptyOrder.jsp`)
- Order confirmation page (NewFashion: `confirmation.jsp`, Business ToolTech:`confirmation.jsp`)
- Logout page (Business ToolTech:`Logoff.jsp`)

Note: Although the shopping cart is designated as a personal page, CSRs can track the number of items in the shopping cart. They cannot see what the shopping cart contains using these method, only the number of items in it. However, a CSR can view the contents of the shopping cart using the **View Shopping Cart** button. For more information, see the WebSphere Commerce online help.

The sample stores determine the number of items in the shopping cart by adding the following code to the above pages:

- First an int variable is defined

  ```
  int liveHelpShoppingCartItems = 0;
  ```
- Next, the following line of code is used to add the quantity to liveHelpShoppingCartItems whenever there is an orderitem addition to the cart:

  ```
  liveHelpShoppingCartItems+= orderItem.getQuantityInEJBType().intValue();
  ```
- Then, the following code is added at the end of the page to set the customer name to the guest shopper ID, and to obtain the number of items in the customer's shopping cart.

  ```
  <script language="javascript">
   if (typeof parent.setShoppingCartItems == 'function')
    parent.setShoppingCartItems(<%=liveHelpShoppingCartItems%>);
  </script>
  ```

The following code is used in the empty shopping cart page and the order confirmation page to reset the number of items in the cart to zero:

```
<script language="javascript">
 if (typeof parent.setShoppingCartItems == 'function')
  parent.setShoppingCartItems(0);
</script
```

Adding customer care to your store

To add customer care to a store that is not based on a sample, do the following:

Part 1: Installing pre-requisites

In order for customer care to work in your store, you must do the following:

- Install a Sametime server. For more information, see the *WebSphere Commerce Additional Software guide*.
- Install the WebSphere Commerce Sametime integration package. For more information, see the *WebSphere Commerce Additional Software guide*.
- Stop the WebSphere Commerce instance, then enable Sametime in the Configuration Manager, then restart the instance. For more information, see the *WebSphere Commerce Additional Software guide*.
- Create a CSR and register the CSR for customer care using the Administration Console. For more information, see the WebSphere Commerce online help.

Part 2: Copying the customer care integration files from the sample store

The sample stores NewFashion and ToolTech include the following files which are used to integrate customer care into the store:

- `Sametime.js`: Contains JavaScript functions that are included for all frames. The functions from this file are called with a parent prefix from pages in the main frame, for example, parent.setCustomerName.
- `StBlank.jsp`: An empty JSP file.
- `StFrame.jsp`: Contains JavaScript functions and embeds the customer care applet for the store front.
- `StReadyJS.jsp`: Indicates that the applet is loaded properly.
- `StHeader1.jsp`: A header file that passes in a parameter to the applet indicating whether the page that includes this header is a personal page or not.
- `StUpdate.jsp`: Updates the customer's name information and ID.

To copy the Sametime integration files from the sample store to your store, do the following:

1. Locate the store archive file for the NewFashion store or the ToolTech store. The store archive files are located in the following directory:
   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
   - AIX `/usr/WebSphere/CommerceServer/samplestores`
   - Solaris `/opt/WebSphere/CommerceServer/samplestores`
   - Linux `/opt/WebSphere/CommerceServer/samplestores`
   - 400 `/qibm/proddata/WebCommerce/samplestores`
2. Open either the ToolTech or NewFashion folder, then select a ToolTech or NewFashion store archive.
3. Open the store archive file using WinZip or a similar tool.
4. Locate the `webapp.zip` file. Open it using WinZip of a similar tool.
5. Select the following files:
   - `Sametime.js`

- `StBlank.jsp`
- `StFrame.jsp`
- `StReadyJS.jsp`
- `StHeader1.jsp`
- `StUpdate.jsp`

6. Extract the files to the directory that contains the web assets for your store.

   Note: `StHeader1.jsp` is in an include directory in the sample store archives. If you have a separate directory for the files that you include into your store pages, save `StHeader1.jsp` in that directory. If not, save it in the same directory as your other store files.

Part 3: Adding the frameset to your store

As discussed in "Using the frameset" on page 284, the customer care applet runs in a frameset. This frameset should be added to your store home page, or the page customer's are most likely to enter the store through. To add the frameset to your store, do the following:

1. Determine which page is the point of entry for your store.
2. Open the store archive file for ToolTech or NewFashion. The store archive files are located in the following directory:
   - NT `drive:\WebSphere\CommerceServer\samplestores`
   - 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
   - AIX `/usr/WebSphere/CommerceServer/samplestores`
   - Solaris `/opt/WebSphere/CommerceServer/samplestores`
   - Linux `/opt/WebSphere/CommerceServer/samplestores`
   - 400 `/qibm/proddata/WebCommerce/samplestores`
3. Open the store archive file using WinZip or a similar tool.
4. Locate the `webapp.zip` file. Open it using WinZip or a similar tool.
5. Open the `index.jsp` file.
6. Copy the following code:

```
<script src="<%="Sametime.js"%></script>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml1-transitional.dtd">
<html>
<head>
<title></title>
<script language="javascript">
var MainPageURL="";
 if (MainPageURL=="")
MainPageURL="/webapp/wcs/stores/servlet/Logoff?storeId=<%=storeId%>
&langId=<%=langId%>&URL=LogonForm?
storeId=<%=storeId%>&catalogId=<%=catalogId%>";
 function loadFrame()
 {
  main.document.location.href=MainPageURL;
  }
</script>
</head>
<%
String sBasePath="/webapp/wcs/stores/servlet/";
String sSametimeUrl="StFrame.jsp?storeId="+storeId;
String sBlankUrl="StBlank.jsp?storeId="+storeId;
String sUpdateUrl="StUpdate.jsp?storeId="+storeId;
```

```
%>
<FRAMESET border=0 frameBorder=0 ROWS="100%,1,1,1,1,1" onLoad="loadFrame();">
<FRAME NAME="main"
 SRC="javascript:top.loadFrame();" MARGINWIDTH=0 SCROLLING="Auto"
FRAMEBORDER="no" noresize>
<FRAME NAME="JSFrame"
 SRC="<%=sBlankUrl% >" MARGINWIDTH=0 SCROLLING="no"
FRAMEBORDER="no" noresize>
<FRAME NAME="sametime" SRC="<%=sSametimeUrl%>" MARGINWIDTH=0 SCROLLING="no"
FRAMEBORDER="no" noresize>
<FRAME NAME="StUpdate" SRC="<%=sUpdateUrl%>" MARGINWIDTH=0 SCROLLING="no"
FRAMEBORDER="no" noresize>
</FRAMESET>
</html>
```

Note: In the example above, from the ToolTech sample store, the source (SRC) for the ″main″ frame is a command to another page. Depending on how your store is set up, the SRC can be a command, a JSP file, or an HTML file.

7. Paste the code you copied in step 6 into the page you have designated as the store's entry point. Make any necessary changes to the source (SRC) for the main frame.
8. Save the file.

Part 4: Adding the code to obtain the customer's name or ID

In order to display the name of shopper ID of the customer using customer care to the CSR, do the following:

1. Review the information in "Obtaining the customer's name or ID" on page 286.
2. Determine where you plan to obtain the customer's name or shopper ID from. For example,
   - Point of entry to the store
   - New registration or update registration
   - Logout
   - Header
3. After determining where you plan to obtain the customer's name or shopper ID from, determine whether those pages include the main header file for your store. If not, determine whether you will add the necessary code to the page itself, or include a header file.
4. From the `webapp.zip` file in the NewFashion store archive, or the ToolTech store archive, open one of the following files
   - NewFashion: `header.jsp`
   - ToolTech: `NavHeader.jsp`

   Note: Both files are located in the include directory in the `webapp.zip` file.
5. Copy the following code:
   ```
   <script language="javascript">
      if (typeof top.updateStInfo == 'function')
    top.updateStInfo()
   </script>
   ```
6. Paste the code you copied in step 5 into either the header file for your store, or directly into the appropriate pages.

   Note: If you prefer, you can include the file `StHeader1.jsp` into the appropriate page, instead of copying the above code into the store header file, or directly into the file.

7. Save the files.
8. (Optional) To reset the customer ID in the Logout page (for example from the customer's ID to -1002) and to reset the number of items in the shopping cart to zero, add the following code to the Logout file:

```
<HTML>
<HEAD>
<SCRIPT language="javascript">
 if (typeof parent.setCustomerName == 'function')
     parent.setCustomerName (parent.WCSGUESTID, '')
   if (typeof parent.setShoppingCartItems == 'function')
     parent.setShoppingCartItems(0);
</SCRIPT>
</HEAD>
</HTML>
```

Part 5: Adding code to determine which page the customer is browsing

To determine which page the customer is browsing, do the following:

1. Include the `StHeader1.jsp` file to the store's header file, for example:

```
<%@ include file="StHeader1.jsp" %>
```

2. Add the following code to any pages that should be marked personal, and thus not available for access by the CSR:

```
<%
// Set header type needed for this JSP for Customer Care.  This must
// be set before Header.jsp
request.setAttribute("liveHelpPageType", "personal");
%>

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

3. Add the following code to any pages that don't use a header, but should be marked personal:

```
<%
 // Set header type needed for this JSP for Customer Care.  This must
 // be set before StHeader1.jsp
 request.setAttribute("liveHelpPageType", "personal");

 String incfile;
 incfile = includeDir + "StHeader1.jsp";
 %>
 <jsp:include page="<%=incfile%>" flush="true"/>
```

Part 6: Adding code to track the number of items in the shopping cart

To allow CSRs to track the items that a customer has in their shopping cart, do the following:

1. Review the information in "Tracking the number of items in the shopping cart" on page 289.
2. Determine where you plan to track the shopping cart items. For example,
   - Shopping cart page
   - Empty shopping cart page
   - Order confirmation page
   - Logout page

3. In the pages that you plan to track the shopping cart items, do the following:
   a. Define an int variable:
      ```
      int liveHelpShoppingCartItems= 0;
      ```
   b. To add the quantity to shoppingCartItems when an order item is added to the cart, add the following line of code:
      ```
      liveHelpShoppingCartItems+= orderItem.getQuantityInEJBType().intValue();
      ```
   c. At the end of the page, add the following code, to set the guest shopper ID (-1002) to the actual shopper ID, and to obtain the number of items in the shopping cart:
      ```
      <script language="javascript">
      if (typeof parent.setCustomerName == 'function')
        parent.setCustomerName(<%=cmdcontext.getUserId()%>, parent.CustomerName);
       if (typeof parent.setShoppingCartItems == 'function')
        parent.setShoppingCartItems(<%=liveHelpShoppingCartItems%>);
      </script>
      ```
   d. If you plan to track data in the empty shopping cart page and order confirmation page, add the following code to those pages, to reset the shopping cart value to zero:
      ```
      <script language="javascript">
       if (typeof parent.setShoppingCartItems == 'function')
        parent.setShoppingCartItems(0);
      </script
      ```

Part 7: Adding a link to customer care

To allow customers to access customer care in your store, do the following:

1. Determine where you would like to place the link to customer care. For example, you may want to place the link in a navigation bar, so it is always available to customers, or in certain pages in the store.
2. Copy the following code into the pages that will contain the link:
   ```
   <a href="javascript:if((parent.sametime != null)) top.interact();">
   <%=infashiontext.getString("LiveHelp")%></a>
   ```

Part 8: Changing messages that display to the customer

The messages that display to a customer when they initially connect to a CSR, for example, ″Hello, how can I help you?″, or ″Our office hours are from 9 a.m. to 9 p.m. are stored in properties files in on the Sametime Server. The properties files are divided into two types of files: `Customer.properties` and `Agent.properties`. The `Customer.properties` file contains messages that display to the customer, while the `Agent.properties` file contains information that displays to the CSR. Both of these files also have corresponding locale-specific files, for example `Customer_de_DE.properties` and `Agent_de_DE.properties`, for each locale installed in your instance of WebSphere Commerce.

To change the messages in these files, do the following:

1. Locate the properties files on the Sametime Server. By default, these properties files are located in the following directory:
   - NT `drive:\Sametime\Data\domino\html\wc\properties`
2. Make the necessary changes.
3. Close and save the file.

Chapter 32. Adding e-Marketing Spots to your store

e-Marketing Spots reserve space on your store pages in which personalized marketing content for campaign initiatives displays. When a page is requested by a customer, any e-Marketing Spots present on the page will communicate with the rule server to process the rule-based code associated with the spot. Each e-Marketing Spot has one or more campaign initiatives associated with it. For more information on campaigns and campaign initiatives, see Chapter 12, “Campaign assets” on page 113 and the WebSphere Commerce online help.

In order to for campaign initiatives to display correctly on your store pages, an e-Marketing Spot must be added to the JSP file, and then registered in the database using the WebSphere Commerce Accelerator. This chapter discusses how to add e-Marketing Spots to the store’s JSP files. For more information on registering the e-Marketing Spot in the database using the WebSphere Commerce Accelerator, see the WebSphere Commerce online help.

e-Marketing Spot

The following is an example of an e-Marketing Spot.

```
<!-- ===========================================================================
//*------------------------------------------------------------------------------
//* The sample contained herein is provided to you "AS IS".
//*
//* It is furnished by IBM as a simple example and has not been thoroughly tested
//* under all conditions.  IBM, therefore, cannot guarantee its reliability,
//* serviceability or functionality.
//*
//* This sample may include the names of individuals, companies, brands and
//* products
//* in order to illustrate concepts as completely as possible.
//*All of these names
//* are fictitious and any similarity to the names and addresses used by actual
 persons
 //*or business enterprises is entirely coincidental.
//*--------------------------------------------------------------------------------
//*
================================================================================-->
<%
 /**
  * START - the following code should exist only once in a page, it initialize the
  *         command context and store data bean.
  */

 // create the store bean to get the store directory
 String collateralPath = "/webapp/wcs/stores/";
 com.ibm.commerce.command.CommandContext emsCommandContext =
(com.ibm.commerce.command.CommandContext) request.getAttribute(
ECConstants.EC_COMMANDCONTEXT);
 com.ibm.commerce.common.beans.StoreDataBean storeDataBean =
new com.ibm.commerce.common.beans.StoreDataBean();
 storeDataBean.setStoreId(emsCommandContext.getStoreId().toString());
 com.ibm.commerce.beans.DataBeanManager.activate(storeDataBean, request);
 if (storeDataBean.getDirectory() != null) {
  collateralPath += storeDataBean.getDirectory() + "/";
 }
%>
<!-- ===============================================================================
// The following HTML form submits the request on the e-marketing spot to the
 ClickInfo
// command which captures the campaign statistics, and redirect to the location
//specified by the URL parameter.
```

```
=============================================================================== -->
<form name="storeEmsForm" method="POST" action="/webapp/wcs/stores/servlet/ClickInfo">
  <input type="hidden" name="evtype">
  <input type="hidden" name="mpe_id">
  <input type="hidden" name="intv_id">
  <input type="hidden" name="URL">
 </form>
 <%
 /**
  * END - the following code should exist only once in a page, it initialize the
  *       command context and store data bean.
  */
%>
<%
  /**
  * START - the following code can be used to drop multiple e-marketing
spots onto the page.
  *         Customize the appropriate EMarketingSpot instance name and the
e-marketing spot
  *         name before use.  Duplicate this code if more than 1 spot is
needed, do not use
  *         the same spot name.
  */

 // create the e-Marketing Spot
 com.ibm.commerce.marketing.beans.EMarketingSpot eMarketingSpot =
new com.ibm.commerce.marketing.beans.EMarketingSpot();

 // IMPORTANT - set the correct name here
 eMarketingSpot.setName("eMarketingSpotName");

 // the maximum number of products/categories/ad copies that display
 //through this e-marketing spot can be set here
 eMarketingSpot.setMaximumNumberOfCatalogEntries(20);
 eMarketingSpot.setMaximumNumberOfCategories(20);
 eMarketingSpot.setMaximumNumberOfCollateral(20);

 // instantiate the bean
 com.ibm.commerce.beans.DataBeanManager.activate(eMarketingSpot,
 request);
%>

<%
 // The following block is used to display the advertisements associated with this
 // e-marketing spot.  The URL link defined with an advertisement can be referenced
 //through the submittion of the HTML form attached above.

 if (eMarketingSpot.getCollateral() != null && eMarketingSpot.getCollateral().
length > 0) {
%>
<TABLE>
<%  for (int i=0; i eMarketingSpot.getCollateral().length; i++) { %>
  <TR>
<%    if (eMarketingSpot.getCollateral()[i].getTypeName().equals("Image")) { %>
   <TD>
    A HREF="javascript:document.storeEmsForm.evtype.value='CpgnClick'
;document.storeEmsForm.mpe_id.value=' <%= eMarketingSpot.getId() %>
';document.storeEmsForm.intv_id.value='<% = eMarketingSpot.getCollateral
()[i].getInitiativeId() %>';document.storeEmsForm.URL.value='
<%= eMarketingSpot.getCollateral()[i].getUrlLink() %>';
document.storeEmsForm.submit();">
    IMG SRC=" <%= collateralPath + eMarketingSpot.getCollateral()[i].getLocation() %>">
    </A>
  </TD>
   <TD>
    <%= eMarketingSpot.getCollateral()[i].getMarketingText() %>
   </TD>
 <%  }  else if (eMarketingSpot.getCollateral()[i].getTypeName().equals("Flash")) { %>
  <TD>
    EMBED src=" <% = collateralPath + eMarketingSpot.getCollateral()[i].getLocation()
 %>" quality=high bgcolor=#FFFFFF WIDTH=120 HEIGHT=90
 TYPE="application/x-shockwave-flash"> </EMBED>
```

```
   </TD>
   <TD>
    A HREF="javascript:document.storeEmsForm.evtype.value='CpgnClick'
;document.storeEmsForm.mpe_id.value='<% = eMarketingSpot.getId()
 %>';document.storeEmsForm.intv_id.value='<% = eMarketingSpot.getCollateral()
[i].getInitiativeId() %>';document.storeEmsForm.URL.value='
<% = eMarketingSpot.getCollateral()[i].getUrlLink() %>';
document.storeEmsForm.submit();">
    <%= eMarketingSpot.getCollateral()[i].getMarketingText() %>
    </A>
   </TD>
<%      } %>
 </TR>
<%  } %>
 </TABLE>
<%  } %>

<%
 // The following block is used to display the categories associated with this e-marketing
 // spot.  The category display page which shows the selected category in the campaign will
 // be referenced through the submittion of the HTML form attached above.

 if (eMarketingSpot.getCategories() != null && eMarketingSpot.getCategories().length > 0) {
%>
 <TABLE>
<%  for (int i=0; i eMarketingSpot.getCategories().length; i++) { %>
  <TR>
   <TD>
    A HREF="/webapp/wcs/stores/servlet/ClickInfo?evtype=CpgnClick
&mpe_id=<% = eMarketingSpot.getId() %>&intv_id=<% = eMarketingSpot.
getCategories()[i].getInitiativeId()
 %>%URL=/webapp/wcs/stores/servlet/CategoryDisplay&
 <%= ECConstants.EC_STORE_ID %>= <%= emsCommandContext.getStoreId().toString() %>&
<% = ECConstants.EC_CATEGORY_ID %>=<% =
eMarketingSpot.getCategories()[i].getCategoryId() %>&<%
 = ECConstants.EC_CATALOG_ID %>=<%= eMarketingSpot.getCategories()[i].getCatalogId() %>
&<% = ECConstants.EC_LANGUAGE_ID %>=<% = emsCommandContext.getLanguageId().toString() %>">
    <%= eMarketingSpot.getCategories()[i].getDescription
(emsCommandContext.getLanguageId()).getName() %>
    </A>
  </TD>
   <TD><% = eMarketingSpot.getCategories()[i].getDescription
(emsCommandContext.getLanguageId()).getLongDescription() %> </TD>
  </TR>
<%  } %>
 </TABLE>
<%  } %>

<%
 // The following block is used to display the products associated with this e-marketing
 // spot.  The product display page which shows the selected product in the campaign will
 // be referenced through the submittion of the HTML form attached above.

 if (eMarketingSpot.getCatalogEntries() != null &&
eMarketingSpot.getCatalogEntries().length > 0) {
%>
 <TABLE>
 <%  for (int i=0; i eMarketingSpot.getCatalogEntries().length; i++) { %>
  <TR>
   <TD>
    A HREF="/webapp/wcs/stores/servlet/ClickInfo?evtype=CpgnClick
&mpe_id=<%= eMarketingSpot.getId() %>&intv_id=<%=
 eMarketingSpot.getCatalogEntries()[i].getInitiativeId()
 %>&URL=/webapp/wcs/stores/servlet/ProductDisplay
&<%= ECConstants.EC_STORE_ID %>=<%= emsCommandContext.getStoreId().toString()
 %>&<%= ECConstants.EC_PRODUCT_ID %>=<%=
eMarketingSpot.getCatalogEntries()[i].getCatalogEntryID() %>&<%
= ECConstants.EC_LANGUAGE_ID %>=<%= emsCommandContext.getLanguageId().toString() %>">
    IMG SRC="<%= collateralPath + eMarketingSpot.getCatalogEntries()
[i].getDescription(emsCommandContext.getLanguageId()).
getThumbNail() %>" ALT="<%= eMarketingSpot.getCatalogEntries()[i].getDescription
(emsCommandContext.getLanguageId()).getShortDescription()%>" BORDER=0 WIDTH=60>
```

```
     </A>
    </TD>
    <TD><%= eMarketingSpot.getCatalogEntries()[i].getDescription
(emsCommandContext.getLanguageId()).getShortDescription() %> </TD>
   </TR>
  <%  } %>
  </TABLE>
<%  } %>


<%  /**
   * END - the following code is used to drop multiple e-marketing spots onto the page.
   *       Customize the appropriate e-marketing spot name before use.
   *       Duplicate this code if more than 1 spot is needed, do not use the same spot name.
   */
%>
```

The preceding e-Marketing Spot supports three types of campaign initiatives:

- Product recommendation
- Category recommendation
- Awareness advertisement

Note: For more detailed information on each of these initiatives, see Chapter 12, "Campaign assets" on page 113.

e-MarketingSpot bean

e-Marketing Spots use the e-MarketingSpot bean to return the results of campaign initiatives that are currently scheduled onto the spot. Using different properties of the bean allows you to customize your e-Marketing Spot and the corresponding campaign initiative. For more information on the e-MarketingSpot bean and its properties, see the WebSphere Commerce online help.

Adding an e-Marketing Spot to your store pages

In order to add an e-Marketing Spot to your store pages, do the following:

1. Determine on which JSP files the spot will display. The spot can be added to multiple JSP files.
2. Determine where on the JSP file to place the spot.
3. Copy the sample e-Marketing Spot in "e-Marketing Spot" on page 295.
4. Paste the sample e-Marketing Spot to the desired location on your JSP file(s).
5. Customize the sample e-Marketing Spot to fit the layout of your JSP file(s).
6. Within the e-Marketing Spot code, give the e-Marketing Spot a name.

   Note: e-Marketing Spots should be descriptively named so as to include their location, such as HomePageAd, or CheckOutPageRecommendation. This helps to reduce confusion about where it will appear, and what content it should contain. If necessary, numbers can be added to the name to differentiate between two e-Marketing Spots appearing on the same page. e-Marketing Spot names must be valid Java identifiers. You must use this same name when registering the e-Marketing Spot in the database using the WebSphere Commerce Accelerator.
7. If you require more than one e-Marketing Spot per JSP file, repeat steps 2 through 6. When adding another e-Marketing Spot on a JSP file, ensure that you copy all of the second section of the sample e-Marketing Spot provided in "e-Marketing Spot" on page 295 and paste the entire second section on the JSP file again. Then, change the name of the e-Marketing Spot in the JSP file.

8. Register the e-Marketing Spot in the database using the WebSphere Commerce Accelerator. For detailed instructions, see the WebSphere Commerce online help.

   Note:

   a. If you plan to add the store ID, catalog ID, or language ID to the URL using the following convention, ″langId=<%= languageId %>″, note that the JSP in which the e-marketing spot is embedded must make the appropriate ID available. The IDs can also be retreived through the command context, for example, getCommandContext().getLanguageId()?).
   b. As a result of the structure of the NewFashion sample store, only products can be recommended through the e-Marketing spot, not items in stores based on the NewFashion sample store.
   c. The URL parameter, CatalogDisplay should start with ″&″ instead of ″?″ because the code isn′t referencing the command directly.

Part 9. Appendixes

Appendix A. UML legend

Unified Modeling Language is a standard graphical language for presenting different elements of software design. The following examples are some of the most common elements of UML. For further detail about formal specifications, refer to `http://www.rational.com` and `http://www.omg.org`.

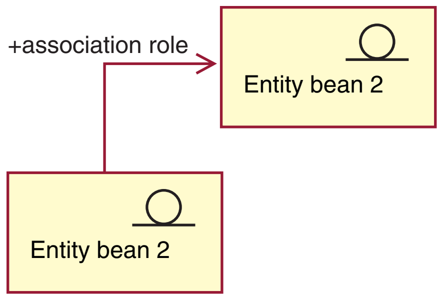
UML diagrams consist of the following items:

- Boxes: Boxes represent classes of objects. The class names appear at the top of the box. Attributes, if shown, appear below the class name. The class name and attributes are separated by a line.
- Lines: Lines represent possible relationships between objects of two classes. Objects of the class on one of end of the line can be ″associated″ with objects of the other class.
- Solid diamonds: Solid diamonds on the end of a line indicate containment by value. Objects of the class on the other end of the line are part of one and only one object of the class the diamond touches.
- Open diamonds: Open diamonds on the end of a line indicate containment by reference. Objects at the diamond end of the line can be thought of as grouping objects of the class at the other end of the line.
- Cardinality numbers: These appear at the end of relationship lines to indicate a cardinality restriction. The following table summarizes cardinality restrictions:

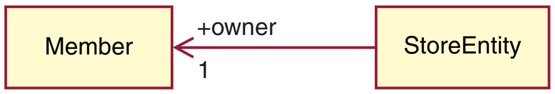
Cardinality number	Relationship type
1	one and only one
0..1	zero or one
0..n	zero or more
1..n	one or more

  If no cardinality restriction is shown, the cardinality is assumed to be 0..n, unless a solid diamond appears on the end of a relationship line. In that case, the cardinality must be 1.
- Plus signs: Plus signs appearing at the end of relationship lines indicate the object of the class at the end of the line plays a role in the relationship. Text following the plus sign indicates the object’s role in the relationship.
- Arrows: Arrows at the end of a relationship line indicate the direction of the relationship between two objects is in the direction of the arrow. The absence of any arrows on a relationship line indicates the direction of the relationship between the objects is normally in both directions.

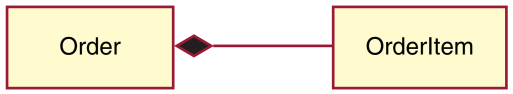
The following diagrams illustrate the above concepts:



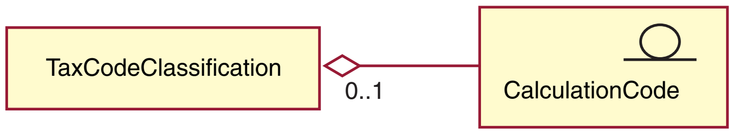
This diagram shows two entity beans with the decoration stereotype symbol indicating an Enterprise Java Bean. There is a unidirectional association from the first bean to the second entity bean. The plus sign is followed by text that describes what role Entity bean 2 plays the association.



In this diagram, a StoreEntity has one and only one owner, which is a Member. A Member may own zero or more StoreEntities. The plus sign indicates that the Member plays a role in the relationship. In this case the Member is the owner of the StoreEntity. The arrow indicates that you would normally find out the owner of a StoreEntity by asking the StoreEntity for its owner, and not asking a Member for all the StoreEntities it owns.



In this diagram, an OrderItem is always part of one and only one Order. An Order has zero or more OrderItems.



This diagram indicates that a CalculationCode is grouped by zero or one TaxCodeClassifications and a TaxCodeClassification groups zero or more CalculationCodes.

Appendix B. Creating your data

Before creating store data in the form of XML files, do the following:

- Determine the order of the information you are creating. The information in each of the store data chapters advises you on the order in which to create the data, but when creating XML files remember that information for a parent table must precede information for a child table.
- Determine how you want to use your store. If you are creating a sample store archive file (.sar), that is, a store archive that is meant to be copied and used as a base upon which to create new stores, you will need to create your data slightly differently than if you aren't creating a sample store. For more information, see "Creating data for sample stores".

Creating data for sample stores

Data in sample store archives takes the form of well-formed, XML files valid for the Loader package. 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, which are resolved by the ID Resolver at the time of publish. The use of these conventions allows the sample store archives to be copied and published multiple times.

It is not necessary to use these conventions when creating store data for your store in the form of XML files, unless you plan to create a sample store archive that will be used to generate several stores, or unless you want to create a store archive that is portable, that is a store archive that can be published to another WebSphere Commerce instance.

As a result, the sample store archives use the following conventions:

- & as in `member_id="&MEMBER_ID;"`The `&XXX;`convention is a DTD macro (known in XML as an entity).

  Note: You must define MEMBER_ID as a DTD macro when you are creating a sample store archive.

  WebSphere Commerce defines a set of macros in the following file:

  - NT `drive:\WebSphere\CommerceServer\xml\sar\DBLoadMacros.dtd`
  - 2000 `drive:\Program Files\WebSphere\CommerceServer\xml\sar\DBLoadMacros.dtd`
  - AIX `/usr/WebSphere/CommerceServer/xml/sar/DBLoadMacros.dtd`
  - Solaris `/opt/WebSphere/CommerceServer/xml/sar/DBLoadMacros.dtd`
  - Linux `/opt/WebSphere/CommerceServer/xml/sar/DBLoadMacros.dtd`
  - 400 `/qibm/proddata/WebCommerce/xml/sar/DBLoadMacros.dtd`

  Macros like en_US and es_ES are set to the appropriate language IDs. For example:

  ```
  <!ENTITY en_US "-1">
  ```

  The information will be specified using the tools in Store Services. For example, the user selects the MEMBER_ID in the Create Store Archive page in

Store Services. The MEMBER_ID macro is a place holder for the ID of the member who owns the store. When you create a store archive, you select a member to be the store owner. The MEMBER_ID macro is set to the member's ID. For example, if you select member ID -2000, then MEMBER_ID is set to -2000 as follows:

```
<!ENTITY MEMBER_ID "-2000">
```

- @ as in `ffmcenter_id="@ffmcenter_id_1"`. The use of the @ symbol is known as internal-alias resolution. The ID Resolver, which is a Loader package utility, generates identifiers for XML elements that require them. One of the techniques ID Resolver uses is internal-alias resolution. When using internal-alias resolution, an alias is substituted in place of the primary key (identifier) in the XML document. This alias is then used elsewhere in the XML file to refer to that element. This eliminates the need to know the unique indexes necessary to build the XML file. During publish in Store Services, or using the Loader package, the ID Resolver replaces the @ symbol with a unique value. See the following examples from an XML file:
  - Pre-ID Resolver

    ```
    <catalog
    catalog_id="@catalog_id_1"
    member_id="&MEMBER_ID;"
    identifer=InFashion"
    description="InFashion Catalog"/>
    ```
  - Post ID Resolver

    ```
    <catalog
    catalog_id="10001"
    member_id="-2000"
    identifer=InFashion"
    description="InFashion Catalog"/>
    ```

  where 10001 is the unique ID assigned by the ID Resolver and -2000 is the member ID selected by the user in Store Services. The resulting XML file then gets loaded using the Loader package. Running the files through the ID Resolver ensures that numerous stores can be created from a single set of XML files.

Store Services and sample stores

The **New** and Create Store Archive options in the Store Services are dependent on the conventions described above. If you want to use your store as a template to create other stores through Store Services, you must follow these conventions when creating your data assets.

However, if you are not creating a sample store archive, you can still edit or publish a store archive that does not follow these conventions using the tools in Store Services.

Appendix C. sarinfo.xml

Each store archive must include a sarinfo.xml file. This file, known as the descriptor, contains information about the store archive that is used when a store archive is published, including the names of the file asset ZIP files and the store database XML files, and the order in which they are published. If a store archive includes files in multiple languages, the sarinfo.xml file also includes that information, and determines the order in which each language file is published.

Note: The order in which the data assets are published is important, since some data assets must be published before others. As a result, the order of your assets, as specified in yoursarinfo.xml file, should match the order of the assets specified in the sarinfo.xml files for the sample stores. The sample store archives are located in the following directory.

- NT drive:\WebSphere\CommerceServer\samplestores
- 2000 drive:\Program Files\WebSphere\CommerceServer\samplestores
- AIX /usr/WebSphere/CommerceServer/samplestores
- Solaris /opt/WebSphere/CommerceServer/samplestores
- Linux /opt/WebSphere/CommerceServer/samplestores
- 400 /qibm/proddata/WebCommerce/samplestores

To view the content of the store archive, decompress it by using a ZIP program. The sarinfo.xml is located in the SAR-INF directory.

Example of sarinfo.xml

The following example is the ToolTech sarinfo.xml file. For more information on the elements, attributes and attribute values, see the information below. For more information on the XML specifications for a store archive, see the sarinfo.dtd 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
- Linux /opt/WebSphere/CommerceServer/xml/sar
- 400 /qibm/proddata/WebCommerce/xml/sar

```
<?xml version = "1.0"?>
<!DOCTYPE sarinfo SYSTEM "sarinfo.dtd">
<sarinfo complete-store="yes" multi-language="yes" version="1.0">

<store-info asset-name="store"/>

<file name="webapp.zip" type="zip">


<asset fragmented="no" name="webapp">
<file name="webapp.zip" type="zip">
<display-name>My Web App Display Name</display-name>
```

```
<description>My Web App</description>
</file>
</asset>

<asset fragmented="no" name="properties">
<file name="properties.zip" type="zip" />
</asset>

<asset fragmented="no" name="dbloadmacros">
<file name="data/DBLoadMacros.dtd" type="dtd"/>
</asset>

<asset fragmented="no" name="fulfillment">
<file name="data/fulfillment.dtd" type="dtd"/>
<file name="data/fulfillment.xml" priority="1" type="db-load"/>
<file name="data/en_US/fulfillment.xml" priority="31" type="db-load">
<locale>en_US</locale>
</file>
<file name="data/es_ES/fulfillment.xml" priority="31" type="db-load">
<locale>es_ES</locale>
</file>
</asset>

<asset fragmented="yes" name="store">
<file name="data/store.dtd" type="dtd"/>
<file name="data/store.xml" priority="2" type="db-load"/>
<file name="data/en_US/store.xml" priority="3" type="db-load">
<locale>en_US</locale>
</file>
<file name="data/es_ES/store.xml" priority="3" type="db-load">
<locale>es_ES</locale>
</file>
</asset>

<asset fragmented="yes" name="catalog">
<file name="data/catalog.dtd" type="dtd"/>
<file name="data/catalog.xml" priority="4" type="db-load"/>
<file name="data/en_US/catalog.xml" priority="5" type="db-load">
<locale>en_US</locale>
</file>
<file name="data/es_ES/catalog.xml" priority="5" type="db-load">
<locale>es_ES</locale>
</asset>

<asset fragmented="yes" name="tax">
<file name="data/tax.dtd" type="dtd"/>
<file name="data/tax.xml" priority="6" type="db-load"/>
<file name="data/en_US/tax.xml" priority="7" type="db-load">
<locale>en_US</locale>
</file>
<file name="data/es_ES/tax.xml" priority="7" type="db-load">
<locale>es_ES</locale>
</file>
</asset>

<asset fragmented="no" name="taxfulfill">
<file name="data/taxfulfill.dtd" type="dtd"/>
<file name="data/taxfulfill.xml" priority="8" type="db-load"/>
</asset>

<asset fragmented="yes" name="shipping">
<file name="data/shipping.dtd" type="dtd"/>
<file name="data/shipping.xml" priority="9" type="db-load"/>
file name="data/en_US/shipping.xml" priority="10" type="db-load">
<locale>en_US</locale>
/file>
file name="data/es_ES/shipping.xml" priority="10" type="db-load">
```

```
<locale>es_ES</locale>
<file>
</asset>

<asset fragmented="no" name="shippingfulfill">
<file name="data/shipfulfill.dtd" type="dtd"/>
<file name="data/shipfulfill.xml" priority="11" type="db-load"/>
</asset>

<asset fragmented="no" name="store-catalog">
<file name="data/store-catalog.dtd" type="dtd"/>
<file name="data/store-catalog.xml" priority="12" type="db-load"/>
</asset>

<asset fragmented="no" name="storefulfill">
<file name="data/storefulfill.dtd" type="dtd"/>
<file name="data/storefulfill.xml" priority="13" type="db-load"/>
</asset>

<asset fragmented="yes" name="offering">
<file name="data/offering.dtd" type="dtd"/>
<file name="data/offering.xml" priority="14" type="db-load"/>
</asset>

<asset fragmented="no" name="command">
<file name="data/command.dtd" type="dtd"/>
<file name="data/command.xml" priority="16" type="db-load"/>
</asset>

<asset fragmented="yes" name="currency">
<file name="data/currency.dtd" type="dtd"/>
<file name="data/currency.xml" priority="17" type="db-load"/>
<file name="data/en_US/currency.xml" priority="18" type="db-load">
<locale>en_US</locale>
</file>
<file name="data/es_ES/currency.xml" priority="18" type="db-load">
<locale>es_ES</locale>
</file>
</asset>

<asset fragmented="yes" name="campaign">
<file name="data/campaign.dtd" type="dtd"/>
<file name="data/campaign.xml" priority="20" type="db-load"/>

<file name="data/en_US/campaign.xml" priority="24" type="db-load">
<locale>en_US</locale>
</file>
<file name="data/es_ES/campaign.xml" priority="24" type="db-load">
<locale>es_ES</locale>
</file>
</asset>

<asset fragmented="no" name="store-catalog-tax">
<file name="data/store-catalog-tax.dtd" type="dtd"/>
<file name="data/store-catalog-tax.xml" priority="21" type="db-load"/>
</asset>

<asset fragmented="no" name="store-catalog-shipping">
<file name="data/store-catalog-shipping.dtd" type="dtd"/>
<file name="data/store-catalog-shipping.xml" priority="22" type="db-load"/>
</asset>

<asset fragmented="no" name="store-defaults">
<file name="data/store-defaults.dtd" type="dtd"/>
<file name="data/store-defaults.xml" priority="23" type="db-load"/>
</asset>
```

```
<asset fragmented="no" name="consistency_check">
<file name="data/sarrule.dtd" type="dtd"/>
<file name="data/sarrule.xml" priority="25" type="config"/>
</asset>

<asset fragmented="no" name="payment">
<file name="data/es_ES/paymentinfo.xml" type="config"/>
<file name="data/paymentinfo.dtd" type="dtd"/>
</asset>

<asset fragmented="yes" name="policy">
<file name="data/businesspolicy.dtd" type="dtd"/>
<file name="data/businesspolicy.xml" priority="26" type="db-load"/>
<file name="data/en_US/businesspolicy.xml" priority="27" type="db-load">
<locale>en_US</locale>
</file>
<file name="data/es_ES/businesspolicy.xml" priority="27" type="db-load">
<locale>es_ES</locale>
</file>
</asset>

<asset fragmented="no" name="organization">
<file name="data/organization.dtd" type="dtd"/>
<file name="data/organization.xml" priority="28" type="db-load"/>
</asset>


asset fragmented="no" name="businessaccount">
 <file name="data/businessaccount.xml" type="xml"/>
/asset>
asset fragmented="yes" name="contract">
 <file name="data/contract.xml" priority="1" type="xml"/>
 <file name="data/en_US/contract.xml" priority="2" type="xml">
<locale>en_US</locale>
</file>
file name="data/es_ES/contract.xml" priority="2" type="xml">
<locale>es_ES</locale>
</file>
</asset>

<asset fragmented="yes" name="accesscontrol">
<file name="data/accesscontrol.dtd" type="dtd"/>
<file name="data/accesscontrol.xml" priority="29" type="db-load"/>
<file name="data/en_US/accesscontrol.xml" priority="30" type="db-load">
<locale>en_US</locale>
</file>
<file name="data/es_ES/accesscontrol.xml" priority="30" type="db-load">
<locale>es_ES</locale>
</file>
</asset>

<!-- next priority should be 32 -->

</sarinfo>
```

where

`sarinfo` is all information contained in the `sarinfo.xml` file. The attributes in the following chart contain general information about the store archive.

Attribute name	Attribute value(s)
multi-language (required)	Determines whether multi-languages will be supported in this store archive: • yes • no
complete-store (required)	Determines whether the store archive includes the assets necessary for a complete store: • yes • no
version	The version of the store archive. For example: 1.0, 1.1
display-name	The name of the store archive.
description	A brief description of the store archive.
standard-schema (required)	Does the store archive follow the standard WebSphere Commerce database schema: • yes • no
store-info asset-name (required)	The asset that functions as the anchor for the store archive. All information for the store is found in the files belonging to this asset. For example: store
locale	The locale(s) supported by the store archive. The locale variables (listed below) are composed of the language_country: • de_DE • en_US • es_ES • fr_FR • it_IT • ja_JP • ko_KR • pt_BR • zh_CN • Zh_TW

`asset` (mandatory) An asset is a logical collection of related files. For example, tax is the name for the group of files that all relate to the stores taxes.

Attribute name	Attribute value(s)
name (required)	The name of the asset type. For example: • store • catalog • payment • tax

Attribute name	Attribute value(s)
fragmented (required)	Determines whether the asset information is split into multiple files according to languages. • yes • no

`file`

Attribute name	Attribute value(s)
name (required)	The name of the file.
type (required)	Type of file format. For example: • db-load - files to be loaded into the database • dtd - document type definition files • zip - ZIP file for file assets, for example, webapp.zip • config - configuration files
priority	The order that the files in the store archive will be published. 1,2,3,4 . . . **Note:** If two files share the same priority, the load order does not matter. If files must be loaded in a certain order, ensure they are assigned different priorites.
display-name	The name of the file.
description	Description for reference.
locale	The locale. The locale variables (listed below) are composed of the language_country. • de_DE • en_US • es_ES • fr_FR • it_IT • ja_JP • ko_KR • pt_BR • zh_CN • Zh_TW

Appendix D. sarrule.xml

Each store archive contains a `sarrule.xml` file, which acts a consistency checker when you publish using Store Services. While publishing, the publish utility uses the rules in the `sarrule.xml` file to check that your store archive contains the Web assets that are listed in the XML files.

Example of sarrule.xml

The following `sarrule.xml` file is from the ToolTech sample store.

The store archive files are located in the following directory:

- NT `drive:\WebSphere\CommerceServer\samplestores`
- 2000 `drive:\Program Files\WebSphere\CommerceServer\samplestores`
- AIX `/usr/WebSphere/CommerceServer/samplestores`
- Solaris `/opt/WebSphere/CommerceServer/samplestores`
- Linux `/opt/WebSphere/CommerceServer/samplestores`
- 400 `/qibm/proddata/WebCommerce/samplestores`

To view the `sarrule.xml` file in the store archive, decompress it using a ZIP program. The `sarrule.xml` file is located in the `data` directory.

The `sarrule.dtd` file is 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`
- Linux `/opt/WebSphere/CommerceServer/xml/sar`
- 400 `/qibm/proddata/WebCommerce/xml/sar`

Note: It is recommended that you use the `sarrule.xml`file provided with the sample stores in your store archive. However, if you want, you can add new rules to the existing `sarrule.xml` file.

```
<?xml version="1.0"?>
<!DOCTYPE SAR-rules SYSTEM "sarrule.dtd">
<SAR-rules>
 <asset name = "command">
  <check type="webasset registration">
   <rule entry="viewreg" attribute="properties" type="java.lang.String"
         removeStoreDir="false" file="webapp.zip"/>
  </check>
   </asset>
 !--
   <asset name = "catalog">
  <check type="webasset registration">
      <rule entry = "catalogdsc" attribute="thumbnail" type="java.lang.String"
    removeStoreDir="false" file="webapp.zip"/>
   <rule entry = "catalogdsc" attribute="fullimage" type="java.lang.String"
```

```
         removeStoreDir="false" file="webapp.zip"/>
   <rule entry = "catentdesc" attribute="thumbnail" type="java.lang.String"
         removeStoreDir="false" file="webapp.zip"/>
   <rule entry = "catentdesc" attribute="fullimage" type="java.lang.String"
         removeStoreDir="false" file="webapp.zip"/>
   <rule entry = "catgrpdesc" attribute="thumbnail" type="java.lang.String"
         removeStoreDir="false" file="webapp.zip"/>
   <rule entry = "catgrpdesc" attribute="fullimage" type="java.lang.String"
         removeStoreDir="false" file="webapp.zip"/>
  </check>
   </asset>
 -->
   <asset name = "store">
  <check type="webasset registration">
       <rule entry = "dispentrel" attribute="pagename" type="java.lang.String"
            removeStoreDir="false" file="webapp.zip"/>
       <rule entry = "dispcgprel" attribute="pagename" type="java.lang.String"
            removeStoreDir="false" file="webapp.zip"/>
  </check>
   </asset>
</SAR-rules>
```

where in

```
<asset name = "command">
  <check type="webasset registration">
   <rule entry="viewreg" attribute="properties" type="java.lang.String"
         removeStoreDir="false" file="webapp.zip"/>
  </check>
   </asset>
```

the publish utility in Store Services checks that each JSP file listed in the `command.xml` file exists in the Web assets in the store archive.

where in

```
   <asset name = "catalog">
  <check type="webasset registration">
      <rule entry = "catalogdsc" attribute="thumbnail" type="java.lang.String"
         removeStoreDir="false" file="webapp.zip"/>
   <rule entry = "catalogdsc" attribute="fullimage" type="java.lang.String"
         removeStoreDir="false" file="webapp.zip"/>
   <rule entry = "catentdesc" attribute="thumbnail" type="java.lang.String"
         removeStoreDir="false" file="webapp.zip"/>
   <rule entry = "catentdesc" attribute="fullimage" type="java.lang.String"
         removeStoreDir="false" file="webapp.zip"/>
   <rule entry = "catgrpdesc" attribute="thumbnail" type="java.lang.String"
         removeStoreDir="false" file="webapp.zip"/>
   <rule entry = "catgrpdesc" attribute="fullimage" type="java.lang.String"
         removeStoreDir="false" file="webapp.zip"/>
  </check>
    </asset>
```

the publish utility in Store Services checks that each catalog asset listed in the `catalog.xml` file exists in the Web assets in the store archive.

where in

```
<asset name = "store">
  <check type="webasset registration">
       <rule entry = "dispentrel" attribute="pagename" type="java.lang.String"
            removeStoreDir="false" file="webapp.zip"/>
       <rule entry = "dispcgprel" attribute="pagename" type="java.lang.String"
            removeStoreDir="false" file="webapp.zip"/>
  </check>
   </asset>
```

the publish utility in Store Services checks that each store asset listed in the `store.xml` file exists in the Web assets in the store archive.

If memory is an issue while publishing, comment out the catalog asset rule in `sarrule.xml` before attempting to publish.

Appendix E. Database asset groups

All WebSphere Commerce database assets are divided into groups for creation and loading. These groups are a logically related set of tables. The order in which these database asset groups are organized is important to data loading, since certain objects must exist before loading the relationship between objects.

When loading database assets in XML format for your store, you can choose to load only selected groups. These groups consist of the database assets created in the previous chapters, such as catalog or fulfillment. Before loading data groups as instructed in "Loading database asset groups" on page 267, do the following:

- Determine which database asset group you are loading. Each group contains dependencies which must be met before the assets can be loaded. Review the information in "Database asset groups dependencies".
- Ensure that you have created or updated the XML files for the selected database asset group. The information in each of the asset chapters advises you on the order in which to create the database assets, but when creating or updating XML files, remember that information for a parent table must precede information for a child table.

Database asset groups dependencies

Each database asset group draws its information from WebSphere Commerce database tables. Database assets have dependencies within their own group. That is, a database asset group cannot draw data in other XML files from a different data group, and each group is independent minus the foreign keys. However, if the database asset group needs to refer to the external data defined in another group, then you need to provide that data manually. This means that the data from one group has an *external dependency* on data defined outside of its domain, that is, on another database asset group. External dependencies occur when a database asset group has a foreign key relationship to the primary key of a table in another group. To load a database asset group, its external dependencies must be satisfied. To use an example from the chart below, one of the external dependencies for the store database asset group is `fulfillment.FFMCENTER.FFMCENTER_ID`, which indicates that the fulfillment database assets must already exist in the WebSphere Commerce database before you can load the store database asset group.

Consider the following chart before you begin your loading process. Each group of database assets is dependent on other database tables, from which the data is loaded.

Some points to remember:

- Some external dependencies may not be satisfied by a single group. Site wide or general database assets, used by every store, are pre-populated at instance creation in the *bootstrap* and can be readily accessed. Tables contained in database asset groups have foreign key references to this type of data. Bootstrap data is divided into common and locale-specific data. If you have a multilingual store, you need to choose the common and the locale-specific bootstrap data. For example, you need the language and member bootstrap data. The instance creation process populates the LANGUAGE table with the supported WebSphere Commerce languages for your store and creates a root organization (`MEMBER.MEMBER_ID=-2001`) and a default organization (`MEMBER.MEMBER_ID=-2000`).

You must use the root organization where required, but you should create a store owner organization instead of using the default organization. For more information about organizations and their hierarchy, refer to the WebSphere Commerce online help.

- The files listed under the **External dependencies** column use the following naming structure: *database asset group*.*database table*.*database column*. Using the `store.STOREENT.STOREENT_ID` file as an example, the data is taken from the *store* database asset group, *STOREENT* table, and *STOREENT_ID* column. File names beginning with *bootstrap* indicate that the data was populated during the WebSphere Commerce instance creation.
- The files listed under the **External dependencies** column contain foreign key references to the **Related tables**. These tables must be populated first.
- For presentation purposes only, the tables have been split to indicate the locale-specific tables containing multilingual information, such as product descriptions.
- The tables in the chart represent the database assets from the WebSphere Commerce sample stores. The tables may vary according to your store's size, function, and needs. Depending on your store's requirements, ensure that you include all database tables containing your store's assets, even if that particular table is not listed below.

Access control database assets		
External dependencies	**Related tables from the database asset XML files**	**Related locale-specific tables from the database asset XML files**
`bootstrap.LANGUAGE.LANGUAGE_ID` (root and store owner organizations), `bootstrap.MEMBER.MEMBER_ID` (root and store owner organizations)	**`accesscontrol.xml`** ACACTACTGP, ACACTGRP, ACACTION, ACPOLICY, ACRESCGRY, ACRESGPRES, ACRESGRP	**`accesscontrol.xml`** ACACGPDESC, ACACTDESC, ACPOLDESC, ACRSCGDES, ACRESGPDES
Business policy database assets		
External dependencies	**Related tables from the database asset XML files**	**Related locale-specific tables from the database asset XML files**
`bootstrap.LANGUAGE.LANGUAGE_ID`, `boostrap.MEMBER.MEMBER_ID`, `store.STOREENT.STOREENT_ID` (store owner organization)	**`businesspolicy.xml`** POLICY, POLICYCMD	**`businesspolicy.xml`** POLICYDESC
Campaign database assets		
External dependencies	**Related tables from the database asset XML files**	**Related locale-specific tables from the database asset XML files**
`store.STOREENT.STOREENT_ID`	**`campaign.xml`** CAMPAIGN, COLLATERAL, EMSPOT, STENCALUSG	**`campaign.xml`** COLLDESC
Catalog database assets		

External dependencies	Related tables from the database asset XML files	Related locale-specific tables from the database asset XML files
`bootstrap.LANGUAGE.LANGUAGE_ID`, `bootstrap.MEMBER.MEMBER_ID` (store owner organization), `store.STOREENT.STOREENT_ID`, `shipping.CALCODE.CALCODE_ID`, `tax.CALCODE.CALCODE_ID`	**`catalog.xml`** BASEITEM, CATALOG, CATENTREL, CATENTRY, CATGROUP, CATGRPREL, CATTOGRP, ITEMSPC, ITEMVERSN, RA, RADETAIL, STOREITEM, STORITMFFC, VERSIONSPC **`offering.xml`** CATGRPTPC, MGPTRDPSCN, OFFER, OFFERPRICE, TRADEPOSCN **`storefulfill.xml`** INVENTORY **`store-catalog.xml`** DISPCGPREL, DISPENTREL, STORECAT, STORECENT, STORECGRP **`store-catalog-shipping.xml`** CATENTCALCD, CATENTSHIP **`store-catalog-tax.xml`** CATENTCALD	**`catalog.xml`** ATTRIBUTE, ATTRVALUE, BASEITMDSC, CATALOGDSC, CATENTDESC, CATGRPDESC, PKGATTR, PKGATTRVAL,
Command database assets		
External dependencies	**Related tables from the database asset XML files**	**Related locale-specific tables from the database asset XML files**
`store.STOREENT.STOREENT_ID`	**`command.xml`** CMDREG, VIEWREG	N/A
Business **Contract database assets**		
External dependencies	**Related tables from the database asset XML files**	
`store.STOREENT.STOREENT_ID`	The contract database tables are not loaded directly and follow a different process than the other WebSphere Commerce data groups. Refer to “Publishing contract assets” on page 275for more information.	
Currency database assets		

External dependencies	Related tables from the database asset XML files	Related locale-specific tables from the database asset XML files
store.STOREENT.STOREENT_ID	**currency.xml** CURCVLIST	**currency.xml** CURCONVERT, CURLIST
Fulfillment database assets		
External dependencies	**Related tables from the database asset XML files**	**Related locale-specific tables from the database asset XML files**
bootstrap.LANGUAGE.LANGUAGE_ID, boostrap.MEMBER.MEMBER_ID (store owner organization)	**fulfillment.xml** FFMCENTER, STADDRESS	**fulfillment.xml** FFMCENTDS
Organization database assets		
External dependencies	**Related tables from the database asset XML files**	**Related locale-specific tables from the database asset XML files**
bootstrap.LANGUAGE.LANGUAGE_ID (root and store owner organizations), boostrap.MEMBER.MEMBER_ID (root and store owner organizations)	**organization.xml** ADDRBOOK, ADDRESS, MBRREL, MEMBER, ORGENTITY	N/A
Shipping database assets		
External dependencies	**Related tables from the database asset XML files**	**Related locale-specific tables from the database asset XML files**
bootstrap.LANGUAGE.LANGUAGE_ID, bootstrap.MEMBER.MEMBER_ID (store owner organization), fulfillment.FFMCENTER.FFMCENTER_ID, store.STOREENT.STOREENT_ID	**shipping.xml** CALCODE, CALRULE, CRULESCALE, JURST, JURSTGPREL, JURSTGROUP, SHIPMODE, STENCALUSG **shipping.xml** SHPJCRULE, SHPARRANGE	**shipping.xml** CALCODEDSC, CALRANGE, CALRLOOKUP, CALSCALE, SHPMODEDSC
Store database assets		
External dependencies	**Related tables from the database asset XML files**	**Related locale-specific tables from the database asset XML files**
bootstrap.LANGUAGE.LANGUAGE_ID, bootstrap.MEMBER.MEMBER_ID (store owner organization), bootstrap.SETCURR.SETCURR_ID, fulfillment.FFMCENTER.FFMCENTER_ID	**store.xml** INVADJCODE, RTNREASON, STORE, STORENT, STORELANG, VENDOR	**store.xml** FFMCENTDS, INVADJDESC, RTNRSNDESC, STADDRESS, STOREENTDS, STORLANGDS, VENDORDESC
Store default database assets		

External dependencies	Related tables from the database asset XML files	Related locale-specific tables from the database asset XML files
`shipping.SHIPMODE.SHIPMODE_ID` (if applicable, this file must be loaded first), `contract.CONTRACT.CONTRACT_ID`, `store.STOREENT.STOREENT_ID`	**`store-default.xml`** STOREDEF	N/A
Tax database assets		
External dependencies	**Related tables from the database asset XML files**	**Related locale-specific tables from the database asset XML files**
`bootstrap.LANGUAGE.LANGUAGE_ID`, `boostrap.MEMBER.MEMBER_ID` (store owner organization), `store.STOREENT.STOREENT_ID`, `fulfillment.FFMCENTER.FFMCENTER_ID`, `store.STOREENT.STOREENT_ID`	**`tax.xml`** CALCODE, CALRANGE, CALRLOOKUP, CALRULE, CALSCALE, CRULESCALE, JURST, JURSTGROUP, JURSTGPREL, STENCALUSG, TAXCGRY, TAXJCRULE **`taxfulfill.xml`** TAXJCRULE	**`tax.xml`** CALCODEDSC, CALSCALEDS, TAXCGRYDS

Appendix F. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

©Copyright International Business Machines Corporation 2001. Portions of this code are derived from IBM Corp. Sample Programs. ©Copyright IBM Corp. 2000, 2001. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Credit card images, trademarks, and trade names provided in this product should be used only by merchants authorized by the credit card mark's owner to accept payment via that credit card.

Trademarks

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

AIX	IBM	WebSphere
AS/400	IBM Payment Manager	
DB2	iSeries	
DB2 Universal Database™	OS/400®	
eServer	VisualAge	

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

Oracle is a registered trademark and Oracle8 is a trademark of Oracle Corporation.

SET Secure Electronic Transaction, SET™ and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC. Use of the trademarks without a written license from SET Secure Electronic Transaction LLC is strictly prohibited.

Solaris, Solaris Operating Environment, Java, JavaServer Pages, JavaBeans, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc.

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

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

Printed in U.S.A.