IBM® WebSphere Commerce

Store Development Guide

Version 5.5

IBM® WebSphere Commerce

Store Development Guide

Version 5.5

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

First Edition, First Revision (September 2003)

This edition applies to IBM WebSphere Commerce Business Edition Version 5.5, IBM WebSphere Commerce - Express Version 5.5, and IBM WebSphere Commerce Professional Edition Version 5.5 (product number 5724-A18), and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition also applies 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 using the online IBM WebSphere Commerce documentation feedback form, available at the following URL:

`http://www-3.ibm.com/software/genservers/commerce/rcf.html`

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Before you begin

The *IBM® WebSphere® Commerce Store Development 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 process
- Business models supported by WebSphere Commerce
- WebSphere Commerce architecture
- Developing your storefront
- Developing your store data
- Store data architecture
- Store data information model
- Globalizing your store
- Adding access control to your store
- Packaging your store
- Publishing your store
- Adding WebSphere Commerce features to your store

Conventions and terminology 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, icons, or menu choices.
`Monospace type`	Indicates examples of text you enter exactly as shown, file names, and directory paths and names.
Italic type	Used to emphasize words. Italics also indicate names for which you must substitute the appropriate values for your system.

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

Important
These sections highlight especially important information.

Note
These sections highlight significant information.

▶Business	Indicates information specific to WebSphere Commerce Business Edition.
▶Professional	Indicates information specific to WebSphere Commerce Professional Edition.

Express	Indicates information-specific to WebSphere Commerce - Express.
Developer	Professional Business Indicates information specific to the WebSphere Commerce development environment. The development environment is WebSphere Commerce Studio, Version 5.5. Express Indicates information specific to the WebSphere Commerce development environment. The development environment is WebSphere Commerce - Express Developer Edition, Version 5.5.
AIX	Indicates information specific to programs running on AIX®.
400	Indicates information specific to programs running on OS/400®.
Linux	Indicates information that is specific to WebSphere Commerce for Linux for xSeries™, information that is specific to WebSphere Commerce for Linux for Eserver zSeries™ and S/390®, information that is specific to WebSphere Commercefor Linux for Eserver iSeries™, and information that is specific to WebSphere Commerce for Linux for Eserver pSeries™.
Solaris	Indicates information specific to programs running on Solaris Operating Environment.
2000	Indicates information specific to programs running on Windows® 2000.
DB2	Indicates information specific to DB2 Universal Database™.
Oracle	Indicates information specific to Oracle9i Database.

Variables used in this book

Some of the key variables in this book are as follow:

businessmodel
The name of the sample business model with which you are working (for example, consumer direct or B2B direct).

Professional Business *cell_name*
Cells are arbitrary, logical groupings of one or more nodes in a WebSphere Application Server distributed network that are managed together. In this definition, a node is a single WebSphere Application Server. One or more cells managed by a single-occurrence of WebSphere Application Server deployment manager are called a WebSphere Application Server deployment manager cell.

Express *cell_name*
In WebSphere Commerce - Express, cell_name is equal to host_name.

host_name
This variable represents the fully qualified host name of your WebSphere Commerce Server (for example, `server.mydomain.ibm.com` is fully qualified).

instance_name
: This variable represents the name of the WebSphere Commerce instance with which you are working (for example, mall1).

storedir
: This variable represent the name of the store directory in which your store is located.

WAS_instance_name
: This variable represents the name of the WebSphere Application Server with which your WebSphere Commerce instance is associated.

Path variables

This guide uses the following variables to represent directory paths:

WC_installdir
: This is the installation directory for WebSphere Commerce. The following are the default installation directories for WebSphere Commerce on various operating systems:

AIX /usr/WebSphere/CommerceServer55

400 /QIBM/ProdData/CommerceServer55

Linux /opt/WebSphere/CommerceServer55

Solaris /opt/WebSphere/CommerceServer55

Windows C:\Program Files\WebSphere\CommerceServer55

WCDE_installdir
: The installation directory for the WebSphere Commerce development environment. For WebSphere Commerce Business Edition and WebSphere Commerce Professional Edition, your development environment is WebSphere Commerce Studio, Version 5.5. The following is the default installation directory:
C:\WebSphere\CommerceStudio55.

For WebSphere Commerce - Express, the development environment is WebSphere Commerce - Express Developer Edition, Version 5.5. The following is the default installation directory:
C:\WebSphere\CommerceDev55

400 *WC_userdir*
: This is the directory for all the data that is used by WebSphere Commerce which can be modified or needs to be configured by a user. An example of such data is WebSphere Commerce instance information. This directory is unique to OS/400.

The *WC_userdir* variable represents the following directory:
`/QIBM/UserData/CommerceServer55`

WAS_installdir
: This is the installation directory for WebSphere Application Server. The following are the default installation directories for WebSphere Application Server on various operating systems:

AIX	/usr/WebSphere/AppServer
400	/QIBM/ProdData/WebAS5/Base
Linux	/opt/WebSphere/AppServer
Solaris	/opt/WebSphere/AppServer
Windows	C:\Program Files\WebSphere\AppServer

400 *WAS_userdir*
This is the directory for all the data that is used by the WebSphere Application Server that can be modified or needs to be configured by a user. An example of such data is WebSphere Application Server instance information. This directory is unique to OS/400.

The *WAS_userdir* variable represents the following directory: /QIBM/UserData/WebAS5/Base/*WAS_instance_name*

WC_userdir

The *WC_userdir* variable represents the following directory: /QIBM/UserData/WebAS5/Base/*WAS_instance_name*

workspace_dir
Used in the development environment. The variable represents *drive*:\WebSphere\workspace_db2

Where to find new information

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

http://www.ibm.com/software/commerce/library/

Updates may include new information.

Part 1. Overview

Chapter 1. Store development overview

This chapter provides an overview of the site or store development process in WebSphere Commerce, and introduces many of the concepts discussed in this guide.

Note: This guide uses the phrase *store development* to refer both to the processes involved in creating a single store, and the processes involved in creating a multi-store or site environment.

Understanding store development in WebSphere Commerce

Before starting to develop your site or store with WebSphere Commerce you need to understand how the following factors affect the store development process. Each of these factors is introduced in this chapter, but in most cases are explained in more detail throughout this guide, and in some cases in other documents in the WebSphere Commerce library.

How you choose to develop your store in WebSphere Commerce depends on the following factors:

- The purpose of your store
- The representative business model for your store
- The number of stores being developed and their types
- The foundation for your store
- The degree of required customization

The purpose of your store

Stores are usually developed for one of the following purposes:

- Production: Production stores are fully functional stores in a production environment, ready for use by customers or partners.
- Demo: Demo stores demonstrate certain capabilities for sales purposes. Demo stores may be only partially functional.
- Sample: Sample stores are fully functional stores that are designed to be used as a base on which you create your online store.

The representative business model for your store

Before developing your store, you need to understand which of the business models supported by WebSphere Commerce best represents your store. WebSphere Commerce supports sites or stores that are an instance of one of the following business models:

Note: These business models are discussed in more detail in Chapter 2, “Supported business models in WebSphere Commerce,” on page 15, but a brief introduction of each business model is provided here.

- Direct sales business model: As in previous releases, WebSphere Commerce supports the direct sales business model. Using WebSphere Commerce you can create sites or stores that support commerce transactions involving products, services, or information directly between businesses and consumers or between two businesses or parties. WebSphere Commerce supports the following types of direct sales business models:

  - Consumer direct business model: Consumer direct supports commerce transactions involving products, services, or information between businesses and consumers. Consumers typically purchase goods or services directly from a business in a consumer direct scenario.
  - Business B2B direct business model: B2B direct supports commerce transactions involving products, services, or information between two businesses or parties. Typical B2B direct transactions occur among buyers, suppliers, manufacturers, resellers, distributors, and trading partners.
- Business Hosting business model: WebSphere Commerce also supports hosting of merchants or other businesses by an Internet Service Provider or other hosting provider.
- Business Value chain business model: Value chains support transactions involving multiple enterprises or parties. Products, goods, services, or information are delivered through the parties of the value chain from producers to end users. A value chain also has relationship and administrative aspects, that is, you can manage the relationship of the partners or enterprises in your value chain, as well as offer some administrative services to those parties. WebSphere Commerce supports the transactions and relationship management of the following two types of value chains:
  - Demand chains: Demand chains support both indirect sales channels and direct sales channels.
  - Supply chains: Supply chains support procurement and sourcing of goods. WebSphere Commerce supports sourcing of goods through private marketplaces. A private marketplace provides a forum for vendors to offer their goods and services for sale to buyers with whom they have contractual relationships.

Note: These business models are discussed in more detail in Chapter 2, “Supported business models in WebSphere Commerce,” on page 15.

The number of stores being developed

Depending on your business, you may need to develop more than one store or more than one type of store. For example, if your business sells directly to customers, you may only need one store, which your customers access and purchase goods from. However if you are supporting your demand chain, you may need one main hub store for your business, and several stores that allow you to connect to or administer your channels. You may also choose to host stores for the organizations or businesses in your channels. For more information on the demand chain, see Chapter 2, “Supported business models in WebSphere Commerce,” on page 15.

If you are in the business of hosting stores for merchants or other businesses, you will also need to develop a hub store for managing merchants and handling registration requests, and a method to develop sites for those you are hosting. For more information on the hosting business model, see Chapter 2, “Supported business models in WebSphere Commerce,” on page 15.

You can develop numerous types of stores as well as multiple stores per site with WebSphere Commerce. For more details on stores types, see Chapter 7, “Store architecture,” on page 63.

The foundation for your store

Before creating a site or store with WebSphere Commerce you must decide where you want to start development. WebSphere Commerce offers several samples that you can use as the starting point for development, or you can choose to start from scratch. For more details about the samples provided with WebSphere Commerce, see the *WebSphere Commerce Sample Store Guide*.

Starting from a sample

The samples provided with WebSphere Commerce are packaged as store archives.

The store archive: A store archive file (.sar) is a ZIP archive file that contains all the assets necessary to create a site or store. It is primarily used as a vehicle for packaging and delivering 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.
- Property resource bundles: Contains the text for your store pages. If your store supports more than one language, the store archive will contain multiple resource bundles, one per supported language, plus a default resource bundle (which does not include a locale). For example, `AddressText_en_US.properties` and `AddressText.properties`.
- 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 6, "Developing your store data," on page 107.

  The store data 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 enables the portability of the sample store archives. For more information, see Part 9, "Packaging your store," on page 311.

  For more information on the Loader package, see Chapter 37, "Overview of loading store data," on page 335.

  Note: Store data assets also include the information to create a contract. The contract information is not loaded through the Loader package; it provides input to a command that creates contracts.
- Payment assets: Configuration information for WebSphere Commerce Payments. The payment information is not loaded through the Loader package; it provides input to a command that configures WebSphere Commerce Payments.
- Descriptors: XML files that describe the store archive and information on how it should be published. These files include `store-refs.xml`, `ibm-wc-load.xml`, `unpack.xml`, and `ForeignKeys.dtd`.

For more information on the store archive, see Chapter 35, "Packaging a store," on page 313.

Publishing a store archive*:* You can publish a store archive using either the publish utility in the Administration Console, or through the command line. For more details on how to publish a store archive, see the WebSphere Commerce Production online help, topic "Publishing a store archive".

Types of samples: The samples provided with WebSphere Commerce are categorized as follows:

- Composite stores archives
- Business Component store archives
- Basic store archive

Composite store archives: A composite store archive contains all the necessary assets to create a working site. The sample composite store archives provided with WebSphere Commerce usually contain the organization structure, predefined user roles, and necessary access control policies to create the appropriate environment for the corresponding business models. Composite store archives also contain the necessary assets to create the stores or sites needed. For example, the demand chain sample composite store archive contains a sample channel hub site, shared catalog, and reseller and distributor stores.

WebSphere Commerce includes several composite store archives that contain fully functional online sample sites that you can use as the basis for creating your own store. These samples, which include direct sales stores (both consumer direct and Business B2B direct), Business a demand chain business, Business a supplier business, and Business a hosting site, 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 samples provided with WebSphere Commerce, see the *WebSphere Commerce Sample Store Guide*.

Why start with a sample composite store archive?: Starting with a sample composite store archive loads all of the necessary data into the WebSphere Commerce Server to create a fully functional site.

WebSphere Commerce requires that certain data be loaded into the WebSphere Commerce Server database to create a functional site, and that this data be loaded in the order determined by the schema. Since the sample component store archives 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 site saves you a substantial amount of time during the initial creation period.

After publishing a sample composite store archive, you can edit it a lot or a little, depending on your store 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 the development environment. 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 flow and features. Or 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. For more information on editing store data, see Part 6, "Developing your store data," on page 107.

Component store archives: Business Each of the parts that make up the composite store archive are also available as separate store archives. These store archives are known as component store archives. A component store archive may be an organization structure store archive, which contains the organization structure and predefined user roles, or it may be a functioning store, or it may be a collection of file or data assets that can be used as resources by other types of stores. For more information about the samples provided with WebSphere Commerce, see the *WebSphere Commerce Sample Store Guide*.

Why start with a sample component store archive?: Starting with a sample component store archive, or a combination of sample component store archives provides you with more flexibility than starting with a sample composite store archive, as publishing a composite store archive creates a fully functional site. Parts of this site may be appropriate for your needs, but other parts may not. 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 choose to publish only certain parts of a provided sample, rather than the entire sample. For example, you may choose to publish only the sample organization structure, and then develop all of the assets to create the stores in your site. Or you may choose to publish a sample organization structure, and one or more of the sample component archives that create either a store in the organization structure, or provide resources to be used by other stores.

Note: If you are creating an instance of a value chain business model, it is recommended that you start by publishing the sample organization structure, as the organization structure that is needed for sites that contain multiple entities is quite complex. For more information on how organization structures work in WebSphere Commerce, see "Understanding the WebSphere Commerce organization structure" on page 25.

Basic store archive: WebSphere Commerce also provides a basic sample store that provides the minimal set of assets needed to create a store in the WebSphere Commerce Server.

Why start with the sample basic store?: Starting with the sample basic store allows you to establish a store entity in the commerce server, in that the JSP files can be invoked using the store ID. If you are creating a store that is very different than any of the samples stores provided with WebSphere Commerce, you may want to start with the sample basic store, as starting with the basic store allows a developer to add assets as necessary, and does not require that you remove or change assets which are not applicable to your store. For more information about the basic store provided with WebSphere Commerce, see the *WebSphere Commerce Sample Store Guide*.

Note: You can use the sample basic store in conjunction with one of the sample organization structures provided.

Starting from scratch

It is also possible to start from scratch, that is not use any of the samples provided with WebSphere Commerce.

The degree of required customization

Once you have decided on the foundation for your store, whether it be a sample store, a sample organization structure or sample basic store, or from scratch, you need to determine what types of changes you will make to it. In general, most store development in WebSphere Commerce falls into one of the following categories:

- Adding or changing store functionality, including adding new features or changing the store flow
- Creating or changing the look and feel of a store
- Creating or changing store data

In many cases, your store development effort will include a combination of all three.

Adding or changing store functionality

Adding or changing store functionality, including changing the flow of your store, or adding new features to your store, usually necessitates changes in business logic. Tools for developing the business logic, including creating and extending commands, creating customized code, and implementing business logic are discussed in the *WebSphere Commerce Programming Guide and Tutorials*.

Note: Developers who are creating or changing the business logic must have programming skills in Java™, Enterprise JavaBeans™, WebSphere Studio Application Developer, J2EE programming, and be familiar with the WebSphere Commerce programming model and object model.

WebSphere Commerce Accelerator provides the ability to change some of the features and store flows provided with the consumer direct and B2B direct sample stores. For more information on what flows and functionality you can change and how, see the WebSphere Commerce Production online help.

Creating or changing the look and feel of a store

Changing the look and feel of a store usually involves changing the storefront. Storefront assets include Web assets such as HTML pages, JSP files, style sheets, images, graphics and other multimedia file types. Developing your storefront 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 storefront assets:

- WebSphere Studio Application Developer

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

  For more information on using the tools in WebSphere Commerce Studio to create and edit your storefront assets, see the WebSphere Commerce Studio online help. For more information on creating your storefront in WebSphere Commerce, see Chapter 8, “Developing your storefront,” on page 75.

  Note: Developers who are creating or changing the storefront must have programming skills in Java, JavaScript™, HTML, JSP technology, and be familiar with the WebSphere Commerce store architecture.
- WebSphere Commerce Accelerator

WebSphere Commerce Accelerator includes the following tools to change the look and feel of your store:

– Change Pages notebook
– Upload Logo notebook
– Change Style wizard
– Manage Files notebook
– Store Profile notebook

For more information these tools, see the WebSphere Commerce Production online help.

Note: The tools listed above only work with stores based on the consumer direct sample store, and Business hosted stores (in the hosting model and demand chain model) created with the Store Creation wizard.

Creating or changing store data

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

- 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 Chapter 37, "Overview of loading store data," on page 335.

  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. The Loader package can also be used to automate a regular data feed from a back end system.

- 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 create or edit data, you can use also use it as a store development tool, particularly when you are changing small amounts of data. For a list of the database assets you can edit with the WebSphere Commerce Accelerator, see Part 6, "Developing your store data," on page 107.

  When to use WebSphere Commerce Accelerator: Use the WebSphere Commerce Accelerator when you are to create or update data.

- Editing the database directly

  You always have the option of editing the database directly using SQL inserts, updates or deletes.

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.

Developers who are creating or changing the store data must be familiar with the WebSphere Commerce store architecture, store data and store archives. To modify and extend the WebSphere Commerce database schema for the purpose of implementing customized store functions, or integrating with existing database information, the developer should have database administrator skills for DB2® or Oracle.

Scenario: Developing and deploying a production store

This section outlines the recommended scenario for developing a production store with WebSphere Commerce.

Table 1. Scenario: Developing and deploying a production store

Task	*Subtasks*	*Reference*
Determine which supported business model reflects your business		Chapter 2, “Supported business models in WebSphere Commerce,” on page 15
Determine the store flow		Part 4, “Developing your storefront,” on page 73
Create use cases		Part 4, “Developing your storefront,” on page 73
Analyze sample stores provided with WebSphere Commerce		*WebSphere Commerce Sample Store Guide*
Determine which sample store or other sample to use as a starting point		*WebSphere Commerce Sample Store Guide*
Create a baseline set of store assets	Create a project in the development environment (Specific to the WebSphere Commerce development environment)	WebSphere Studio product documentation
	Publish one of the sample store archives in the development environment	WebSphere Commerce Production online help, help topic ″Publishing a store archive″
	If possible, configure the store using the change flow tooling in the WebSphere Commerce Accelerator	WebSphere Commerce Production online help, help topic ″Changing store flows using WebSphere Commerce Accelerator″
	Make any necessary database schema changes	*WebSphere Commerce Programming Guide and Tutorials*
	Check store assets into a source control system creating a master copy	*WebSphere Commerce V5.5 Customization and Deployment Handbook* SG24-6969 Redbook.

Table 1. Scenario: Developing and deploying a production store (continued)

Determine the development that must be done to create store from baseline assets (storefront, data and server development)	Determine changes to look and feel of store	*WebSphere Commerce Sample Store Guide* Part 4, “Developing your storefront,” on page 73
	Determine caching strategy for store pages	Part 4, “Developing your storefront,” on page 73
	Determine changes to store data	*WebSphere Commerce Sample Store Guide* Part 6, “Developing your store data,” on page 107
	Understand the implementation used in the sample store	*WebSphere Commerce Sample Store Guide*
	Analyze existing server functionality to determine where it will need enhancements or customization	WebSphere Commerce Production and Development online help
	Determine the required degree of integration with back end systems	WebSphere Commerce Production and Development online help
Set up team environment	Each developer sets up a development project in the IDE, populates the project with assets from the master copy in the source control system and the server assets shipped with the product	*WebSphere Commerce V5.5 Customization and Deployment Handbook* SG24-6969 Redbook.
	Developer gets baseline set of assets running	
	Team familiarizes themselves with the existing functionality in the store from the customer’s and administrator’s point of view	*WebSphere Commerce Sample Store Guide*
Develop the store assets	Modify and enhance the storefront assets or create new storefront assets	Part 4, “Developing your storefront,” on page 73
	Develop additional server functionality (writing new commands, EJBs, integrating with back end systems)	*WebSphere Commerce Programming Guide and Tutorials*
	Modify the data and create additional data	Part 6, “Developing your store data,” on page 107
Create production-ready data		Part 6, “Developing your store data,” on page 107

Table 1. Scenario: Developing and deploying a production store (continued)

Deploy developed assets into production		*WebSphere Commerce Programming Guide and Tutorials* Additional information can also be found in *WebSphere Commerce V5.5 Customization and Deployment Handbook*, SG24-6969 Redbook.

Part 2. Business models supported by WebSphere Commerce

Chapter 2. Supported business models in WebSphere Commerce

Before starting to develop your store or site with WebSphere Commerce you need to understand what business models WebSphere Commerce supports. Most stores you create with WebSphere Commerce will be an instance of one of these business models.

Note: You can also create stores with WebSphere Commerce that do not conform to the business models described in this chapter.

Understanding supported business models in WebSphere Commerce

WebSphere Commerce provides the architectural infrastructure to put businesses that fit into one of the following business models online:

- Direct sales
- Business Hosting
- Business Value chain

Direct sales

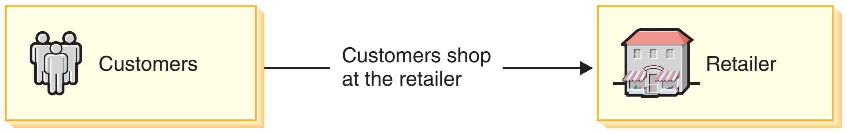
Direct sales supports commerce transactions involving products, services, or information directly between businesses and consumers or between two businesses or parties. WebSphere Commerce supports the following types of direct sales business models:

- Consumer direct
- Business B2B direct

Consumer direct

Consumer direct supports commerce transactions involving products, services, or information between businesses and consumers. Consumers typically purchase goods or services directly from a business in a consumer direct scenario.

The following diagram demonstrates a typical consumer direct business.



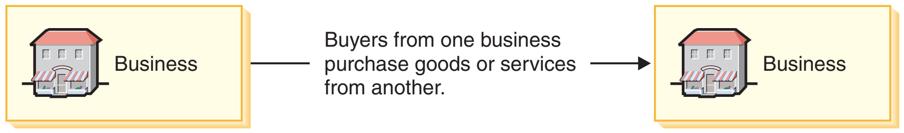
In a typical consumer direct business, customers buy directly from the business, usually a retailer, as shown in this diagram. The business can be a retailer, a manufacturer who sells their goods directly to consumers through their own retail outlet, or any other business that sells goods or provides services directly to consumers. For example, a business that sells to consumers directly through a catalog would be considered a consumer direct business.

Organizations that are not traditionally considered businesses, such as governments can also be considered consumer direct businesses. Governments may provide goods and services directly to customers.

B2B direct

Business B2B direct supports commerce transactions involving products, services, or information between two businesses or parties. Typical B2B direct transactions occur between buyers, suppliers, manufacturers, resellers, distributors, and trading partners.

The following diagram demonstrates a typical B2B direct business.


Business
Buyers from one business purchase goods or services from another.
Business

In a typical B2B direct business, businesses purchase goods or services directly from another business. The selling business can be a wholesaler, a distributor, a manufacturer, or a retailer who sells to buyers from other businesses.

Organizations that are not traditionally considered businesses, such as governments and the media can also be considered B2B direct businesses. Governments may provide goods and services directly to businesses.

Hosting

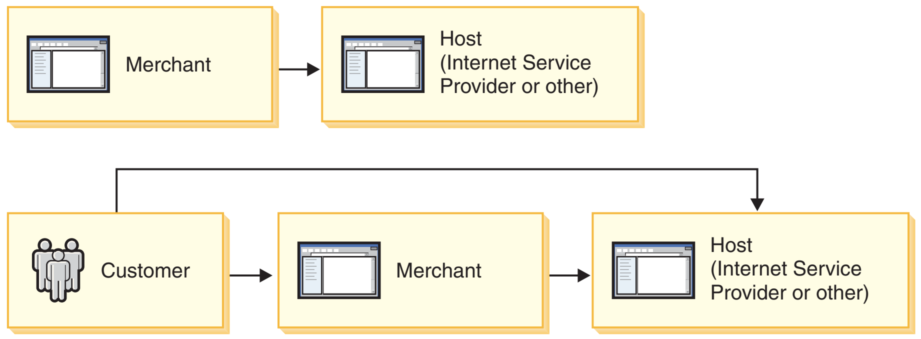
Business The hosting model supports hosting of merchants or other businesses by an Internet Service Provider (ISP) or other hosting provider.

There are two possible sides to the hosting business:
- hosted stores
- (optional) a site that allows customers to locate the stores that are hosted by the provider

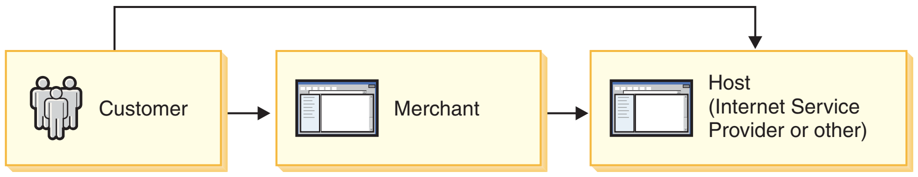
In order to manage relationships with the hosted stores, hosting models usually include a hub (known in WebSphere Commerce as a hub store). This hub provides self-provisioning tools that allow the merchant to create and administer a store, as well as tools that allow the hosting provider to manage all hosted stores.

Hosting providers also usually include a store in which customers can find and access the stores hosted by the provider.

The following diagram illustrates an example of hosting.


Merchant
Host (Internet Service Provider or other)
Customer
Merchant
Host (Internet Service Provider or other)

In this example, the merchant enters the host's site and creating a store that will be hosted by the site. Hosting providers often provide merchants with simple self-provisioning tools that allow the merchant to administer a hosted store. When a hosted store is open for business, customers can access the store via the host's site or by entering the hosted store directly.



In this example, the customer has the option of entering the hosted store or business directly or browsing the host's site and then being transferred to the hosted store or business.

Hosted stores are very similar to consumer direct stores. For specific differences between the two, as implemented in the WebSphere Commerce sample stores, see the *WebSphere Commerce Sample Store Guide*.

Value chain

Business New to WebSphere Commerce version 5.5 is the capability to enable online business transactions involving multiple enterprises. Value chains support transactions involving multiple enterprises or parties. Products, goods, services, or information are delivered through the parties of the value chain from producers to end users. A value chain also has relationship and administrative aspects, that is, you can manage the relationship of the partners or enterprises in your value chain, as well as offer some administrative services to those parties.

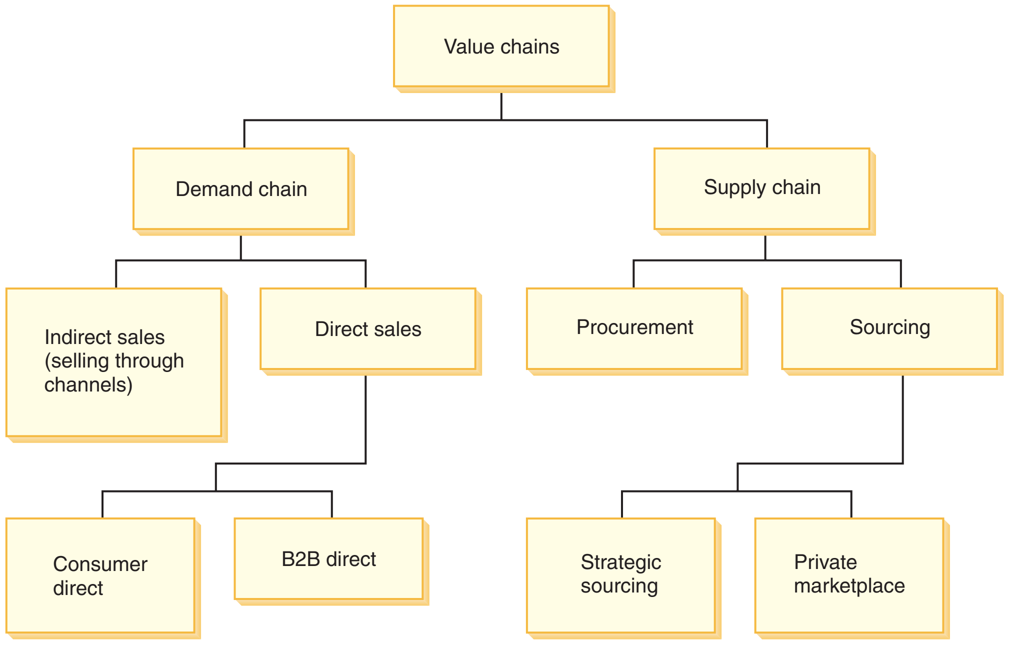
As a result, value chains must manage the two sides of their businesses: their customers and direct sales, and their channel partners and suppliers. Each of these sides requires its own management channels and practices.

In order to manage their relationships with partners or suppliers, value chain business models usually include a hub (in WebSphere Commerce known as a hub store). Value chain administrators can administer the operational aspects of the value chain in the hub store, including enabling partners or suppliers to participate in the value chain, that is, registering them, setting them up, creating collaborations. Partners and suppliers can also access the hub store to complete administrative tasks such as registering users.

In order to sell directly to customers (direct sales), value chains usually include a storefront, where customers can purchase their good or services directly.

WebSphere Commerce supports the transactions through, and relationship management of the following two types of value chains:
- Demand chain
- Supply chain

The following diagram provides an overview of the partners and relationships supported in value chains.



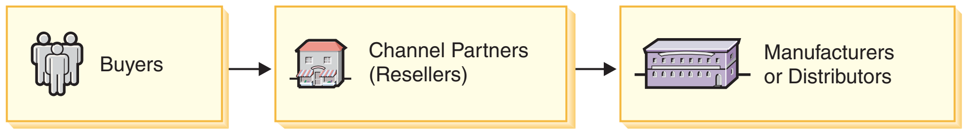
Demand chain

Business A demand chain is composed of the enterprises that sells a business's goods or services. For example, a demand chain may be composed of buyers who initiate the sales transaction, the resellers who sell the manufacturer's goods, and the manufacturer who creates the goods. Or a demand chain may be composed of the resellers who sell a manufacturer's goods, the manufacturer who makes the goods, and the distributors who supply the manufacturer's goods to the resellers. Demand chains also support direct sales channels, in which the demand chain owner sells directly to customers or partners itself. For more information on direct sales, see "Direct sales" on page 15.

Demand chain hosting: The demand chain owner may host stores for its channel partners, for example resellers or distributors.

The following diagrams illustrate examples of some of the demand chains supported by WebSphere Commerce.

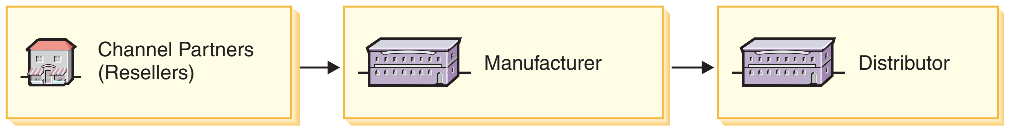
Buyers, channel partners (resellers), and manufacturers:



In this example, buyers purchase goods from a manufacturer's resellers (channel partners). Resellers, in turn, obtain the goods from the manufacturer, via the manufacturer's hub.

Note: The resellers may be hosted by the manufacturer or be remote.

Resellers, manufacturers, and distributors:



In this example the manufacturer provides a hub for their channel partners, including resellers. Resellers and other channel partners may be able to do several functions in this hub, including locating distributors of the manufacturer's goods.

In order to locate suppliers, the reseller may browse a product catalog in the private hub. If the desired products are available from more than one distributor, the reseller can check product availability, distributors' location, and prices for various distributors. Then, if the reseller chooses, they can split their order between several distributors. The order is then sent to the distributor, who completes the transaction and delivers the goods or services to the reseller. The reseller then sells the goods or services directly to the consumer.

The demand chain sample site, the Commerce Plaza, is an example of this reseller, manufacturer and distributor scenario.

Note: The resellers may be hosted by the manufacturer or be remote.

Other scenarios: The examples described in this section are just a few instances of demand chains. The scenario details may change depending on the type of business being conducted. For example, if the enterprise is a manufacturer, the purpose of the hub may be to help the manufacturer's resellers locate the manufacturer's goods from several distributors. If the enterprise is a distributor, the purpose of the hub may be to help the distributor's resellers find goods or services from several different suppliers.

Supply chain

Business

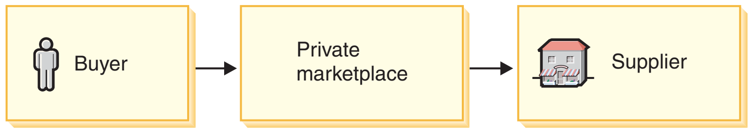
A supply chain is composed of the enterprises that provide services to a business. WebSphere Commerce provides the architectural infrastructure to support supply chains that take the form of a private marketplace.

A private marketplace provides a forum for vendors to offer their wares for sale. Buyers enters this forum and after browsing through the available options, select the appropriate goods or services.

Note: The private marketplace does not support competitive bidding and counter-bidding or other methods of competition.

Supply chain hosting: The supply chain owner may host a stores for its suppliers.

The following diagram illustrates an example of a supplier business.



In this example supply chain, the buyer enters the supplier's hub to interact and browses the an aggregated catalog in which products and offers from multiple suppliers are presented. The buyer can then select the desired offer or request quotes from multiple suppliers. The buy also has the option of conducting business or procuring from online suppliers directly.

Sample stores in WebSphere Commerce

WebSphere Commerce provides several sample stores that you can use to familiarize yourself with how WebSphere Commerce supports the different business models listed in this chapter. The samples available (and corresponding store archive files) are as follows:

Table 2.

Consumer direct	Business **B2B direct**	Business **Hosting**	Business **Demand chain**	Business **Supply chain**
FashionFlow (`ConsumerDirect.sar`) Express Express Store (`ExpressStore.sar`)	ToolTech (`B2BDirect.sar`)	**Note:** All the stores listed below are also available in the composite store archive `Hosting.sar`. It is recommended that you publish `Hosting.sar` to view the entire hosting sample.	**Note:** All the stores listed below are also available in the composite store archive `DemandChain.sar`. It is recommended that you publish `DemandChain.sar` to view the entire demand chain sample.	**Note:** All the stores listed below are also available in the composite store archive `SupplyChain.sar`. It is recommended that you publish `SupplyChain.sar` to view the entire supply chain sample.
		Commerce Hosting Hub (`Hosting Hub.sar`)	Commerce Plaza (`ChannelHub.sar`)	Commerce Supplier Hub (`SupplierHub.sar`)
		Store Directory (`Store Directory.sar`)	Catalog asset store (`CatalogAsset Store.sar`)	Catalog asset store (`Catalog AssetStore.sar`)
		Catalog asset store (`CatalogAsset Store.sar`)	Reseller storefront asset store (`Resellerstore frontAsset Store.sar`)	Supplier asset store (`Supplier AssetStore.sar`)
		Hosted storefront asset store (`HostedStore FrontAsset Store.sar`)	Distributor asset store (`DistributorAsset Store.sar`)	Suppliers
		Hosted stores	Hosted reseller stores	
			Distributor stores	

For more information on the types of stores in these samples, see "Understanding how the store architecture supports the business models" on page 66. For more detailed information on the sample stores, see the *WebSphere Commerce Sample Store Guide*.

Note: Each sample also contains a component store archive that contains the organization structure for the business model.

Note that these samples are representative of a specific instance of stores in each business model and are not meant to demonstrate all possible variations available in the business model. However, even if your specific instance of the business is quite different from the sample provided, you may be able to use the samples as a starting point for your own site, or use portions of it while creating your site. For more detailed information on the samples provided with WebSphere Commerce, see the *WebSphere Commerce Sample Store Guide*.

Part 3. WebSphere Commerce architecture

This section provides an overview of how the WebSphere Commerce architecture supports putting businesses online. In particular, this section discusses how components of the WebSphere Commerce architecture allow the different parties (for example, your customers, business partners, or distributors, resellers, and suppliers) in your business to interact online.

In order to enable the different parties (for example, your customers, business partners, vendors, suppliers, manufacturers, distributors, and administrators) that contribute to your business to interact with your business and each other online, WebSphere Commerce includes the following architectural components:

- Organization structure
- Access control model
- Business policy framework
- Instance architecture
- Store architecture

Together these components create the architecture that allows the different partners in your business to interact with each other.

Chapter 3. WebSphere Commerce organization structure

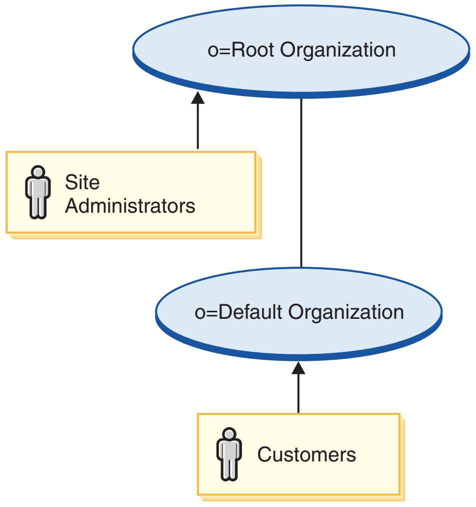
In order to allow customers or buyers to access your site, browse your catalog, and place orders; or to allow employees to administer the site, including updating the catalog, creating new promotions, or managing orders; or to allow resellers or other business partners to complete transactions on your site, all actors in your business scenario must be assigned a position in the WebSphere Commerce organization structure.

Understanding the WebSphere Commerce organization structure

The WebSphere Commerce organization structure provides a framework for the actors, or entities, in your business scenario. This framework is organized in a hierarchical structure, which mimics typical organizational hierarchies with entries for organizations and organizational units and users. The organizations and organizational units in the framework act as owners for the parts of your business. All parts of your business, including customers, administrators, stores, catalogs and distributors, must be owned by an organization or organizational unit.

The organization structure and the access control model, discussed in Chapter 4, “Access control in WebSphere Commerce,” on page 35, are closely related, in that the access control model applies access control policies to organizations rather than to individual entities (stores, customers, administrators and so on). The policies that apply to an entity (or resource) are applied to the organizations that own the entity or resource.

The following diagram outlines the basic WebSphere Commerce organization structure. The basic organization structure is installed during instance creation, regardless of the business model.



- **Root organization**: The root organization is the top level organization and is its own parent. All organizations in the WebSphere Commerce organization structure are descendents of the root organization. The site administrators are owned by the root organization.
- **Default organization:** The default organization is owned by the root organization. All guest customers and all customers in a consumer direct

scenario belong to the default organization. Customers in a B2B direct and value chain scenario can belong to either the default organization, or other organizations.

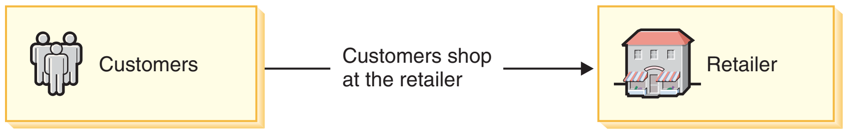
One or more other levels of organizational entities can exist beneath the parent organizational entities. You can add as many child organizational entities as necessary to support your business.

How does the organization structure support the business models?

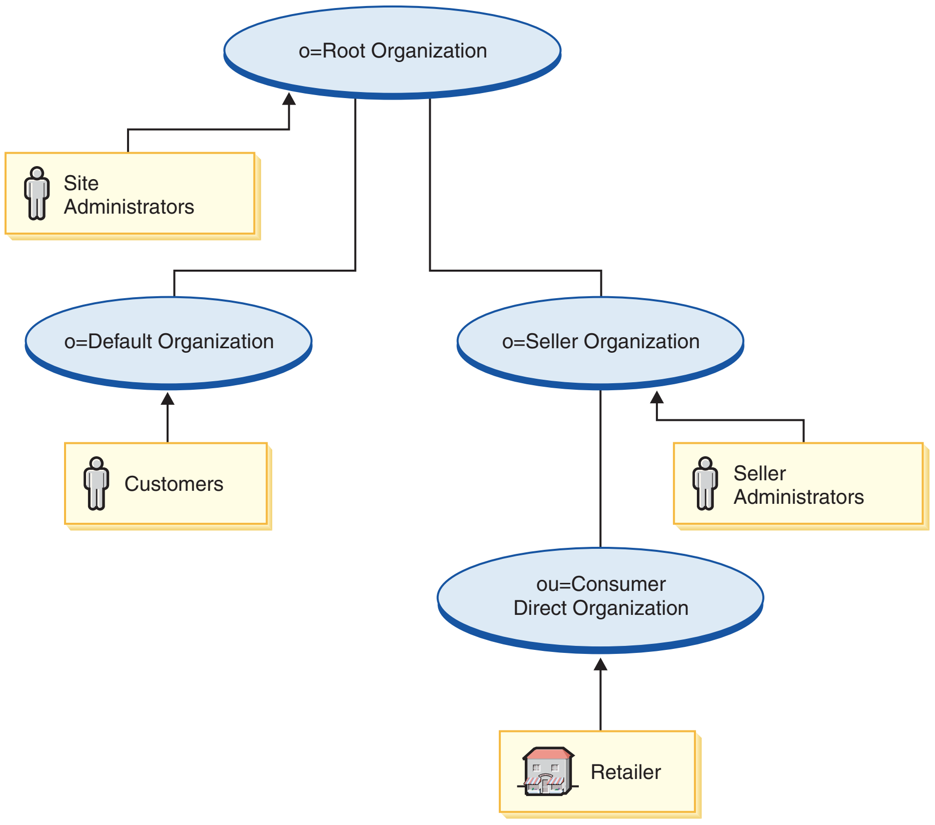
The WebSphere Commerce organization structure is flexible enough to support all entities in the supported business models. The diagrams in the following sections demonstrate how a typical example of each business model can be mapped to the WebSphere Commerce organization structure.

Consumer direct

The following diagram illustrates a typical consumer direct business.



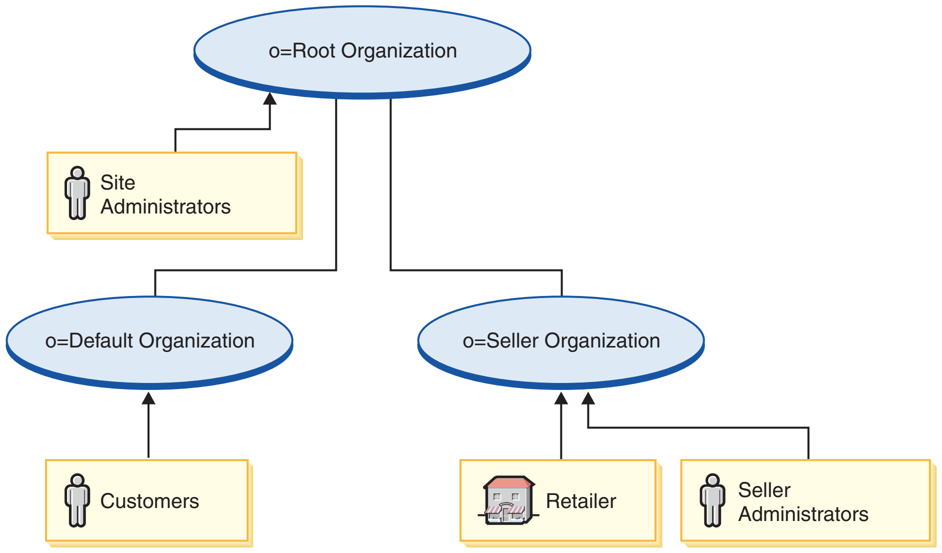
In order to place this business online with WebSphere Commerce, the entities in the preceding diagram must be assigned to the following organizations:



- **Root organization**: All organizations in the business become descendents of the root organization. The site administrators who maintain the online site, are owned by the root.

- **Default organization**: All of the business' customers are owned by the default organization.
- **Seller organization**: A seller organization is created to own all the seller organizations (including stores and the administrators who maintain the store). The administrators who maintain the store's functions (for example customer service representatives, catalog and product managers) are termed Seller administrators and are owned directly by the Seller organization.
  - A child organizational unit (ou), consumer direct organization, is created under the seller organization to own the store (Retailer).

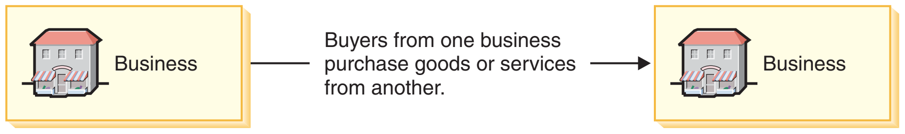
Express The organization structure in WebSphere Commerce - Express is slightly different than the consumer direct organization described above. In order to place a consumer direct business online with WebSphere Commerce - Express, the entities in the consumer direct diagram above must be assigned to the following organizations:



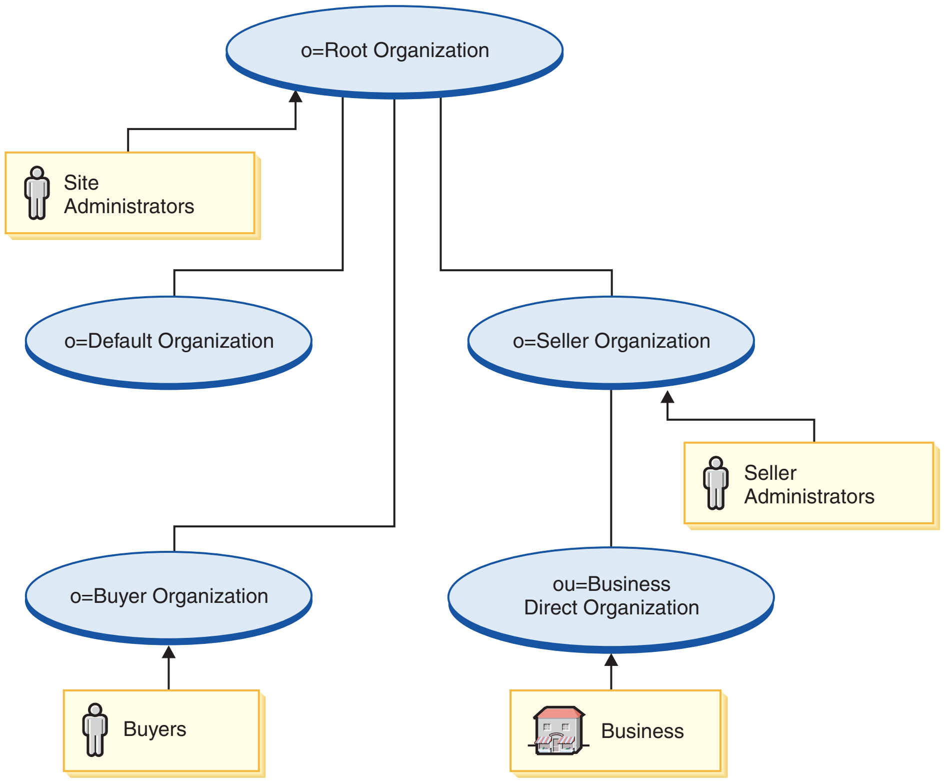
- **Root organization**: All organizations in the business become descendents of the root organization. The site administrators who maintain the online site, are owned by the root.
  - **Default organization**: All of the business' customers are owned by the default organization.
  - **Seller organization**: A seller organization is created to own all the stores (Retailer) and the administrators who maintain the store. The administrators who maintain the store's functions (for example customer service representatives, catalog and product managers) are termed Seller administrators and are owned directly by the Seller organization.

B2B direct

Business The following diagram illustrates a typical B2B direct business.


Business
Buyers from one business purchase goods or services from another.
Business

In order to place this business online, the entities in the preceding diagram must be assigned to the following organizations:

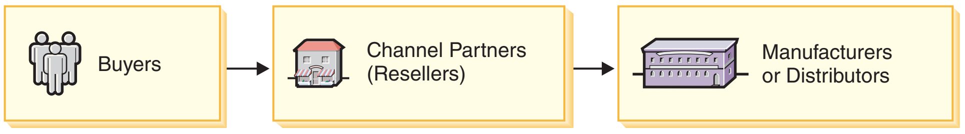


- **Root organization**: All organizations in the business become descendents of the root organization. The site administrators who maintain the online site, are owned by the root.
  - **Default organization**: Unlike the consumer direct organization structure, the customers are not owned by the default organization. Instead the customers are buyers who are owned by the buyer organization.
  - **Buyer organization**: Customers, known in B2B direct businesses as buyers, are assigned their own organization in the B2B direct organization structure.
  - **Seller organization**: A seller organization is created to own all the organizations that own stores. The administrators who maintain the store's functions (for example customer service representatives, catalog and product managers) are termed seller administrators and are owned directly by the seller organization.
    - A child organizational unit (ou), B2B direct organization, is created under the seller organization to own the store (Business).

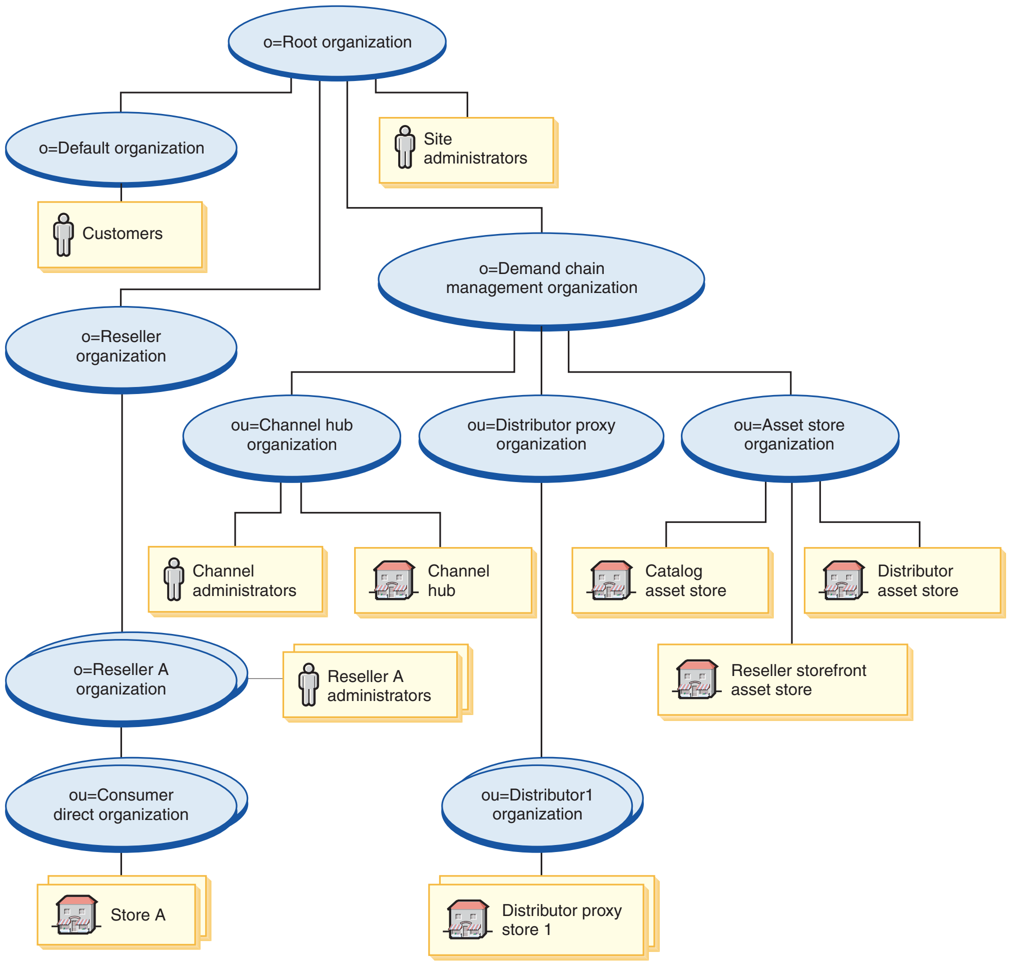
Demand chain

Business

The following diagram illustrates an example of a demand chain business.



In order to place this business online, the entities in the preceding diagram must be assigned to the following organizations:


o=Root organization
o=Default organization
Site administrators
Customers
o=Demand chain management organization
o=Reseller organization
ou=Channel hub organization
ou=Distributor proxy organization
ou=Asset store organization
Channel administrators
Channel hub
Catalog asset store
Distributor asset store
o=Reseller A organization
Reseller A administrators
Reseller storefront asset store
ou=Consumer direct organization
ou=Distributor1 organization
Store A
Distributor proxy store 1

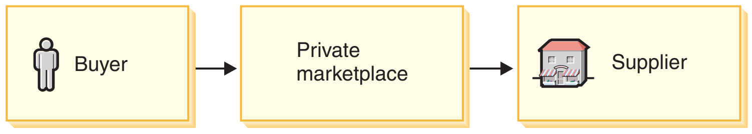
- **Root organization**: All organizations in the business become descendents of the root organization. As well, the administrators who will maintain the online site, the Site Administrators, are added directly under the root.
  - **Default organization**: By default, nothing is placed under the default organization. Customers of the reseller stores may be placed under this organization.
  - **Demand chain management organization**: The demand chain management organization is created to own all of the channel related organizations (with the exclusion of the organization that owns the resellers). The demand chain management organization owns the following child organizational units:
    - **Channel hub organization**: The channel hub organization is created to own the channel hub. The administrators who maintain the channel hubs functions, as well as administering the reseller organization, are termed channel administrators and are owned directly by the channel hub organization
    - **Distributor proxy organization**: The distributor proxy organization is created to own all connections to distributors. A child organizational unit is created for each distributor proxy in the organization.
      - **Distributor organization**: A new distributor organizational unit is created for each distributor proxy in the site.

  - **Asset store organization**: The asset store organization is created to own all assets that are used to create stores for channel partners (resellers and distributors).
- **Reseller organization**: The reseller organization is created to own all of the resellers in the demand chain. A child organization is created for each reseller.
  - **Reseller organization A, B, C**: A new reseller organization is created under the parent reseller organization, for each reseller store. The administrators who maintain the store's functions (for example customer service representatives, catalog and product managers) are termed reseller administrators and are owned directly by the corresponding reseller organization.

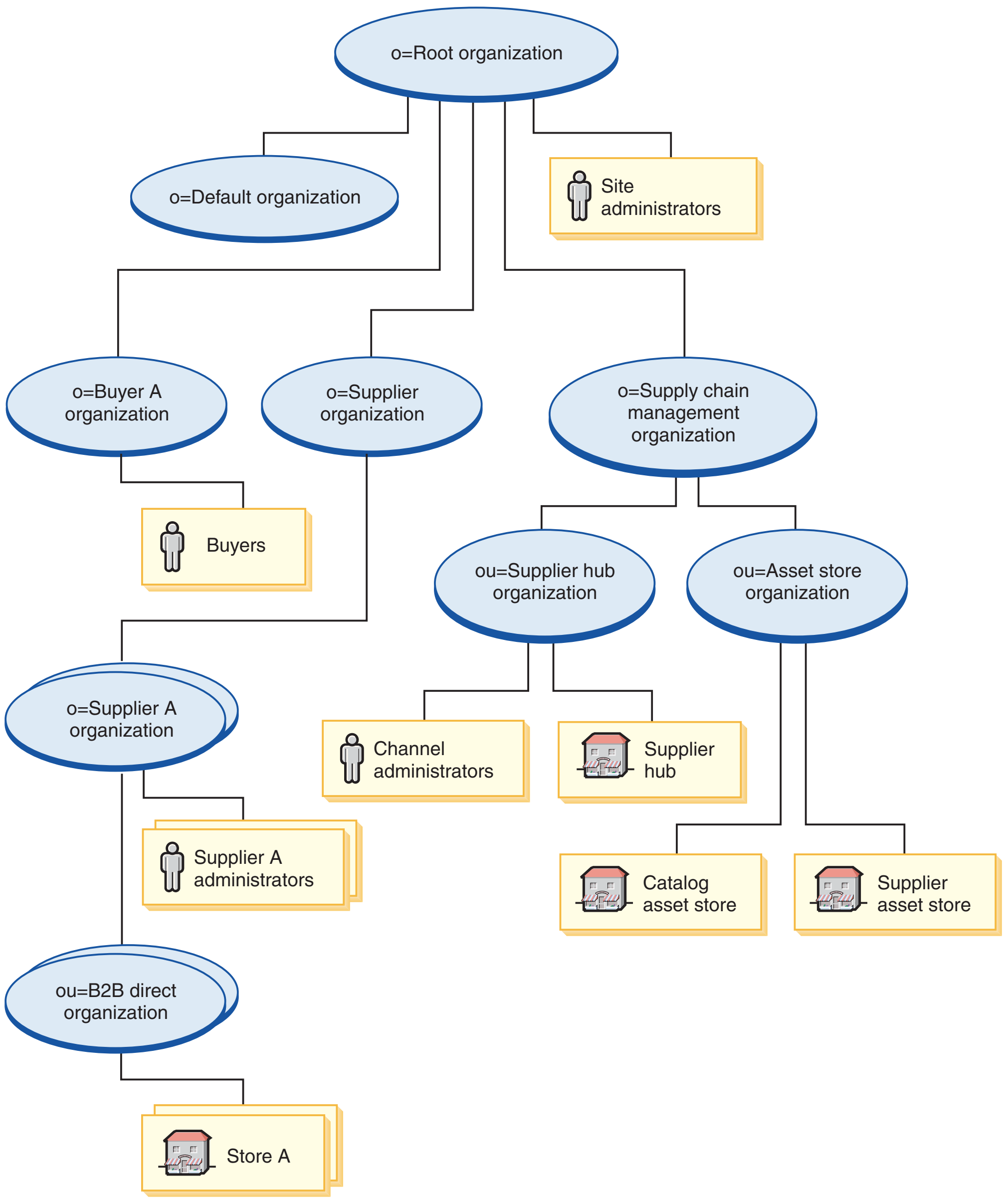
Supply chain

Business

The following diagram illustrates a typical supply chain business.



In order to place this business online, the entities in the preceding diagram must be assigned to the following organizations:



- **Root organization**: All organizations in the business become descendents of the root organization. As well, the administrators who will maintain the online site, the Site Administrators, are added directly under the root.
  - **Default organization**: By default, nothing is placed under the default organization.
  - **Supply chain management organization**: The supply chain management organization is created to own all of the supply chain related organizations (with the exclusion of the organization that owns the suppliers). The supply chain management organization owns the following child organizational units:
    - **Supplier hub organization**: The supplier hub organization is created to own the supplier hub. The administrators who maintain the supplier hubs

functions, as well as administering the supplier organization, are termed channel administrators and are owned directly by the supplier hub organization

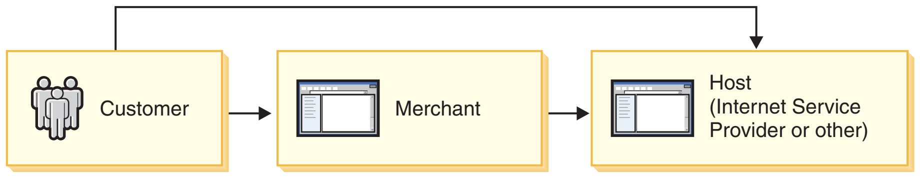
    - **Asset store organization**: The asset store organization is created to own all assets that are used to create stores for suppliers.

- **Supplier organization**: The supplier organization is created to own all of the suppliers in the supply chain. A child organization is created for each supplier.
    - **Supplier organization A, B, C**: A new supplier organization is created under the parent supplier organization, for each supplier store. The administrators who maintain the store's functions are termed supplier administrators and are owned directly by the corresponding supplier organization.
- **Buyer organization**: Buyers are given their own organization under the root. All buyers are owned by the corresponding buyer organization.

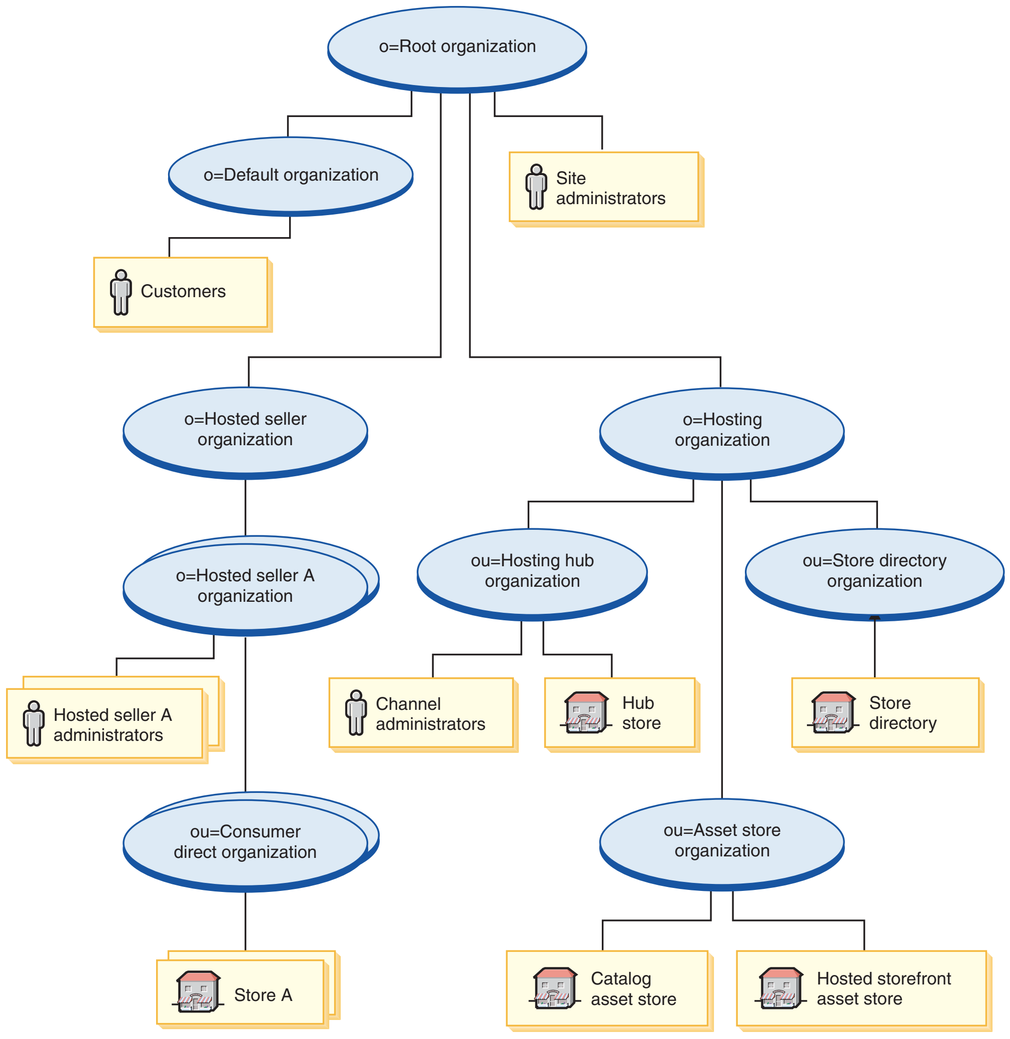
Hosting

Business

The following diagram illustrates a typical hosting business.



In order to place this business online, the entities in the preceding diagram must be assigned to the following organizations:


o=Root organization
o=Default organization
Site administrators
Customers
o=Hosted seller organization
o=Hosting organization
o=Hosted seller A organization
ou=Hosting hub organization
ou=Store directory organization
Hosted seller A administrators
Channel administrators
Hub store
Store directory
ou=Consumer direct organization
ou=Asset store organization
Store A
Catalog asset store
Hosted storefront asset store

- **Root organization**: All organizations in the business become descendents of the root organization. As well, the administrators who will maintain the online site, the Site Administrators, are added directly under the root.
  - **Default organization**: All of the business' customers are owned by the Default organization
  - **Hosting organization**: The hosting organization is created to own all of the hosting related organizations (with the exclusion of the organization that owns the hosted stores). The hosting organization owns the following child organizational units:
    - **Hosting hub organization**: The hosting hub organization is created to own the hosting hub. The administrators who maintain the hosting hub's functions, as well as administering the hosting organization, are termed channel administrators and are owned directly by the hosting hub organization.
    - **Store directory organization**: The store directory organization is created to own the store directory.
    - **Asset store organization**: The asset store organization is created to own all assets that are used to create hosted stores.

– **Hosted seller organization**: The hosted seller organization is created to own all of the hosted stores. A child organization unit is created for each hosted store.
  - **Hosted store organization A, B, C**: A new hosted store organization is created under the parent hosting organization, for each hosted store. The administrators who maintain the store's functions are termed hosted seller administrators and are owned directly by the corresponding hosted store organization.

Sample organization structures

WebSphere Commerce provides sample organizations structures for each supported business model. These sample organization structures are available on their own (as component store archives) allowing you to use the sample organization structure as as starting point for your own site, or as part of the sample businesses. For more information on the sample organization structures, see the *WebSphere Commerce Sample Store Guide*.

Creating organization structures

Rather than create new organization structures for your site, it is recommended that you begin by publishing one of the sample organization structures provided with WebSphere Commerce, and then make changes to that organization structure as necessary. For more information on editing organization data, see "Understanding member assets in WebSphere Commerce" on page 115.

Chapter 4. Access control in WebSphere Commerce

WebSphere Commerce allows you to determine, through access control, which tasks a particular user, be they customers, buyers, administrators, distributors, manufacturers, or suppliers, can perform in relation to your business.

The access control model for WebSphere Commerce is covered in detail in the *WebSphere Commerce Security Guide*. However, in order to understand how access control affects site and store development, a brief summary is provided here.

Understanding access control in WebSphere Commerce

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. One common attribute that is used to determine membership of an access group is roles. Roles are assigned to users on a per organization basis. For more information about roles, see “Roles” on page 117. Some examples of access groups include 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 declared as actions and assigned to an action group before they can be accessed.
- Resources are the entities that are protected. For example, if the action is a view, the resource to be protected is the command that invoked the view, for example 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. For example, users may only be allowed to display the orders that they have created.

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.

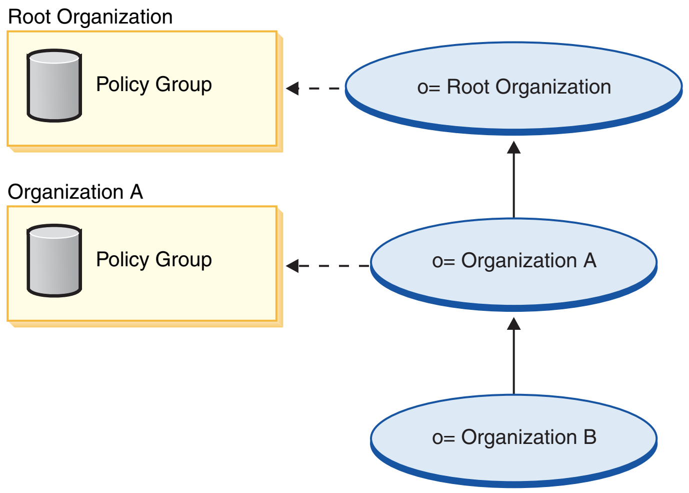
WebSphere Commerce provides over three hundred default access control policies that are loaded during instance creation. These policies cover a wide range of common business activities, including order creation and processing, and trading, such as request for ▶Business quotes and ▶Business contracts. The default policies are documented in the *WebSphere Commerce Security Guide*.

Access control policy groups

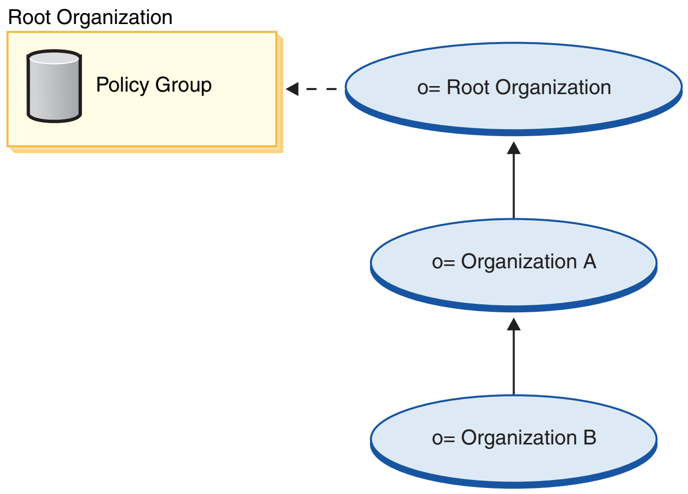
In order for an access control policy to be applied to your store or site, it must belong to an access control policy group and the policy group must be subscribed by the organization that owns the resource. By default, all access control policies provided with WebSphere Commerce are assigned to policy groups. For a list of default policies provided with WebSphere Commerce, see the *WebSphere Commerce Security Guide*.

Although access control policy groups are owned by organizations, they are not automatically applied to the organization. An organization must subscribe to a policy group in order for the access control policies to apply to the organization. If the organization has child organizations, all policy groups the parent subscribes to are automatically applied to the child organizations. However, if the child organization subscribes directly to a policy group, the policy groups subscribed to by the parent organization no longer apply to the child.

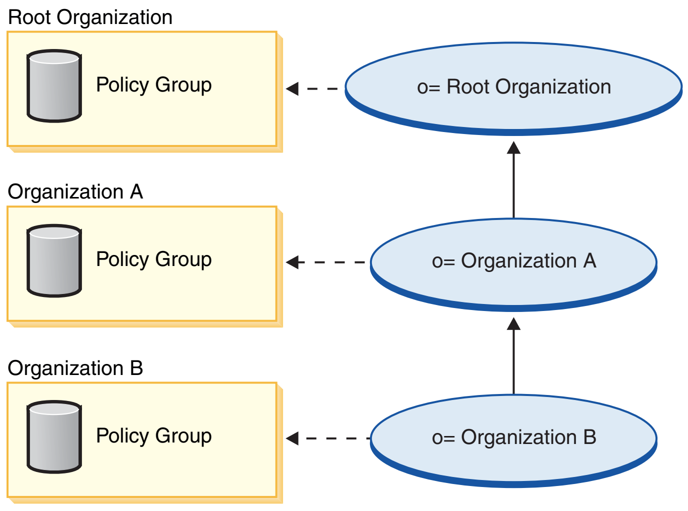
In previous versions of WebSphere Commerce, a policy applied to all resources owned by the descendants of that policy's owner organization. For example, if Organization A had a certain policy and was the parent of Organization B, then Organization B implicitly had that policy as well. In WebSphere Commerce 5.5, organizations can now subscribe to policy groups. In WebSphere Commerce 5.5 , if Organization B does not subscribe to any policy groups, the access control framework will begin searching up the organization hierarchy until it encounters an organization that subscribes to at least one policy group. If Organization B's immediate parent organization, Organization A, subscribes to a policy group, the searching stops, and the policies in Organization A's policy group are applied to Organization A and B. This can be seen in the following diagram.



If Organization A does not subscribe to a policy group, the search continues up the organization hierarchy, until an organization with a subscription is reached. This is seen in the following diagram where the Root Organization subscribes to a policy group. Organization B and Organization A inherit the policies in that group.



If Organization B subscribes to a policy group, the search stops at Organization B and Organization B can only apply to those policies to which it has subscribed, as shown in the following diagram.



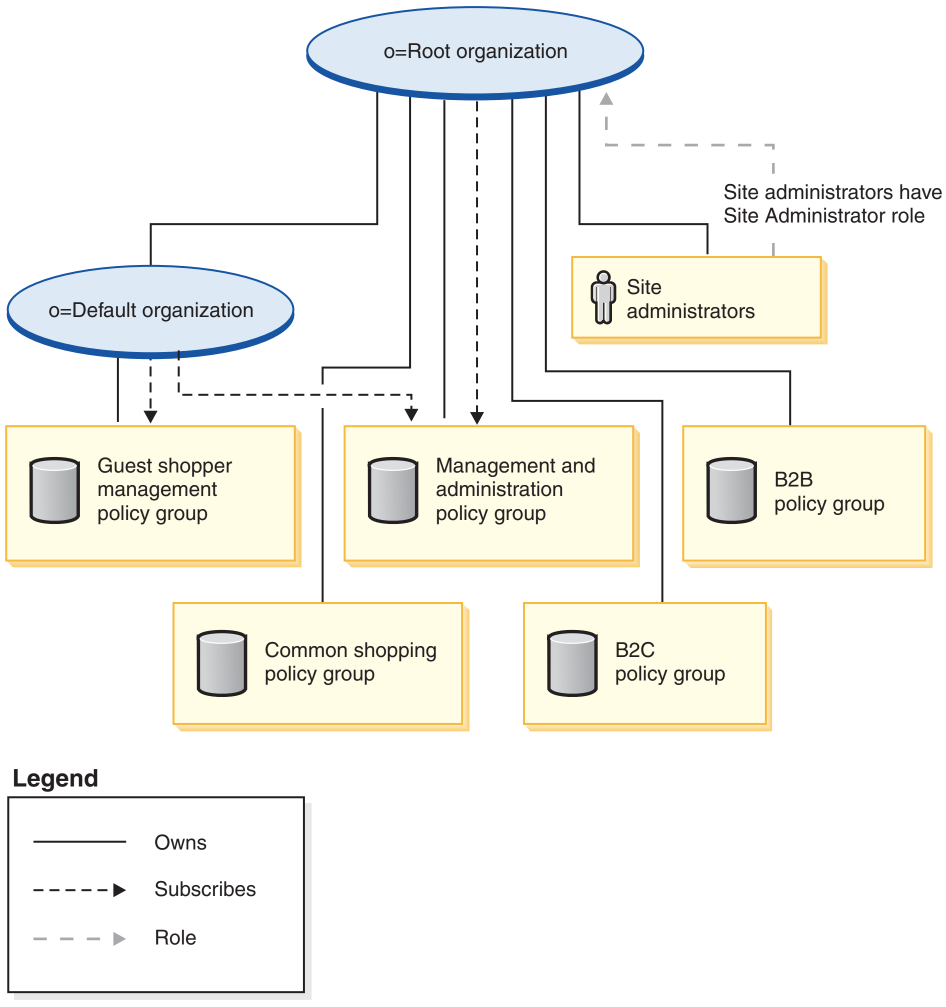
Note: In terms of access control, ownership of resources has a special meaning. All resources must implement the com.ibm.commerce.security.Protectable interface. One of the methods on this interface is getOwner(), which returns the member ID of the owner of the resource. For example, the Order entity 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 *WebSphere Commerce Programming Guide and Tutorials*.

Understanding access control in the business models

The WebSphere Commerce access control structure is flexible enough to support all entities in the supported business models. The diagrams in the following sections demonstrate how access control is applied to a typical example of each business model.

Basic access control structure

The basic access control structure is installed during instance creation, regardless of the business model.



The root organization owns the following default policy groups:

- Management and administration
- Common shopping
- B2C
- B2B

However, the root organization only subscribes to the management and administration policy group. As a result, these policies apply to the site administrators, who are directly under the root.

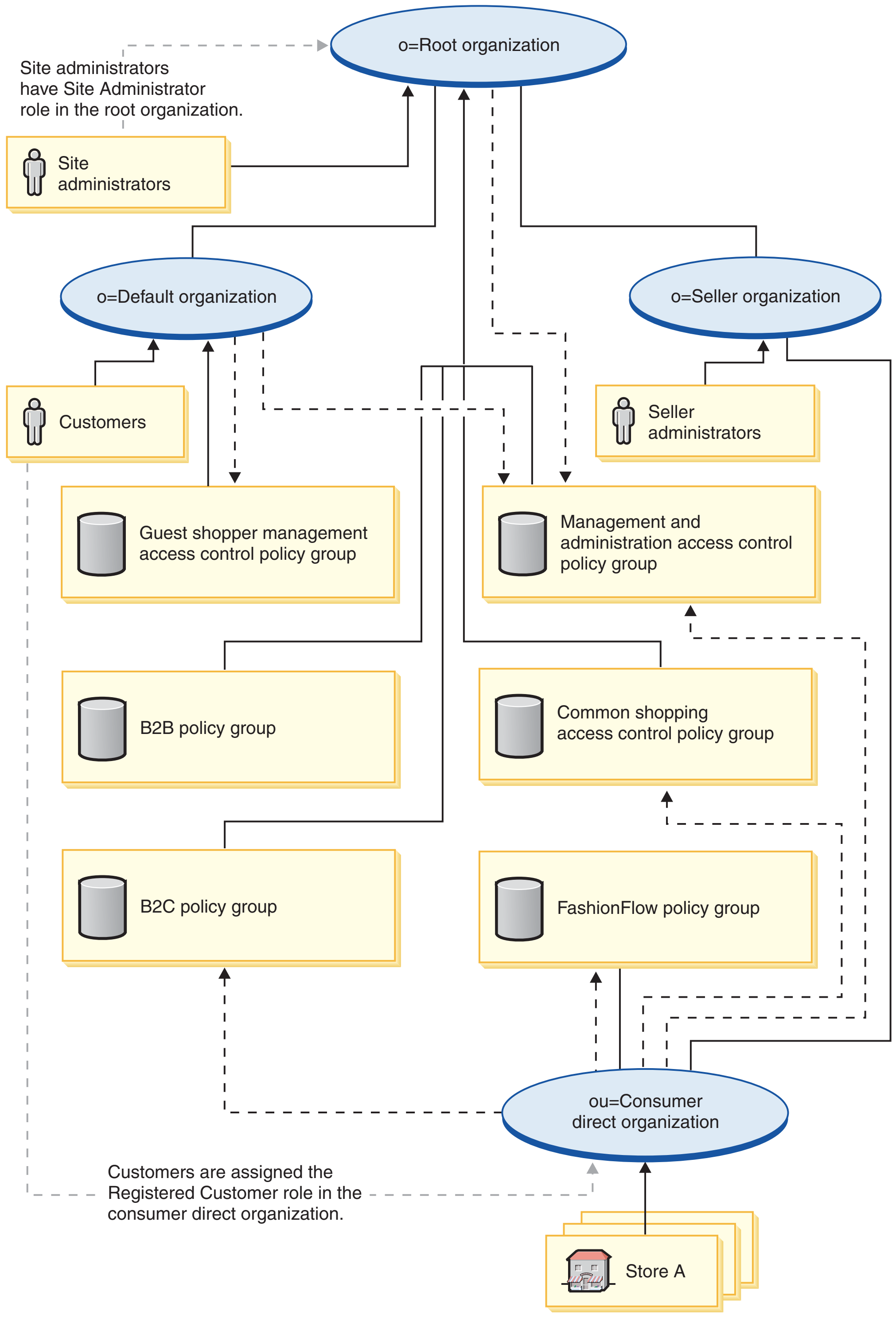
The policies in the management and administration policy group do not apply to the default organization through inheritance, as the default organization subscribes to the guest shopper management policy group. In order for the management and administration policies to apply, the default organization must subscribe to the management and administration policy group explicitly.

The default organization owns the guest shopper management policy group.

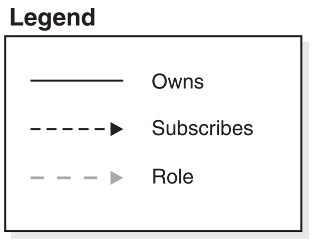
Note: For more detailed information on the default policy groups, see the appendix of the *WebSphere Commerce Security Guide*.

Consumer direct

The following diagram describes a basic consumer direct organization and access control structure.



o=Root organization
Site administrators have Site Administrator role in the root organization.
Site administrators
o=Default organization
o=Seller organization
Customers
Seller administrators
Guest shopper management access control policy group
Management and administration access control policy group
B2B policy group
Common shopping access control policy group
B2C policy group
FashionFlow policy group
ou=Consumer direct organization
Customers are assigned the Registered Customer role in the consumer direct organization.
Store A



In this diagram describing the basic consumer direct organization, the root organization owns and subscribes to the default policy groups as described in "Basic access control structure" on page 38.

The consumer direct organization subscribes directly to the B2C access control policies, the management and administration policy group, and the common shopping policy group.

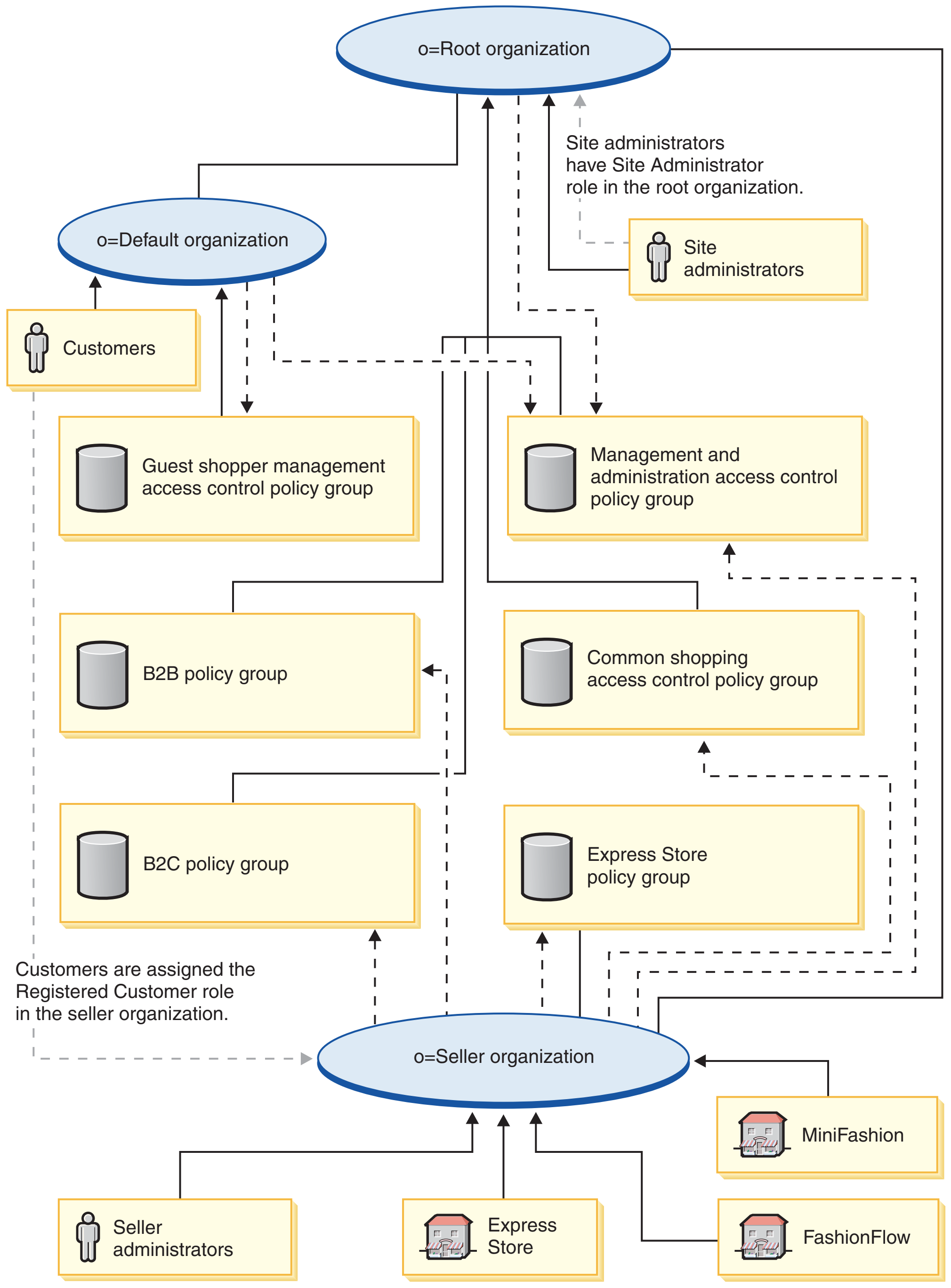
The consumer direct organization also owns and subscribes to the FashionFlow policy group. The FashionFlow policy group contains the following policy:

AllUsersExecuteFashionAllUsersViews

Since access control policy groups are subscribed by organizational entities, if you are creating multiple stores in your site and want to apply different access control policy groups to individual stores, you must create separate organizations to own each store.

▶ Express The access control structure in WebSphere Commerce - Express is slightly different than the consumer direct access control structure described above.

The following diagram describes the access control structure in WebSphere Commerce - Express:



o=Root organization
Site administrators have Site Administrator role in the root organization.
Site administrators
o=Default organization
Customers
Guest shopper management access control policy group
Management and administration access control policy group
B2B policy group
Common shopping access control policy group
B2C policy group
Express Store policy group
Customers are assigned the Registered Customer role in the seller organization.
o=Seller organization
MiniFashion
Seller administrators
Express Store
FashionFlow

In this diagram describing the consumer direct organization in WebSphere Commerce - Express, the root organization owns and subscribes to the default policy groups as described in "Basic access control structure" on page 38.

The Seller organization subscribes directly to the B2C access control policies, B2B access control policies, the management and administration policy group, and the common shopping policy group.

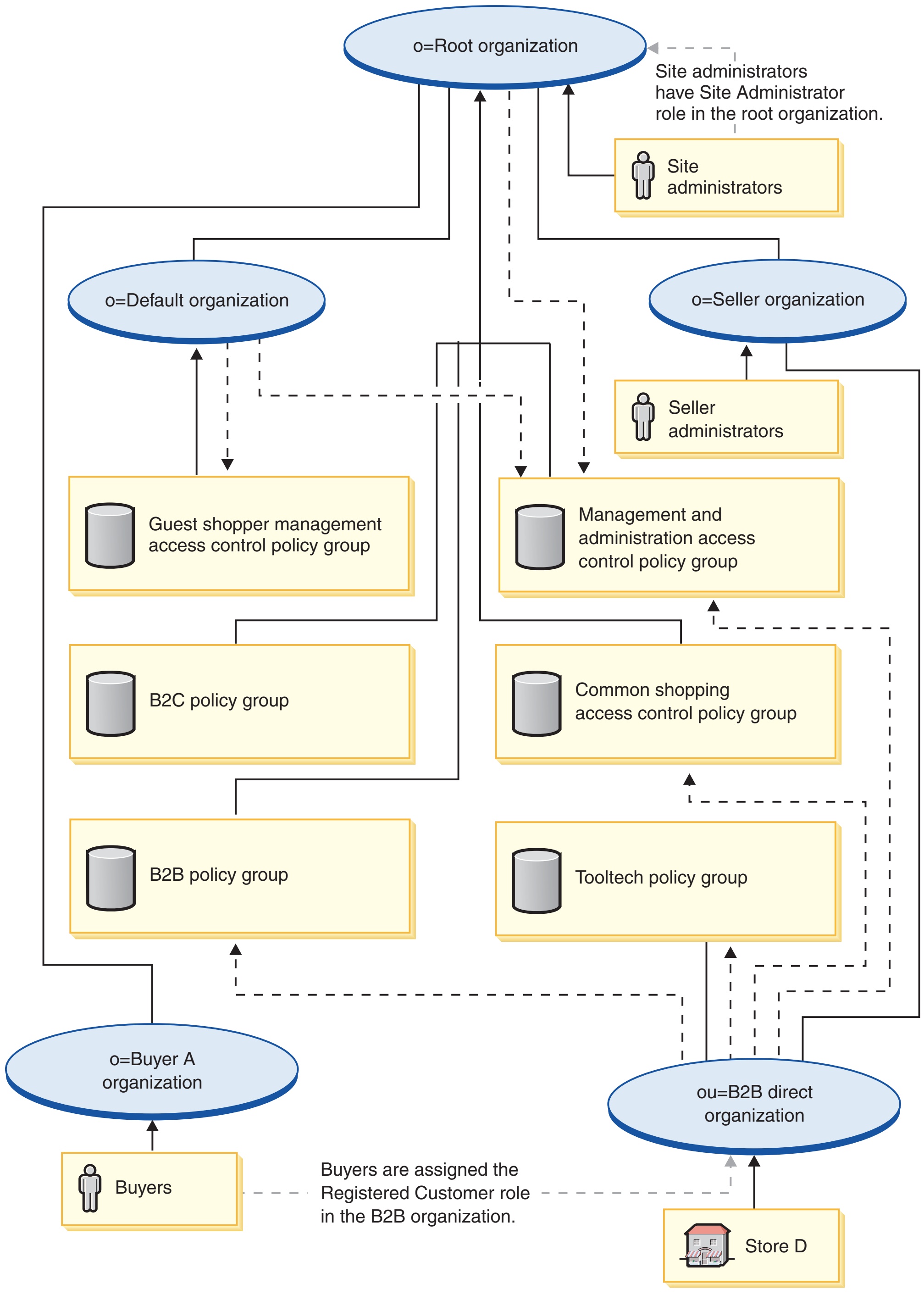
The seller organization also owns and subscribes to the Express policy group. The Express policy group contains the following policies:

- AllUsersExecuteExpressAllUsersViews
- RegisteredUsersExecuteExpressAllUsersViews

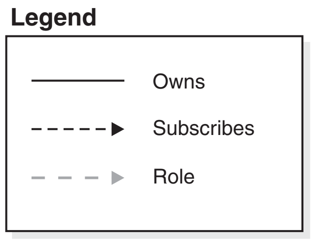
B2B direct

Business

The following diagram describes a basic B2B direct organization and access control structure.



o=Root organization
Site administrators have Site Administrator role in the root organization.
Site administrators
o=Default organization
o=Seller organization
Seller administrators
Guest shopper management access control policy group
Management and administration access control policy group
B2C policy group
Common shopping access control policy group
B2B policy group
Tooltech policy group
o=Buyer A organization
ou=B2B direct organization
Buyers
Buyers are assigned the Registered Customer role in the B2B organization.
Store D



In this diagram, describing a basic B2B direct organization structure, the root organization owns and subscribes to the default policy groups as described in "Basic access control structure" on page 38.

The B2B direct organization subscribes directly to the B2B, management and administration, and the common shopping policy groups.

The B2B direct organization also owns and subscribes to the ToolTech policy group. The ToolTech policy group contains the following policies:
- AllUsersForToolTechExecuteToolTechAllUsersViews
- RegisteredCustomersForOrgForToolTechExecuteToolTech RegisteredCustomerViews

Buyers are customers that place orders in a B2Bdirect store. All buyers must be owned by a buyer organization. Typically, buyer organizations do not subscribe to any policy groups, since management and administration policies inherited from the root organization are sufficient.

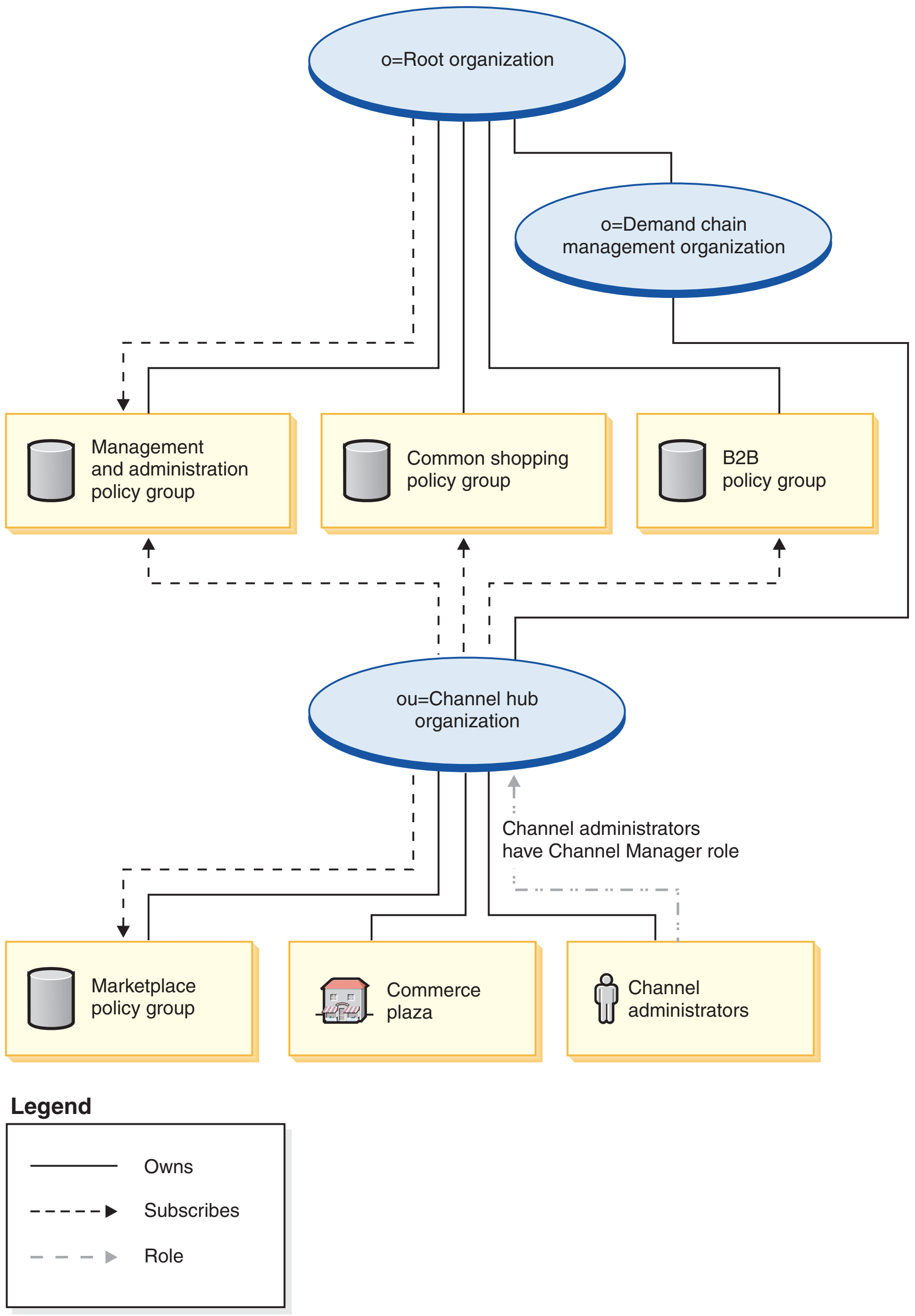
Since access control policy groups are subscribed by organizational entities, if you are creating multiple stores in your site, and want to apply different access control policy groups to individual stores, you must create separate organizations to own each store.

Demand chain

Business

In these diagrams, describing a demand chain organization structure, the root organization owns and subscribes to the default policy groups as described in

"Basic access control structure" on page 38.

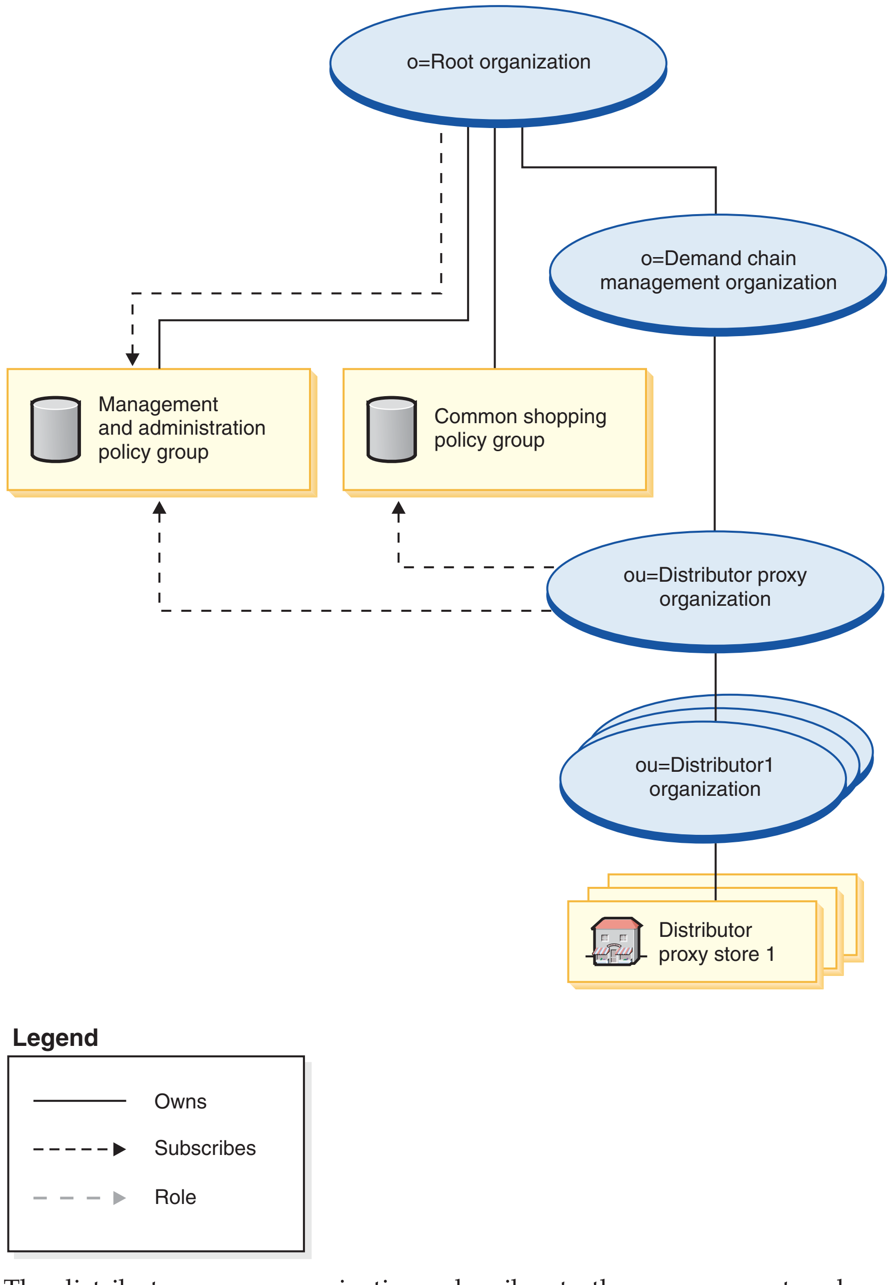

o=Root organization
o=Demand chain management organization
Management and administration policy group
Common shopping policy group
B2B policy group
ou=Channel hub organization
Channel administrators have Channel Manager role
Marketplace policy group
Commerce plaza
Channel administrators
Legend
Owns
Subscribes
Role

The channel hub organization subscribes directly to the Management and administration policy group, the common shopping policy group, the B2B policy group and owns and subscribes the Marketplace policy group. As a result, these policies apply to the channel administrators, who are directly under the channel hub organization, as well as to the channel hub (Commerce Plaza).

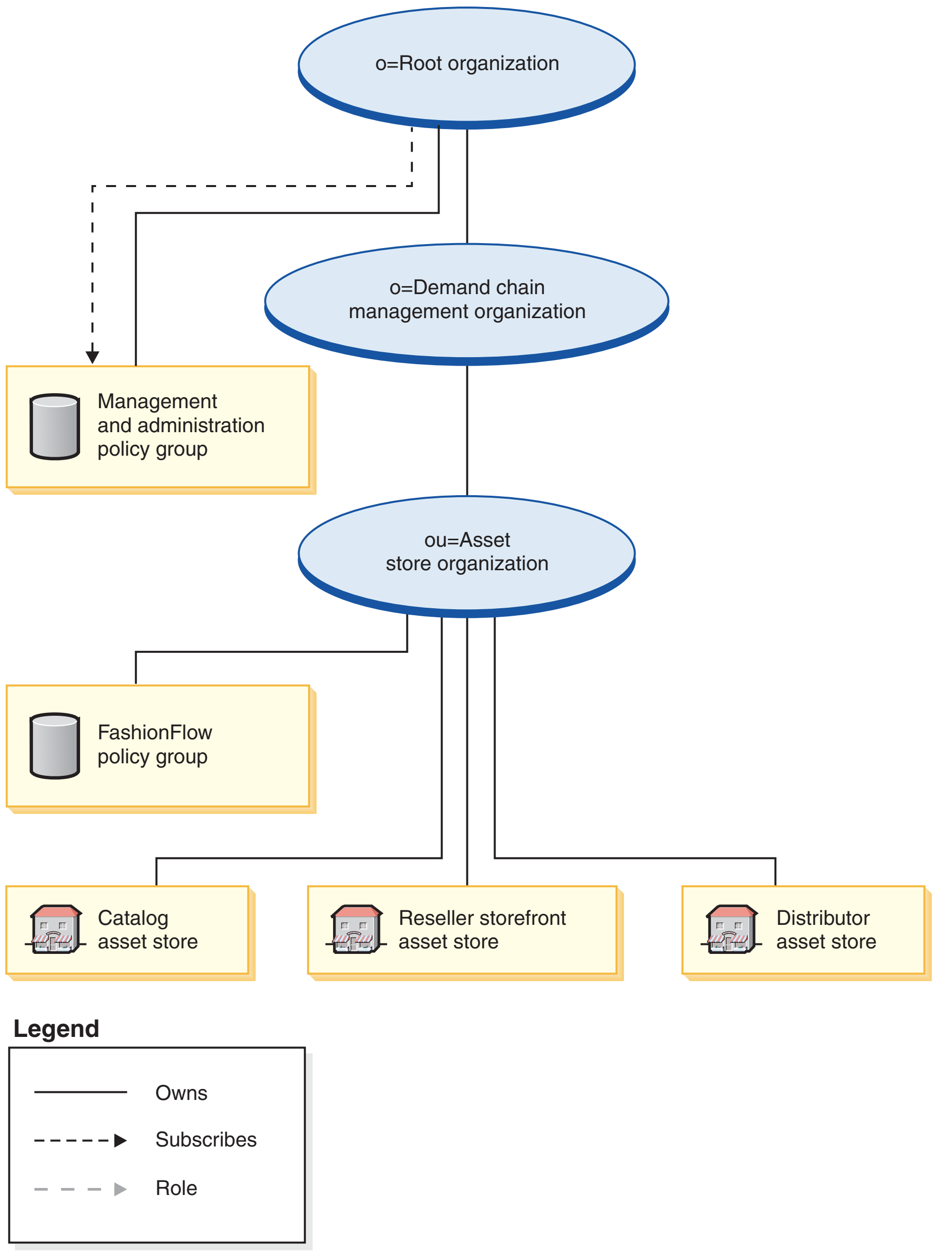
The Marketplace policy group contains the following policies:

- AllUsersExecuteMarketplaceAllUserViews
- RegisteredCustomersForOrgExecuteMarketplaceRegistered CustomerViews

- ContractAdministratorsForChannelOrgExecuteCreate CommandsOnMemberResource
- ContractAdministratorsForChannelOrgExecuteContract DeployCommandsOnContractResource
- ContractAdministratorsForChannelOrgDisplayContract DatabeanResourceGroup

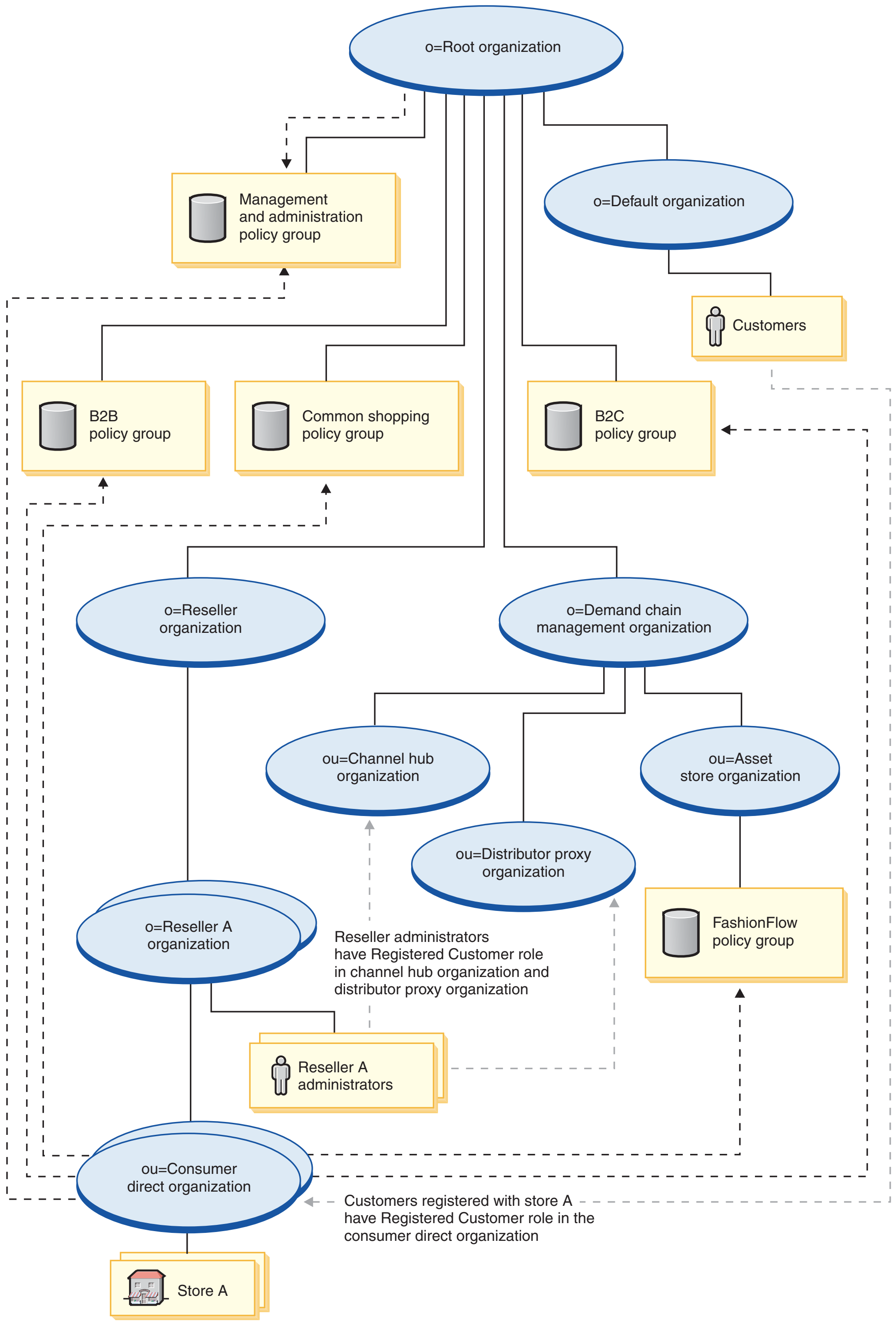

o=Root organization
o=Demand chain management organization
Management and administration policy group
Common shopping policy group
ou=Distributor proxy organization
ou=Distributor1 organization
Distributor proxy store 1
Legend
Owns
Subscribes
Role

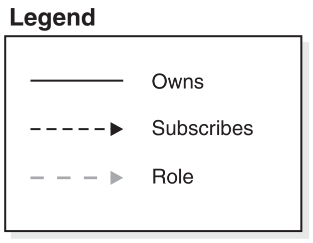
The distributor proxy organization subscribes to the management and administration policy group and the common shopping policy group. As a result, these policies apply to the distributor organizations who are directly under the distributor proxy organization.


o=Root organization
o=Demand chain management organization
Management and administration policy group
ou=Asset store organization
FashionFlow policy group
Catalog asset store
Reseller storefront asset store
Distributor asset store
Legend
Owns
Subscribes
Role

The asset store organization does not subscribe directly to any policy groups. As a result it inherits the management and administration policy group from the root organization. These policies apply to the asset store organization and the asset stores that it owns. The asset store organization owns the FashionFlow policy group, but does not subscribe to it.

Note: The individual reseller consumer direct organizations will subscribe to the FashionFlow policy group when the reseller store is created.


o=Root organization
Management and administration policy group
o=Default organization
Customers
B2B policy group
Common shopping policy group
B2C policy group
o=Reseller organization
o=Demand chain management organization
ou=Channel hub organization
ou=Asset store organization
ou=Distributor proxy organization
o=Reseller A organization
FashionFlow policy group
Reseller administrators have Registered Customer role in channel hub organization and distributor proxy organization
Reseller A administrators
ou=Consumer direct organization
Customers registered with store A have Registered Customer role in the consumer direct organization
Store A



The reseller organization does not subscribe directly to any policy groups. As a result it inherits the management and administration policy group from the root organization. These policies apply to the reseller organization and the reseller A organizations that it owns as well as to the reseller A administrators.

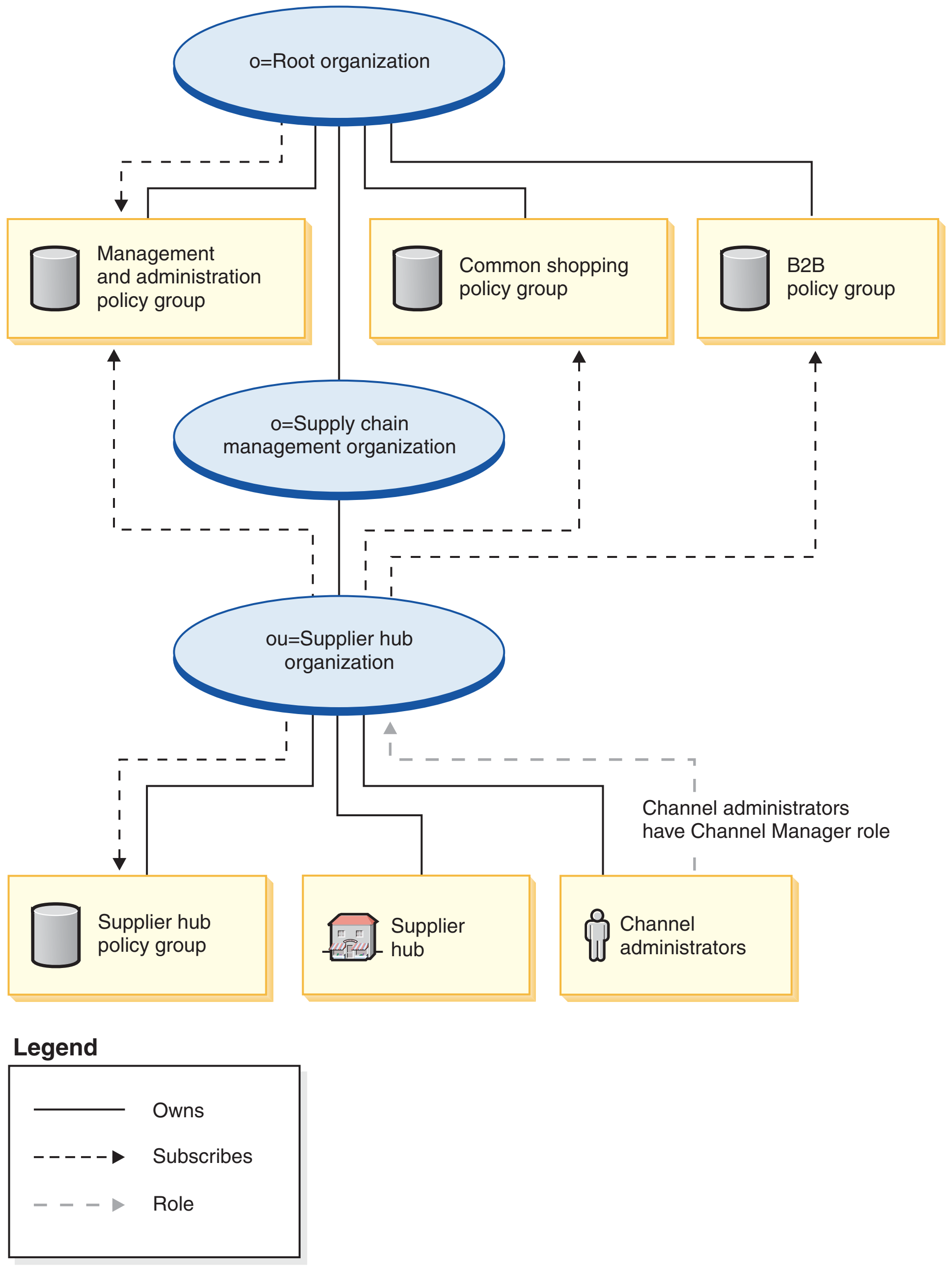
The consumer direct organization subscribes directly to the management and administration policy group, common shopping policy group, B2C and B2B policy groups, as well as to the FashionFlow policy group. These policies apply to all stores owned by the consumer direct organization.

Supply chain

Business

In these diagrams, describing a basic supply chain organization structure, the root organization owns and subscribes to the default policy groups as described in

"Basic access control structure" on page 38.

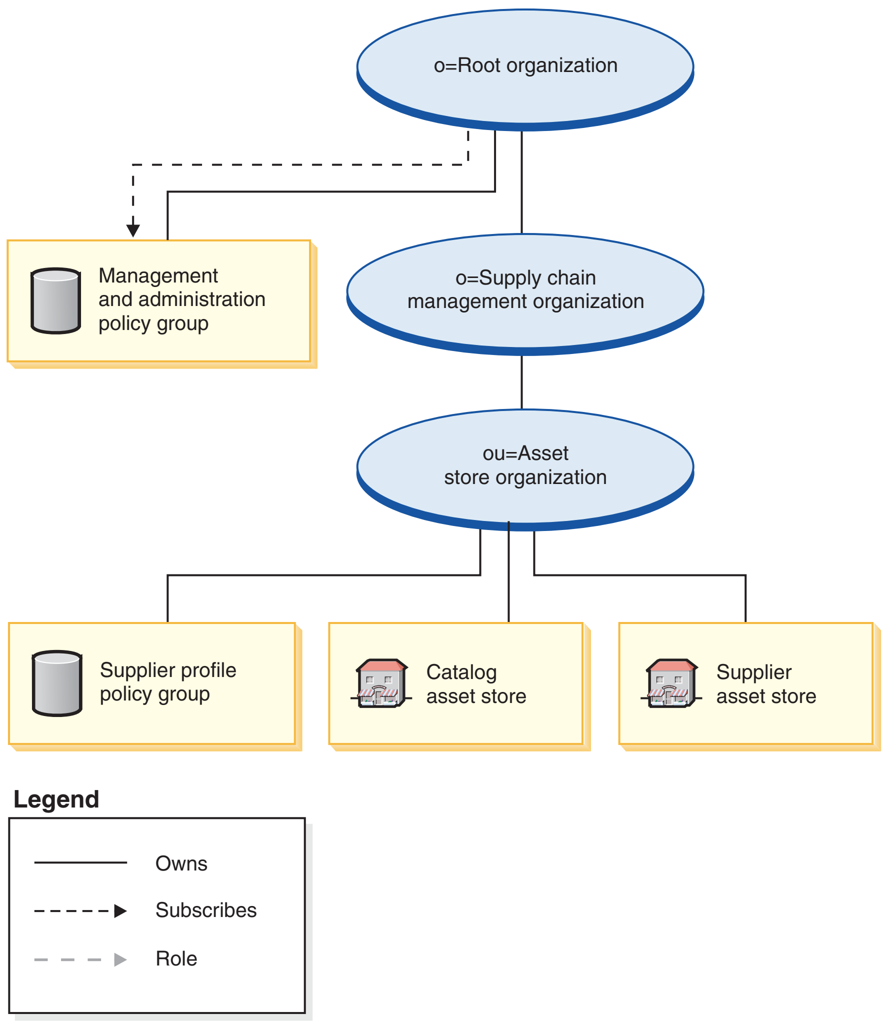

o=Root organization
Management and administration policy group
Common shopping policy group
B2B policy group
o=Supply chain management organization
ou=Supplier hub organization
Channel administrators have Channel Manager role
Supplier hub policy group
Supplier hub
Channel administrators
Legend
Owns
Subscribes
Role

The supplier hub organization subscribes directly to the management and administration policy group, the common shopping policy group, the B2B policy group and owns and subscribes to the Supplier hub policy group. As a result, these policies apply to the channel administrators, who are directly under the supplier hub organization, as well as to the supplier hub.

The Supplier hub policy group contains the following policies:

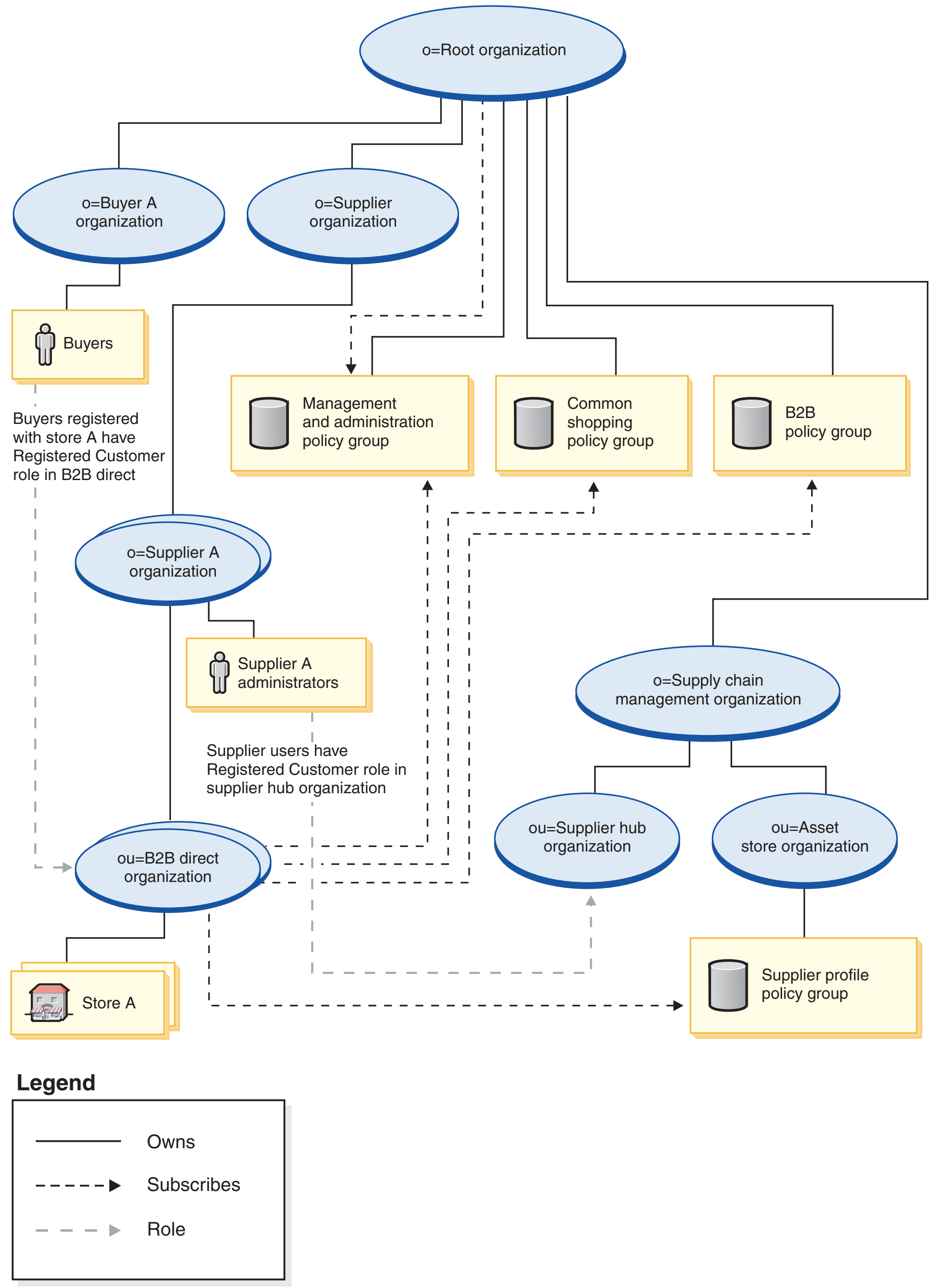
- AllUsersForSupplierHubExecuteSupplierHubAllUsersViews
- RegisteredCustomersForOrgForSupplierHubExecuteSupplierHub RegisteredCustomerViews
- ContractAdministratorsForChannelOrgExecuteCreateCommands OnMemberResource

- ContractAdministratorsForChannelOrgExecuteContractDeploy CommandsOnContractResource
- ContractAdministratorsForChannelOrgDisplayContract DatabeanResourceGroup


o=Root organization
Management and administration policy group
o=Supply chain management organization
ou=Asset store organization
Supplier profile policy group
Catalog asset store
Supplier asset store
Legend
Owns
Subscribes
Role

The asset store organization does not subscribe directly to any policy groups. As a result it inherits the management and administration policy group from the root organization. These policies apply to the asset store organization and the asset stores that it owns. The asset store organization owns the supplier profile policy group, but does not subscribe to it.

Note: The individual supplier's B2B direct organization will subscribe to the supplier profile policy group when the supplier store is created.


o=Root organization
o=Buyer A organization
o=Supplier organization
Buyers
Buyers registered with store A have Registered Customer role in B2B direct
Management and administration policy group
Common shopping policy group
B2B policy group
o=Supplier A organization
Supplier A administrators
Supplier users have Registered Customer role in supplier hub organization
o=Supply chain management organization
ou=B2B direct organization
ou=Supplier hub organization
ou=Asset store organization
Store A
Supplier profile policy group
Legend
Owns
Subscribes
Role

The supplier organization does not subscribe directly to any policy groups. As a result it inherits the management and administration policy group from the root organization. These policies apply to the supplier organization, the supplier A organizations that it owns, and the supplier A administrators.

The B2B direct organization subscribes directly to the management and administration, the common shopping, B2B and supplier profile policy groups. These policies apply to all stores owned by the B2B direct organization.

The Supplier profile policy group contains the following policies:

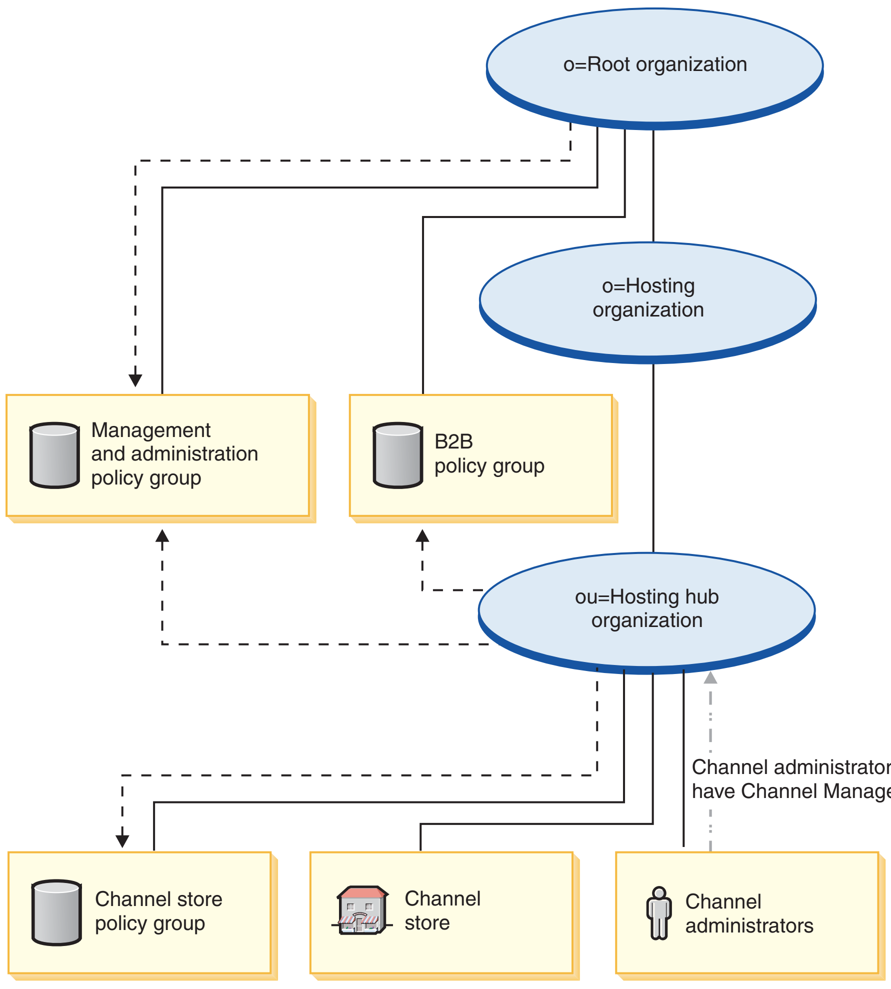
- AllUsersForSupplierExecuteSupplierAllUsersViews
- RegisteredCustomersForOrgForSupplierExecuteSupplierRegisteredCustomerViews

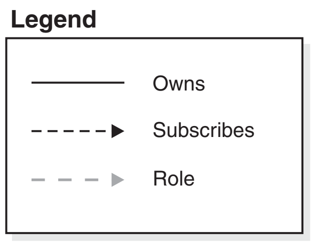
Buyers are customers that place orders in a B2B store. All buyers must be owned by a buyer organization. Typically, buyer organizations do not subscribe to any policy groups, since management and administration policies inherited from the root organization are sufficient.

Hosting

Business

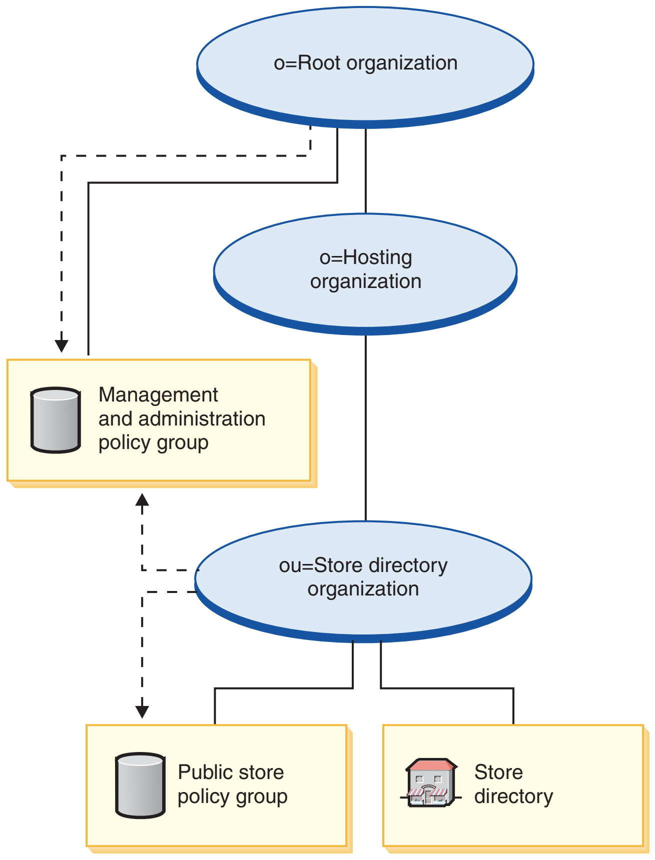
In these diagrams, describing a basic hosting organization structure, the root organization owns and subscribes to the default policy groups as described in "Basic access control structure" on page 38.

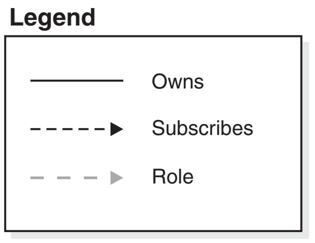

o=Root organization
o=Hosting organization
Management and administration policy group
B2B policy group
ou=Hosting hub organization
Channel administrators have Channel Manager role
Channel store policy group
Channel store
Channel administrators



The hosting hub subscribes directly to the management and administration policy group, the B2B policy group, and owns and subscribes to the channel store policy group. As a result, these policies apply to the channel administrators, who are directly under the hosting hub organization, as well as to the channel store (hosting hub).

The hosting hub policy group contains the following policies:
- AllUsersExecuteChannelStoreAllUsersViews
- ContractAdministratorsForChannelOrgExecuteCreate CommandsOnMemberResource
- ContractAdministratorsForChannelOrgExecuteContract DeployCommandsOnContractResource
- ContractAdministratorsForChannelOrgDisplayContract DatabeanResourceGroup

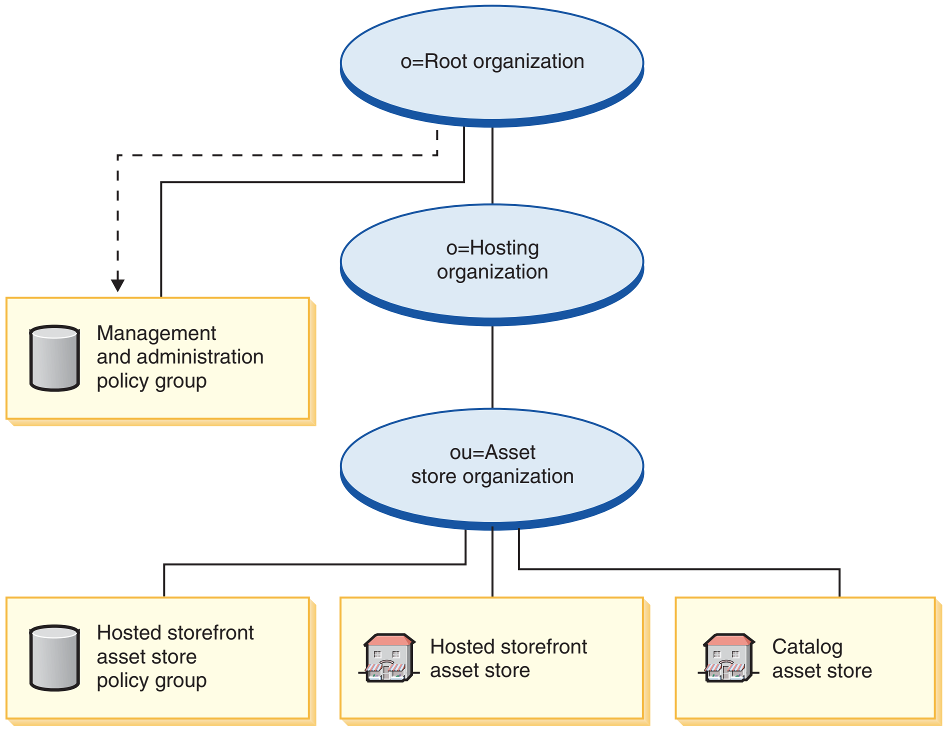

o=Root organization
o=Hosting organization
Management and administration policy group
ou=Store directory organization
Public store policy group
Store directory

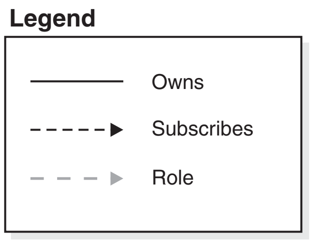


The store directory organization subscribes directly to the management and administration policy group and owns and subscribes to the store directory policy group. As a result, these policies apply to the store directory, which is directly under the store directory organization.

The store directory policy group contains the following policy:

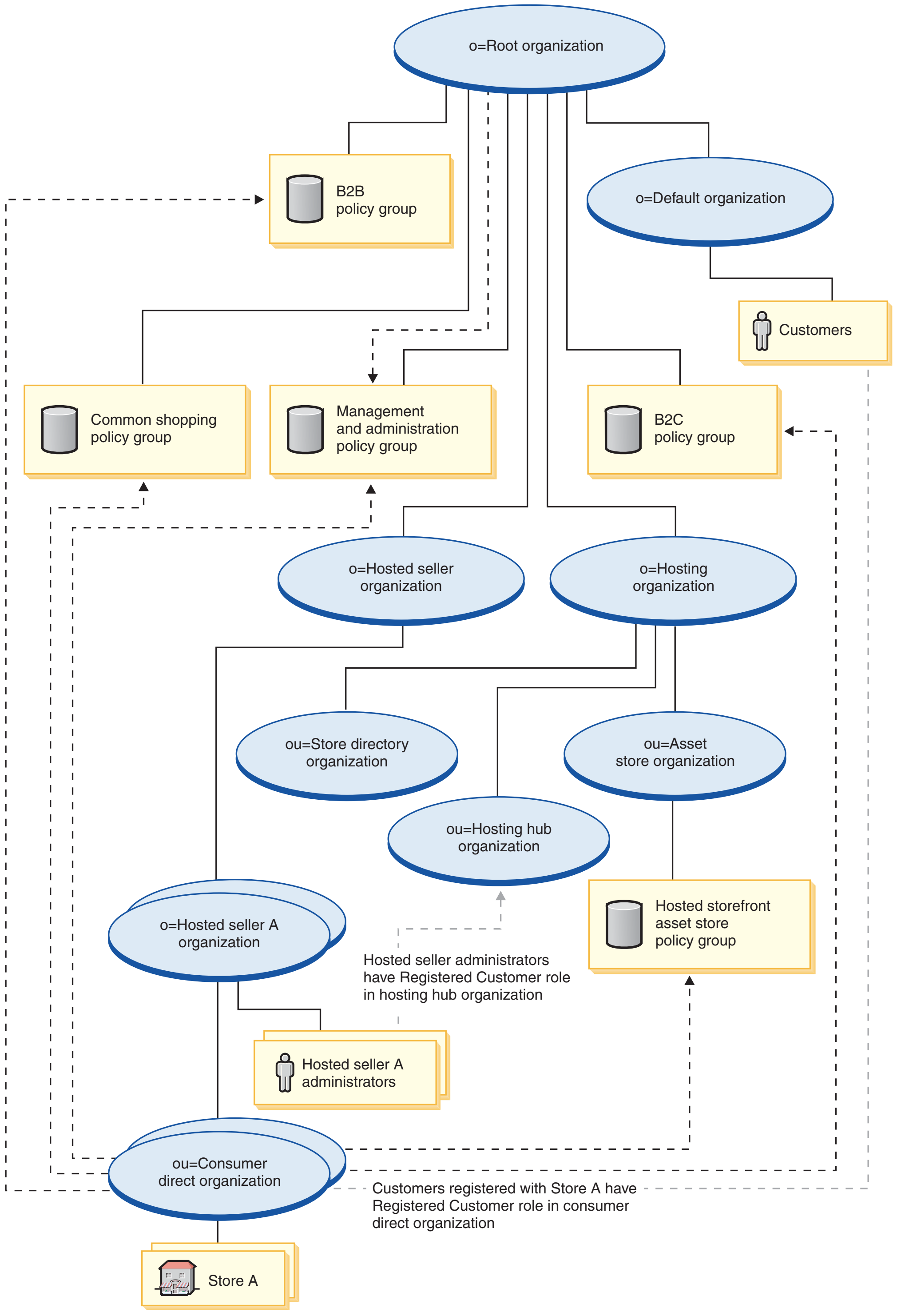
- AllUsersExecutePublicStoreAllUsersViews

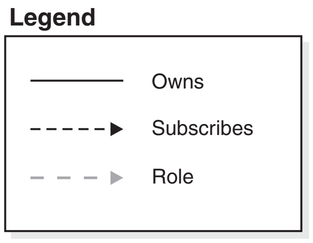

o=Root organization
o=Hosting organization
Management and administration policy group
ou=Asset store organization
Hosted storefront asset store policy group
Hosted storefront asset store
Catalog asset store



The asset store organization does not subscribe directly to any policy groups. As a result it inherits the management and administration policy group from the root organization. These policies apply to the asset store organization and the asset stores that it owns. The asset store organization owns the hosted storefront asset store policy group, but does not subscribe to it.

Note: The individual hosted seller organizations will subscribe to the hosted storefront asset store policy group when the hosted store is created.


o=Root organization
B2B policy group
o=Default organization
Customers
Common shopping policy group
Management and administration policy group
B2C policy group
o=Hosted seller organization
o=Hosting organization
ou=Store directory organization
ou=Asset store organization
ou=Hosting hub organization
Hosted storefront asset store policy group
o=Hosted seller A organization
Hosted seller administrators have Registered Customer role in hosting hub organization
Hosted seller A administrators
ou=Consumer direct organization
Customers registered with Store A have Registered Customer role in consumer direct organization
Store A



The hosted seller organization does not subscribe directly to any policy groups. As a result it inherits the management and administration policy group from the root organization. These policies apply to the hosted seller organization and the hosted seller A organizations that it owns, as well as to the hosted seller A administrators.

The consumer direct organization subscribes directly to the management and administration, the common shopping, B2B and B2C policy groups, as well as to the hosted storefront asset store policy group. These policies apply to all stores owned by the consumer direct organization.

Access control in sample businesses

Each of the sample businesses in WebSphere Commerce includes the access control framework. For more detail on how the access control framework is implemented in these businesses, see the *WebSphere Commerce Sample Store Guide*

Adding access control to your stores

For more information on adding access control to your stores, see Chapter 33, “Access control in your store,” on page 285.

Chapter 5. WebSphere Commerce business policy framework

▶Business Business policies are sets of rules followed by a store or group of stores that define business processes, industry practices, the scope and characteristics of a store or group of stores offerings, and how the store or site interacts with customers and other business partners. For example, your site may have business policies determining when and how customers are allowed to return products to a store, or business policies that determine what payment methods your store accepts.

Understanding the WebSphere Commerce business policy framework

WebSphere Commerce provides a framework that allows you to implement your store's business policies in your online store or site. The business policy framework consists of the following parts:

- Business policies
- ▶Business Business accounts
- Contracts and ▶Business service agreements
- Terms and conditions

Business policies

In most instances, you will have predefined business policies for your business that you need to implement in your online store or site. WebSphere Commerce provides a set of business policies that you can use as is, or change to meet your needs. For more information on the default business policies provided with WebSphere Commerce, see the WebSphere Commerce Production and Development online help. For information on how to edit these business policies, see WebSphere Commerce Production and Development online help.

Business Accounts

▶Business

Business accounts define the relationship between a customer and your business. Business accounts track contracts and orders for customer organizations and configure how buyers from customer organizations shop in a store.

Contracts and service agreements

Before a customer or business partner (for example resellers or distributors) can access your store, you must create a contract or service agreement that defines customer or business partner access to your store. In the WebSphere Commerce business policy framework, you create contracts for customers and service agreements for other types of business partners.

- Contracts: A contract with a customer defines what areas of your store the customer can access, what prices the customer will see, and for how long the customer has access to your site and those prices. All stores must contain at least one contract, as without a contract no one but internal administrators can access your store. WebSphere Commerce provides a default contract that applies to all customers shopping at a store. In WebSphere Commerce Professional Edition, the default contract is the only supported contract.

- Business Service agreements: A service agreement with a business partner (business partners may be resellers, distributors, manufacturers, suppliers, or other partners) defines your arrangement with the business partner. For example a service agreement with a reseller may define what access the reseller has to your site, whether they can share your catalog, or whether you host a store for them. A service agreement with a distributor may define how customers to your site can receive quotes from a distributor, or how customers can access the distributors site from yours.

Terms and conditions

Terms and conditions define how contracts and service agreements are implemented for a particular customer or business partner. For contracts, terms and conditions may define what is being sold under the contract; the price of the items being sold; how the items are shipped to the customer; and how the customer pays for the order. For service agreements with business partners, terms and conditions may restrict the products the business partner is allowed to sell.

Terms and conditions usually reference business policies as most aspects of a site or stores operations are defined by business policies. Terms and conditions provide standard 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 in sample businesses

Each of the sample businesses in WebSphere Commerce includes the business policy framework. For more detail on how the business policy framework is implemented in these businesses, see the *WebSphere Commerce Sample Store Guide*.

Adding business policies to your site

For more information on implementing the business policy framework in your site, see Chapter 18, “Contract assets,” on page 179.

Chapter 6. Instance architecture

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

WebSphere Commerce Server

The WebSphere Commerce Server is a WebSphere Application Server application that handles the store-and commerce-related functions of an e-commerce solution. The storefront assets and business logic reside in a Web application within the WebSphere Commerce Server. WebSphere Commerce provides a default Web application (`Stores.war`) 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 business logic, the assets for each store are separated by store directory (storedir).

WebSphere Commerce Server instance

A WebSphere Commerce Server instance is a deployed 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.

You can create a single store in an instance, or you can create multiple stores in an instance. For more information on multiple stores in instance, see "Multiple stores in a single instance" on page 64.

Chapter 7. Store architecture

In order to support creating online stores, WebSphere Commerce provides a store architecture. This architecture, as well as some examples of stores that can be implemented using it, are described in this chapter.

Understanding the WebSphere Commerce store architecture

In order to support stores in your site, WebSphere Commerce provides a store architecture that allows you to create online stores. The store architecture consists of the following components:

- Store assets
- Support for multiple stores in a single instance
- Relationships between stores

Store assets

In WebSphere Commerce an online store is the place where all transactions for your online business occur. All online stores created with WebSphere Commerce include at least one of the following types of assets:

- Storefront: The external portion of your store, or the portion that displays to your customers, is known as the storefront. The storefront 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 Part 4, "Developing your storefront," on page 73.
- Business logic: The portion of your store that processes customer requests, including the commands, customized code, is known as the business logic. For more detailed information on creating business logic or customized code see the *WebSphere Commerce Programming Guide and Tutorials*.
- 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. Your store must also have methods for processing and collecting payment. The concepts and tasks involved in creating store data are discussed in Part 6, "Developing your store data," on page 107.

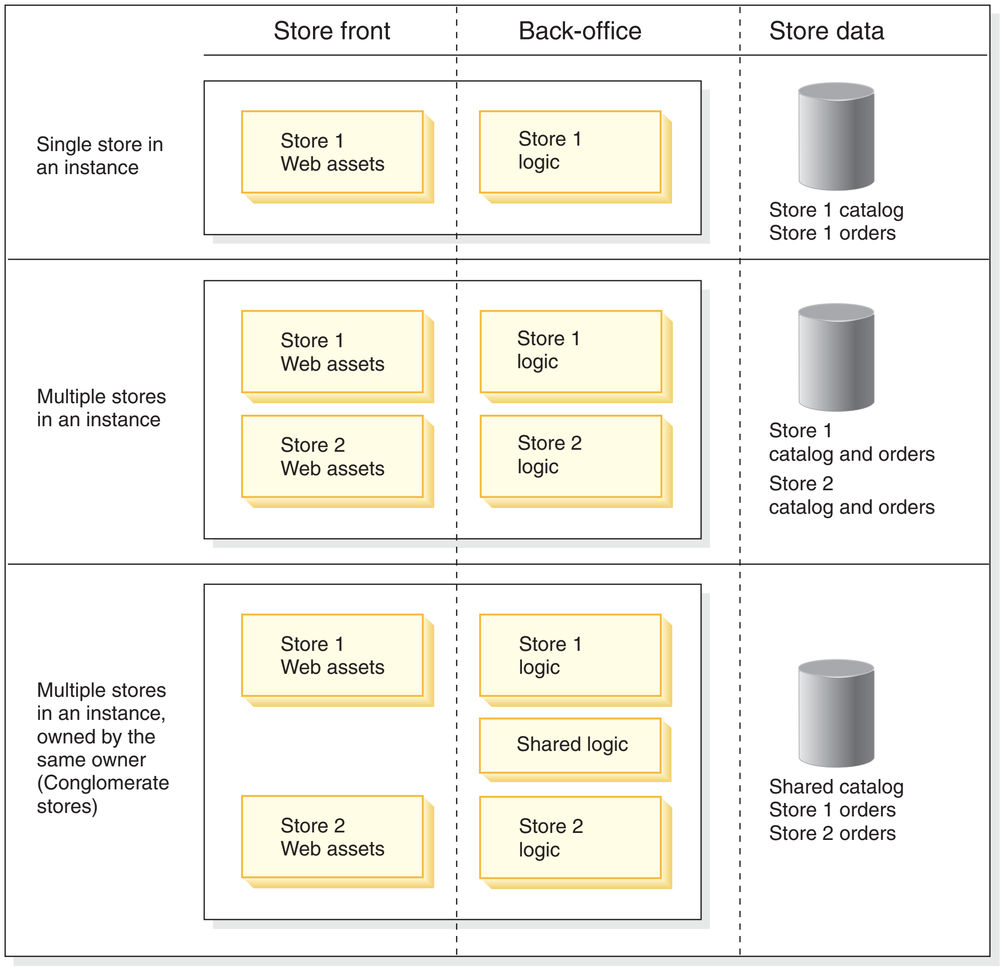
If a store contains all three types of assets, that is storefront assets, business logic, and store data, it is a fully operational store. If a store contains only a subset of the assets, that is it contains storefront assets and business logic, or store data and business logic, or just store data, it is known in WebSphere Commerce as an *asset store*.

Asset stores

Asset stores are collections of sharable resources (business artifacts, business processes and storefront assets) that can be leveraged in other stores. For example, instead of creating a catalog as part of the hub store, a hub store may leverage a catalog asset store, which can also be shared by the hub's channels or partners. An asset store is usually composed of the assets that can be used by multiple stores. For more information, see "Relationships between stores" on page 66.

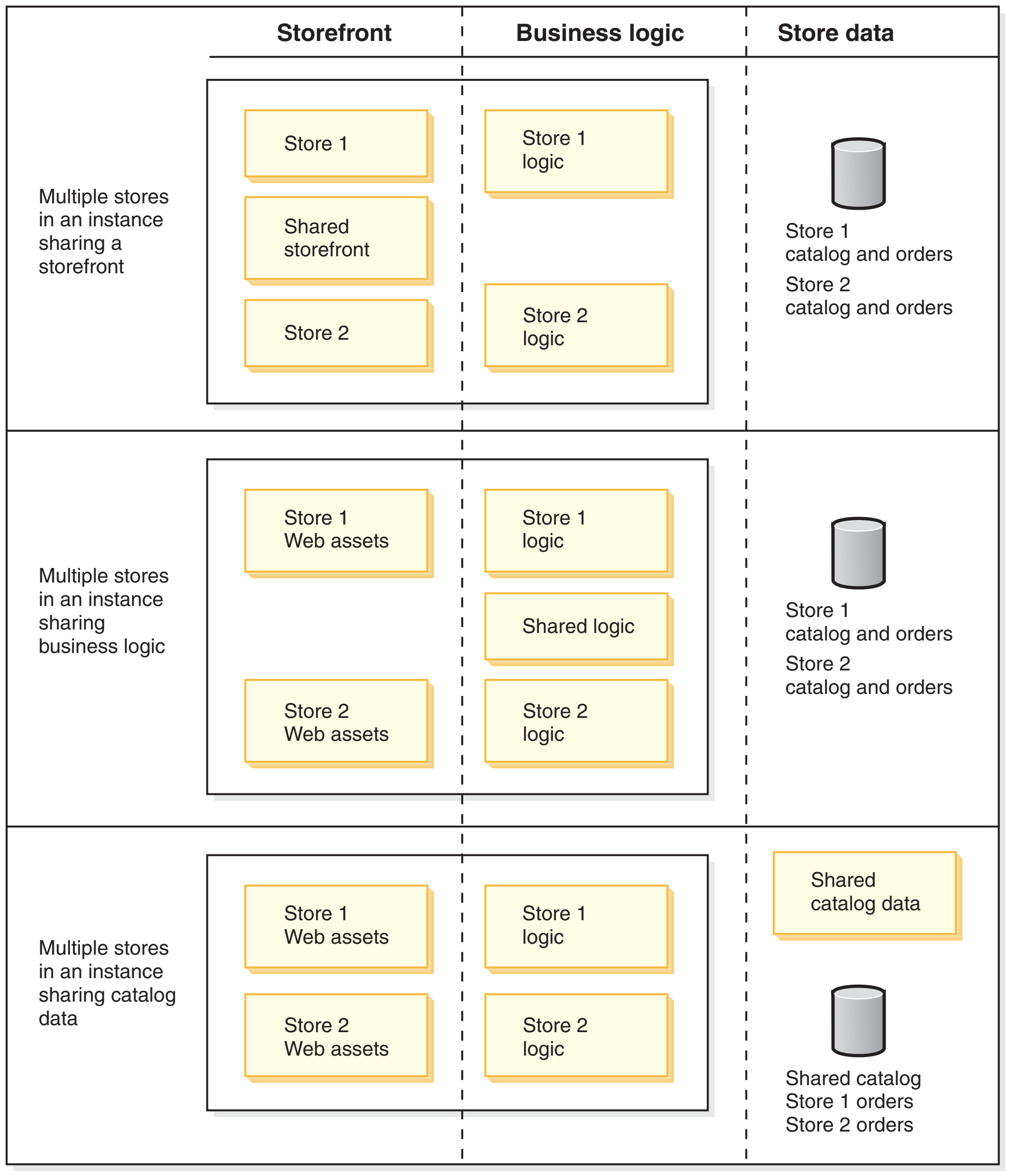
Multiple stores in a single instance

WebSphere Commerce allows you to support multiple online stores within your WebSphere Commerce Server instance. The following diagram illustrates some possible store configurations:



The stores detailed in the preceding diagram are stand alone stores. That is, although they are in the same instance, they do not share any data or have relationships with each other. They have separate storefronts, business logic and store data.

You can also create multiple stores in an instance that share the same storefront, the same business logic, or the same store data, including catalogs, or any combination of the three. The following diagram illustrates some possible configurations in which stores share assets:



Note: The preceding diagram only lists a few possible configurations between multiple stores in an instance. Stores may share more than one asset type, for example multiple stores in a site could share storefronts, business logic and data, or any combination of the three.

For more information on how multiple stores in an instance share common store assets, see "Relationships between stores" on page 66.

Multiple stores can exist in a single Stores Web module. If so, the store assets are separated using the following methods:

- Storefront assets: Storefront assets for each store in the Stores Web module are stored in a separate store directory (*storedir*). For example all storefront assets for MyStore are in the MyStore directory.
- Business logic: The store ID is used to select the command implementation for each store, as specified in the command registry.

- Store data: Data assets are identified for each store by a unique index.

Relationships between stores

Business In order to support multiple stores in a site having the same storefront, business logic or store data or any combination of shared assets, as well as supporting other types of relationships between stores in a site, such as one store hosting another, or transferring shopping carts from one store to another, WebSphere Commerce now provides the architecture for a variety of relationships between stores.

Relationships between stores allow one store to provide a service to another store. For example store A may host store B, or store C may use the catalog data from store D.

In order to implement these store relationships, code that supports each store relationship is required. WebSphere Commerce includes many store relationships and the supporting code. These store relationships can be loosely grouped into the following categories:

- Relationships in which one store provides assets to another store. These types of store relationships include one store providing URLs, commands, business policies, property files, and currencies to another.
- Relationships in which one store has a ″business relationship″ with another store. These types of store relationships including one store hosting another, or one store referring orders to another store.

Note: For a detailed list of the default store relationships provided with WebSphere Commerce, see Chapter 14, “Relationships between stores,” on page 129.

Understanding how the store architecture supports the business models

In order to support the stores needed for the business models, WebSphere Commerce uses the store architecture to create the following types of stores:

- Customer facing stores
- Proxy stores
- Asset stores

Note: These particular stores are recommended for implementing the business models supported by WebSphere Commerce. You can also create your own types of stores using the store architecture.

Customer facing stores

Customer facing stores are stores that customers can access directly. These stores are the main components of your site. WebSphere Commerce supports the following types of customer facing stores:

- Direct sales store: A store that supports commerce transactions involving products, services, or information directly between businesses and consumers, or between two businesses or parties. WebSphere Commerce supports two types of direct sales stores:
  - Consumer direct
  - Business B2B direct

- Business Hub store: A store that enables its customers or partners to access products or services available from one or more partners or clients of the hub owner, through the use of other stores on the site.
- Business Hosted store: A store that is hosted by the site operator for the owner of the store. The store owner may have the option of administering the store.

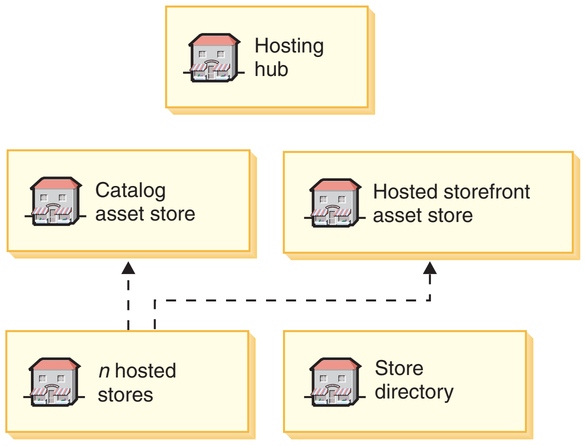
Creating direct sales and hub stores

Direct sales and Business hub stores are the most traditional stores in WebSphere Commerce in terms of store creation. That is, you need to create storefront assets, business logic and store data for each store. You have the option of creating these assets traditionally, by creating the assets for that store only. However, you also have the option of creating the assets to be used by other stores, by creating the storefront and business logic assets either in an asset store or as data that can be used across stores. You may also want to use assets from other stores to create portions of your direct sales or hub store.

For information on creating storefront assets, see Part 4, "Developing your storefront," on page 73. For more information on creating business logic or customized code see the *WebSphere Commerce Programming Guide and Tutorials*. For more information on creating store data, see Part 6, "Developing your store data," on page 107. For more information on sharing assets between stores, see Chapter 14, "Relationships between stores," on page 129.

Creating hosted stores

Business In the samples provided with WebSphere Commerce, the majority of the hosted store is created by sharing assets from existing asset stores. For example, rather than creating the storefront or catalog assets for each store you are hosting, you use the storefront and, depending on your business, the catalog from another store. In order to facilitate creating hosting stores, WebSphere Commerce uses asset stores. The following diagram illustrates how hosted stores use the assets from the hosted storefront asset store and the catalog asset store.



Your hosted business administrators then have the option of making cosmetic changes (such as a new look and feel, their own new logo and some of their own text) to customize their store, as well as changing certain data (filtering the catalog, changing prices and so on).

You can also create hosted stores traditionally, that is by creating the storefront assets, business logic, and store data separately for each hosted store. For information on creating storefront assets, see Part 4, "Developing your storefront," on page 73. For more information on creating business logic or customized code see the *WebSphere Commerce Programming Guide and Tutorials*. For more information on creating store data, see Part 6, “Developing your store data,” on page 107.

The Store Creation wizard: The Store Creation wizard provided with WebSphere Commerce allows you to create hosted stores quickly and easily. The wizard asks a customer to provide some basic data about their store (name, description, and so on), allows the customer to select the storefront or catalog they want to use, and then creates the store for them. The resulting store has some unique data (basic store data that makes it a unique store), but uses the storefront and catalog data from existing asset stores.

The Store Creation wizard’s behavior is governed by a template, which determines what options are available for creating the hosted store, including store relationships, shipping modes, messages, and shared fulfillment center. WebSphere Commerce provides several templates for the Store Creation wizard, one for each supported business model. These templates are located in the following directory:

WC_installdir/xml/trading/xml

A template is associated with the Store Creation wizard, based on the type of storefront asset store chosen in the wizard. For example if you choose to use assets from the reseller storefront asset store (identified as RPS in the STORETYPE field in the STORE table) the Store Creation wizard uses the TemplateHostingContractRPS.xml.

For information on creating a hosted store using the Store Creation wizard, see the WebSphere Commerce Production online help.

Note: If you prefer not to use the Store Creation wizard to create hosted stores, you can create a service agreement based on one of the templates and then import it into WebSphere Commerce. For information, see the WebSphere Commerce Production online help.

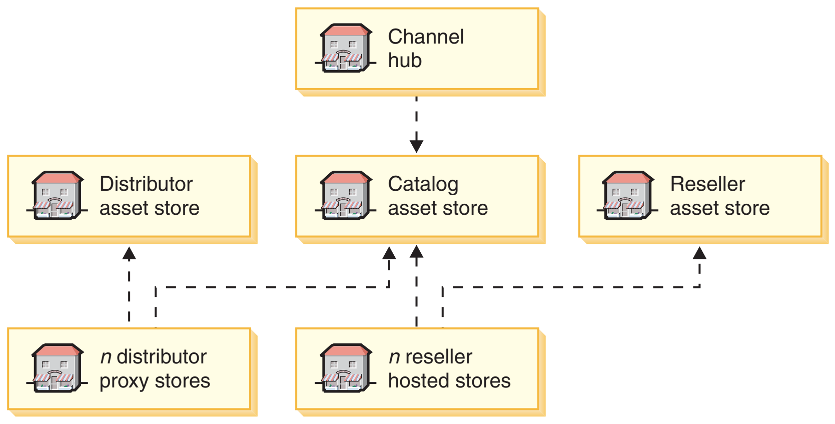
In order to change the assets that the hosted store shares, you must change the asset store. For more information, see “Creating asset stores” on page 69.

Proxy stores

WebSphere Commerce also supports entities known as proxy stores. A proxy store is a store that represents a business partner’s operational assets, provides the business logic that allows the WebSphere Commerce site to interact with an external business partner. For example, a proxy store may capture the orders transferred to a remote order capture system, as well as capturing the suppliers’ inventory information or the information sent to a supplier’s fulfillment centers. Unlike a customer facing store, a proxy store does not include a storefront and cannot be accessed by users.

Creating proxy stores

Creating a proxy store is very similar to creating a hosted store, in that the majority of the proxy’s stores assets are provided from existing stores (including asset stores). As implemented in the samples provided with WebSphere Commerce, the proxy store does not include a storefront. As a result, only the assets from another store’s catalog are shared. The following diagram illustrates the distributor proxy stores using the assets from the distributor asset store and the catalog asset store.



Rather than providing a user interface to create a proxy store, WebSphere Commerce implements proxy stores through service agreements, which are then imported into WebSphere Commerce, creating the proxy store. The service agreement is governed by a template, which determines what information you need to create. The template for creating proxy stores (TemplateReferralContract.xml) is available in the following directory:

WC_installdir/xml/trading/xml

To create the proxy store, create a new service agreement following the template and then import it into WebSphere Commerce. For more information, see the WebSphere Commerce Production online help.

Asset stores

In order to facilitate the creation of customer facing stores and proxy stores, WebSphere Commerce implements asset stores. Asset stores are collections of sharable resources (business artifacts, business processes and storefront assets) that can be leveraged in other stores. For example, instead of creating a catalog as part of the hub store, a hub store may leverage a catalog asset store, which can also be shared by the hub's channels or partners. An asset store is usually composed of the assets that can be used by multiple stores. For more information, see "Relationships between stores" on page 66.

WebSphere Commerce provides sample catalog asset stores and storefront asset stores.

Creating asset stores

Asset stores are stores that provide assets to another store. As implemented for the samples provided with WebSphere Commerce, asset stores are composed of a collection of assets, but are not fully functional stores. To create an asset store, you follow the same methods as you would to create the assets in a direct sales or hub store. That is, if you want the asset store to contain catalog assets, you create catalog data following the instructions in Part 6, "Developing your store data," on page 107. If the asset store will contain storefront assets, see Part 4, "Developing your storefront," on page 73. If the asset store will contain business logic, see *WebSphere Commerce Programming Guide and Tutorials*.

Stores in the supported business models

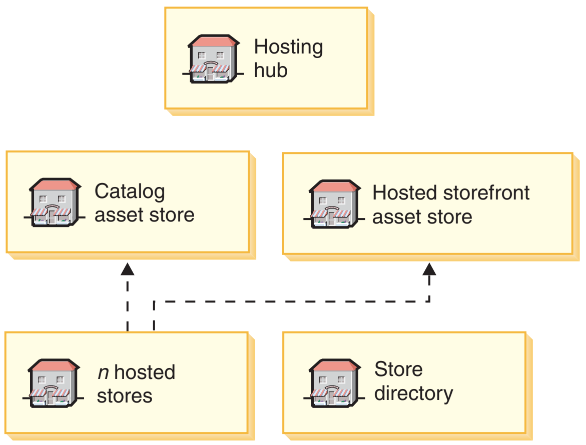
The following sections illustrate how stores are implemented in the sample businesses.

Note: Since the consumer direct and Business B2B direct samples each contain one direct sales store, they are not discussed here.

Hosting

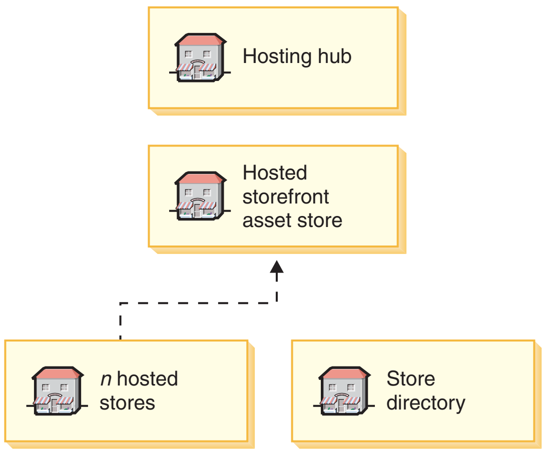
Business

The following diagram illustrates the types of stores that compose the hosting sample.



The sample hosting site contains a hub store (hosting hub), two asset stores (catalog asset store and the hosted storefront asset store) as well as the store directory. The store directory is a listing of all the hosted stores in the site and acts as a gateway to them. The hosting stores are created by using the assets from the two asset stores.

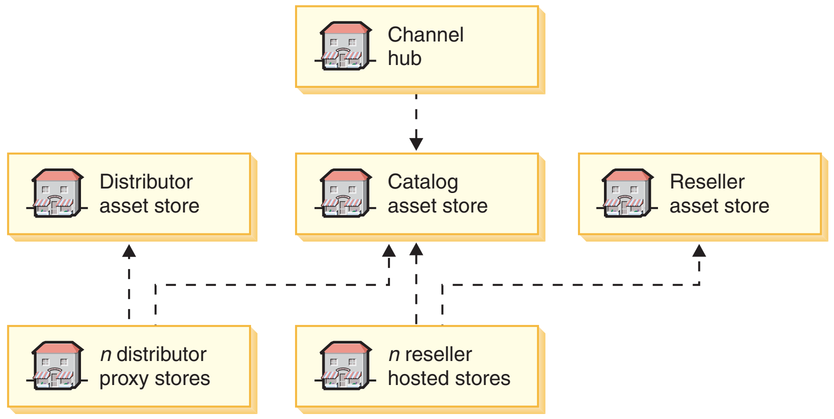
Note that customers may choose to create their own catalog data, rather than using the catalog defined in a catalog asset store. This variation creates a second implementation of the hosting site, as illustrated in the following diagram:



Demand chain

Business

The following diagram illustrates the types of stores that compose the demand chain sample.

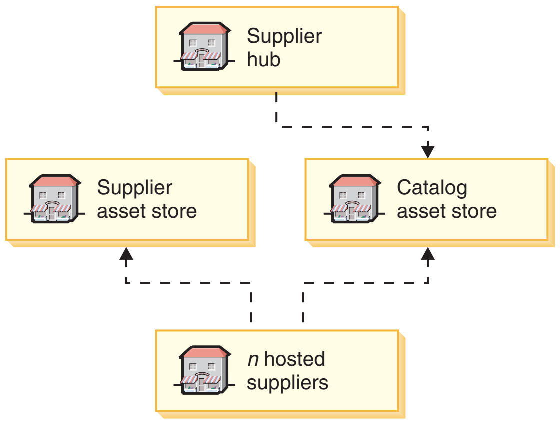


The demand chain sample site contains a hub store (channel hub), and three asset stores (distributor asset store, catalog asset store and reseller storefront asset store). Note that the channel hub uses the catalog assets defined in the catalog asset store. The distributor proxy stores are creating by using the assets from the distributor asset store, while the reseller hosted stores are created by using the assets from the catalog asset store and reseller storefront asset store.

Supply chain

Business

The following diagram illustrates the types of stores that compose the supply chain sample.



The supply chain sample site contains a hub store (supplier hub), and two asset stores (catalog asset store and supplier asset store). Note that the supplier hub uses the assets defined in the catalog asset store. The hosted suppliers are created by using the assets from the catalog asset store and supplier asset store.

Note: The supplier hub owner defines the catalog taxonomy (for example the category structure, and possibly shared products and items) that the hosted suppliers will use in the catalog asset store.

Part 4. Developing your storefront

Chapter 8. Developing your storefront

This chapter provides an overview of the WebSphere Commerce storefront 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.

Storefront 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 WebSphere Commerce Stores Web Application.

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

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:

- *WC_installdir*/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 *WebSphere Commerce Programming Guide and Tutorials*.

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 access control policies. For more information, see Chapter 33, “Access control in your store,” on page 285.

Note: While planning your store pages, you should also create a caching strategy. For more information on caching, see Chapter 9, “Caching your store pages,” on page 83.

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.

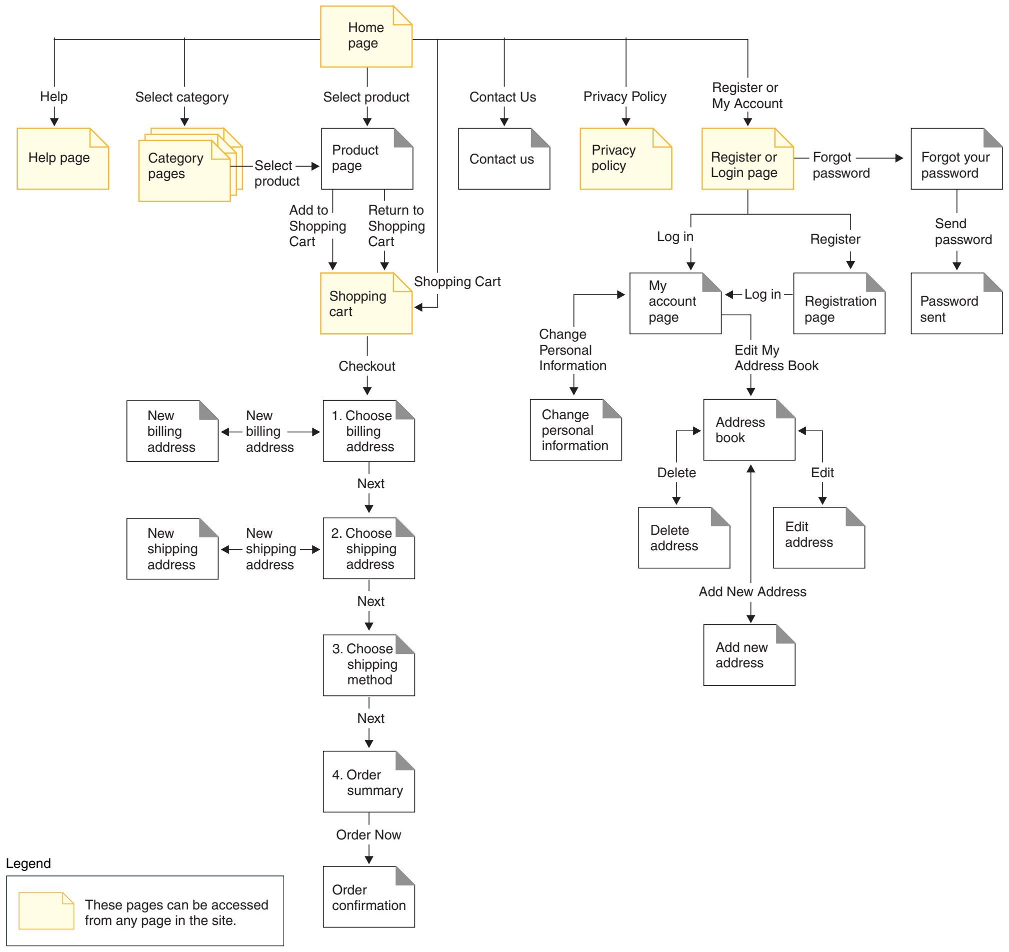
Note: Since use cases often contain flow information such as, ″If the customer selects **Submit**, the Order page displays,″ use cases can provide useful information for creating shopping flow diagrams.

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 FashionFlow sample store's shopping flow does, allows you to see how customers will travel through your store.

Note: This diagram only contain a portion of the FashionFlow store flow. For the complete flow, see the *WebSphere Commerce Sample Store Guide*.



The diagram for the FashionFlow 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 FashionFlow diagram, you would have to create the following pages:

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

FashionFlow 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

FashionFlow shopping flow diagram pages (as seen by customer)	Corresponding view
Password sent	ResetPasswordForm
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	BillingAddressView
New billing address	AddressForm
Choose shipping address	MultipleShippingAddressView
New shipping address	AddressForm
Choose shipping method	MultipleShippingMethodView
Order summary	AllocationCheck
Order confirmation	OrderOKView

Note: Many of the views used in FashionFlow were created specifically for FashionFlow. These views are listed in the `command.xml` file in the FashionFlow store archive. For more information, see "Registering commands, views, and URLs in WebSphere Commerce" on page 137.

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 FashionFlow 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.

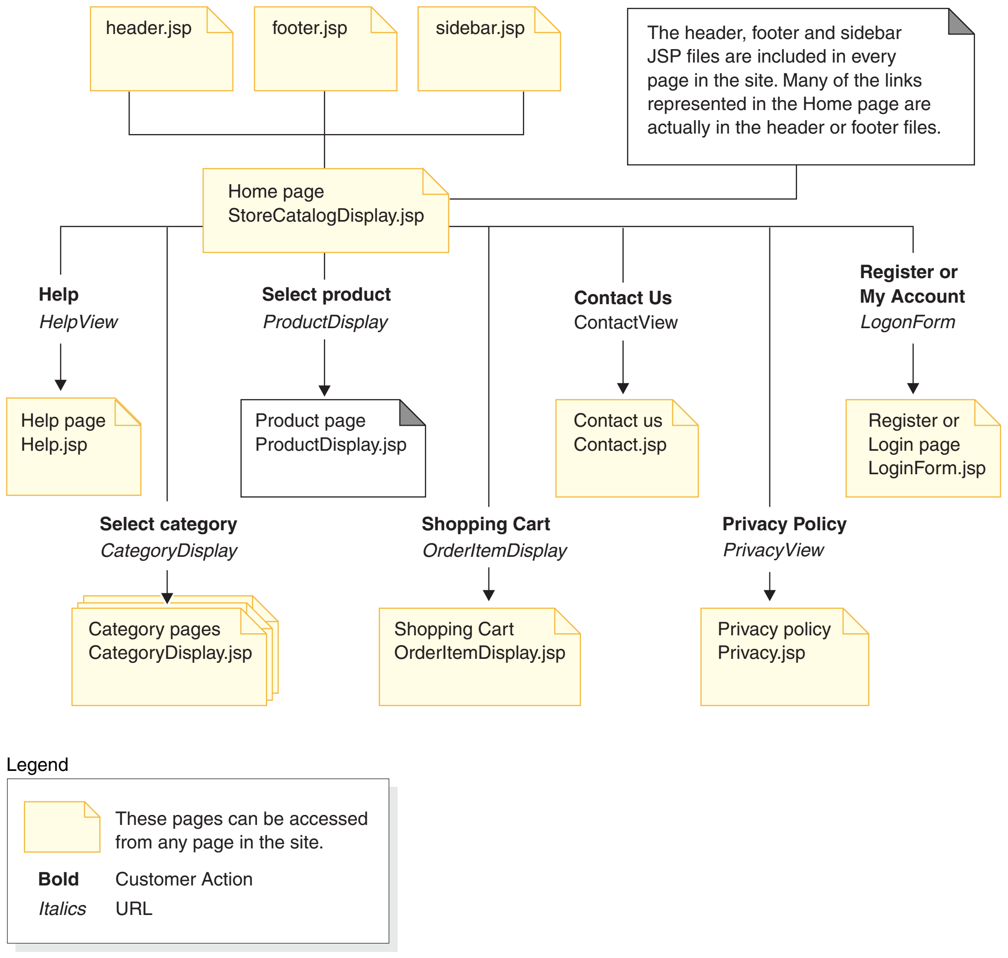
Developing a list of command and view URLs

As demonstrated in the FashionFlow 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 FashionFlow shopping flow diagram. For more details, see the information on the samples stores in the*WebSphere Commerce Sample Store Guide*.


header.jsp
footer.jsp
sidebar.jsp
The header, footer and sidebar JSP files are included in every page in the site. Many of the links represented in the Home page are actually in the header or footer files.
Home page StoreCatalogDisplay.jsp
Help
HelpView
Help page Help.jsp
Select category
CategoryDisplay
Category pages CategoryDisplay.jsp
Select product
ProductDisplay
Product page ProductDisplay.jsp
Shopping Cart
OrderItemDisplay
Shopping Cart OrderItemDisplay.jsp
Contact Us
ContactView
Contact us Contact.jsp
Privacy Policy
PrivacyView
Privacy policy Privacy.jsp
Register or My Account
LogonForm
Register or Login page LoginForm.jsp
Legend
These pages can be accessed from any page in the site.
Bold Customer Action
Italics URL

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 FashionFlow, 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.

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 FashionFlow, 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"
/>
```

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 137.

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 162. You can optionally assign different JSP filenames to display products and categories for each member group or language supported by your store.

The HttpRedirectViewCommandImpl view command is used to render the output of a non-display command (a command that changes the database). A non-display command must be associated with a display command to avoid the command being re-executed accidentally if the customer reloads the page or click the back button.

To redirect to a display command, specify the display command using the `&URL=` parameter on the URL of the non-display command. For example, when you add address information in the FashionFlow sample store Address form and click **Submit**, it invokes the AddressAdd command. The URL used to invoke the AddressAdd command specifies AddressBookForm command as the `&URL=` parameter. This results in a redirect to the AddressBookForm display command, which returns 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.

Chapter 9. Caching your store pages

While developing your storefront you also need to determine how to cache your store pages. This chapter discusses creating and implementing a caching strategy for your store.

Planning your caching strategy

When determining a caching strategy, you first need to consider the following issues:

- What pages should be cached
- Should pages be cached as whole pages or page fragments

What pages should be cached

When creating your high level caching strategy, you first need to determine what pages in your store should be cached. Pages that are good candidates for caching are pages that are accessed frequently, but are also stable for a period of time, and contain content that can be reused by a variety of users. For example, catalog display pages are usually good candidates to enable caching.

Should pages be cached as whole pages or page fragments

In Version 5.5, WebSphere Commerce uses the WebSphere Application Server dynamic cache service, which allows WebSphere Commerce to support both caching of Web pages as a whole, and caching of fragments of pages. Caching a Web page as a whole simply caches the entire page as one entity, even if it is composed of several smaller fragments. Page fragments may include a separate header, sidebar, or footer. Even the main body of the page may be broken into several fragments. For example one fragment on the main body page may show a product, while the a second fragment shows the price. Fragmenting pages allows you to show content personalized for individual users. The sample store pages provided with WebSphere Commerce are composed of several fragments (header, sidebar, footer, main content).

If your store pages are composed of fragments you also have the opportunity of caching the pages by fragments. Caching individual fragments allows you to cache the portions of the page that are reusable for a wider audience. If a page contains personalized information for only a small segment of your audience, caching this page as a whole page will not allow the page to be reused very often because only that segment of your audience can ever reuse that cached page. For example, if a page displays a welcome message for each customer in the header and is cached based on the user ID, then only that particular user can ever reuse that cached page. However, if you decompose that page into fragments, you can cache the fragments that get reused for most of your audience. For example, the footer, sidebar, and product display fragments may be be applicable to all your users, while the price and header fragments may be personalized.

When the page is requested, the individual fragments are reassembled to produce the page.

Your store pages can be cached using whole page caching or fragment caching or a combination of the two methods.

Developing a more detailed caching strategy

After you have determined what pages and page fragments should be cached, you need to determine a more detailed caching strategy. For each page or fragment your are planning to cache, you need to determine the following:

- How the page or fragment is requested
- Whether the page or fragment relies on a store relationship
- How the cached data will be invalidated

How the page or fragment is requested

How the JSP file (whether it is a single page, or a page fragment) is requested determines how the WebSphere Application Server will cache it. For example, the WebSphere Application Server needs to know whether the JSP file is displayed as a response to a servlet, object, EJB, or a command. As a result, you need to compile a list of how each page or fragment you plan to cache will be requested.

Whether the page or fragment relies on a store relationship

Business As discussed in Chapter 14, "Relationships between stores," on page 129 and Chapter 7, "Store architecture," on page 63, stores may have relationships with other stores that allow them to use data from another store. For example store A may use the catalog data defined in store B. Stores may also have relationships with multiple stores, allowing them to use data from several different sources. As part of your detailed caching plan, you need to determine if the data displayed in each page or fragment relies on a relationship with another store. If a page does display information from another store, each time the data from other store is updated, your cached pages will also need to be updated. For information on caching store relationships, see "Implementing caching for store pages that use store relationships" on page 89.

How the cached data will be invalidated

For each page or page fragment that you plan to cache, you also need to determine when the cached page or fragment is no longer valid, and remove the corresponding cache entries from the cache. This process is known as invalidation. In order to determine when a cached page has changed, and thus can no longer is valid, you need to determine what might make the cached page out of date. For example a cached shopping cart page is invalid when a customer adds a new item to the cart. Cached pages may also be invalidated when an administrator updates the store with the WebSphere Commerce Accelerator, or when new catalog data is added with the loader package, or the tooling in the WebSphere Commerce Accelerator.

After you have compiled a list of all possible ways the cached page or fragment can be invalidated, you need to determine which events are used to cause the invalidation. Events that cause invalidation can include a servlet request, a controller command or a task command and so on. For example, if you update a product description using the product management tools in the WebSphere Commerce Accelerator, WebSphere Commerce internally invokes the commands, AddCatalogEntryDescCmd or UpdateCatalogEntryDescCmd to make the changes. If you want to invalidate the cached pages that are changed by these commands, you need to add invalidation policies to the `cachespec.xml` file that will intercept the execution of the commands, and trigger the invalidation. For information on implementing invalidation, see the following:

- The *WebSphere Commerce Administration Guide*, "Dynamic caching" chapter for instructions on setting up new invalidation policies, and an example of cache invalidation.
- "Invalidating cached data in the cachespec.xml file" on page 88 for instructions on how to merge the sample invalidation policies provided by WebSphere Commerce with your store's `cachespec.xml` file.

Implementing your caching strategy

After you have gathered all the details you need for your caching strategy, you implement it by creating a cache policy file that defines the information you have gathered, including what is to be cached and how, and how cached pages will be invalidated. The WebSphere Application Server dynamic cache service uses this cache policy file, known as `cachespec.xml` to implement caching in your store.

Each sample store provided with WebSphere Commerce includes a `cachespec.xml` file that defines the caching strategy for that store. These files are located in the following directory:

WC_installdir/samples/dynacache/*BusinessModel*

You have the option of changing these files if your store is based on a sample, or using one of these files as a base for the `cachespec.xml` file for your store.

Understanding the cachespec.xml file

In order to cache WebSphere Commerce's store pages, you must define cacheable objects in the `cachespec.xml` file. WebSphere Commerce only uses a subset of the elements defined in the `cachespec.xml` file. This subset of elements is explained in this section. For more detailed information about the `cachespec.xml` file, see the WebSphere Application Server Information Center (http://www.ibm.com/software/webservers/appserv/infocenter.html), topic "Cachespec.xml file". For more information, see the *WebSphere Commerce Administration Guide*, "Dynamic caching" chapter.

Understanding the elements used by WebSphere Commerce

WebSphere Commerce used the following elements in the `cachespec.xml`:
- Class
- Name
- Property

The use of these four elements is illustrated in the following example:

```
<cache-entry>
   <class>servlet</class>
      <name>/FashionFlow/ShoppingArea/CatalogSection/CategorySubsection
/StoreCatalogDisplay.jsp</name>
      <property name="save-attributes">false</property>
```

Class: The class element is a required element. It determines how the WebSphere Application Server will interpret the remaining cache policy definition. WebSphere Commerce uses the following class values:
- command
- servlet

The value command refers to classes using the WebSphere Commerce programming model.

The value servlet refers to servlets or JSP files deployed in the WebSphere Application Server servlet engine.

Note: For WebSphere Commerce version 5.5, only command invalidation is supported.

Name: Name is the fully qualified class name of the servlet or command. Name is a required element.

Name values for commands must include the package name. For example, `com.ibm.commerce.dynacache.commands.MemberGroupsCacheCmdImpl`

Name values for servlet and JSP files must include the full URI of the JSP file or servlet to be cached. For example, `com.ibm.commerce.server.RequestServlet.class`

`/ToolTech/ShoppingArea`
`/CatalogSection/CategorySubsection/StoreCatalogDisplay.jsp`.

Property: The property element takes the following form: `<property name=`*`key`*`>`*`value`*`</property>`, where *key* is the name of the property being defined and *value* is the corresponding value. You can set optional properties on a cacheable object. For example,`<property name="consume-subfragments">true</property>`

When caching WebSphere Commerce store pages, the following properties are used:

Property	Value	Valid classes	Description
EdgeCacheable	True or False Default is false.	Servlet	If the property is true, then the given servlet or JSP file is externally requested from an Edge Server. Whether the servlet or JSP file is cacheable depends on the rest of the cache specification.

consume-subfragments	True or False. Default is false	Servlet	When a servlet is cached only the content of that servlet is stored. Placeholders for any other fragments to which it includes or forwards are created. Consume-subfragments (CSF) tells the cache to continue saving content when it encounters a child servlet via an include. The parent entry (the one marked CSF) will include all the content from all fragments in its cache entry, which result in one big cache entry that has no includes or forwards, but the content from the whole tree of entries. This method can save a significant amount of application server processing, but is typically only useful when the external HTTP request contains all the information needed to determine the entire tree of included fragments.
save-attributes	True or False. Default is true.	Servlet	When save-attributes is set to false, the request attributes are not saved with the cache entry.
store-cookies	True or False. Default is true.	Servlet	When store-cookies is set to false, the request cookies are not saved with the cache entry.

By default, DynaCache caches the cookies (when caching by servlet class) and all request attributes (servlet and JSPs) along with the cache entries. However, WebSphere Commerce cookies and request attributes contain user specific information that should not be cached. As a result, the following property names and values are mandatory when caching full pages:

```
<property name="save-attributes">false</property>
<property name="store-cookies">false</property>
```

The following property name and value is mandatory for all cache-entries defined for the JSPs files:

```
<property name="save-attributes">false</property>
```

Understanding cache-ID rules

A cache-ID uniquely identifies a cache entry. In order for the WebSphere Application Server to cache an object, it must know how to generate a unique ID for different invocations of that object. These IDs are created from either user-written custom Java code or from rules defined in a cache entry's cache policy.

In the `cachespec.xml` file the cache-id element defines the rules for generating IDs. Each cache entry may have multiple cache-ID rules that will execute in the defined order until either a rule returns a non-empty cache ID, or no more rules are left to execute. If none of the cache-ID generation rules produce a valid cache ID, then the object is not cached.

These IDs are developed in one of the following ways:

- Using the component elements defined in the cache policy of a cache entry
- Writing custom Java code to build the ID from input variables and system state

Understanding dependency-ID rules

Dependency ID elements specify additional cache group identifiers that associate multiple cache entries to the same group identifier. The dependency ID is generated by concatenating the dependency ID base string with the values returned by its component elements. If a required component returns a null value, then the entire dependency ID is not generated and is not used.

You can validate the dependency IDs explicitly through the WebSphere Dynamic Cache API, or by using another cache-entry invalidation element. Multiple dependency ID rules can exist per cache-entry. All dependency ID rules execute separately. For more information on how to define dependency ID rules, see the *WebSphere Commerce Administration Guide*, "Dynamic caching" chapter.

Understanding invalidation rules

Invalidation rules can be defined in exactly the same manner as dependency IDs. However, the IDs that are generated by invalidation rules are used to invalidate cache entries that have the same dependency IDs. The invalidation ID is generated by concatenating the invalidation ID base string with the values returned by its component element. If a required component returns a null value, then the entire invalidation ID is not generated and no invalidation occurs. Multiple invalidation rules can exist per cache-entry. All invalidation rules execute separately. For more information on how to define invalidation rules, see the *WebSphere Commerce Administration Guide*.

Invalidating cached data in the cachespec.xml file

By default, the `cachespec.xml` files shipped with the sample store archives do not include invalidation policies. If you would like to automate cache invalidation using DynaCache in a sample store, or a store based on a sample, you must add invalidation policies to the store's `cachespec.xml` file. Sample invalidation policies are provided in several `cachespec.xml` files in the following directory:

WC_installdir/samples/dynacache/invalidation

This directory contains separate `cachespec.xml` files for functional areas, including catalog, shopping cart, store and so on. Each file contains invalidation policies for that specific area.

If you plan to cache catalog pages in your store, you should add the invalidation policies from the following files into your store:
- *WC_installdir*/samples/dynacache/invalidation/catalog/cachespec.xml
- *WC_installdir*/samples/dynacache/invalidation/membergroup/cachespec.xml

  Note: For these member group invalidation rules you need to add additional dependency IDs to the cache entries. See the content of this `cachespec.xml` file for more details.
- *WC_installdir*/samples/dynacache/invalidation/store/cachespec.xml

Adding sample invalidation policies to your store's cachespec.xml file

In order to add the invalidation policies provided in the sample invalidation files into your store, do the following:
1. Open the `cachespec.xml` file for your store.
   - *WAS_installdir*/installedApps/*cell_name*/*WC_instanceName.ear*/ Stores.war/WEB-INF directory

   If your store does not have caching policies defined and is based on a sample provided with WebSphere Commerce, you can use a sample `cachespec.xml` file from the following directory:
   - *WC_installdir*/samples/dynacache/*BusinessModel*
2. Open sample invalidation `cachespec.xml` file. The sample invalidation `cachespec.xml` files are located in the following directory:
   - *WC_installdir*/samples/dynacache/invalidation
3. Copy the invalidation policies from the sample invalidation file to the `cachespec.xml` file for your store. You can place the invalidation policies at the end of your store's `cachespec.xml` file after the last element.
4. Ensure the invalidation IDs match the corresponding dependency IDs in the caching policies. If a matching dependency IDs does not exist, then the invalidation policies will not be executed and you should change either the ID of the invalidation rule or the ID of the dependency-id rule so that they match.

   Note: Your store may have additional or different business requirements that require you to add additional invalidation policies and dependency IDs.
5. If necessary, change the name and directory of JSP files in the sections copied from the sample invalidation files to match the information in the rest of your store's `cachespec.xml` file.
6. Save the file.

Implementing caching for store pages that use store relationships

Business If your store is using data defined in another store through a store relationship, you must use the request attributes specified by the cache filter to define the relationships. The cache filter is a servlet filter that defines request attributes from the session and store relationship information that can be used by the WebSphere Application Server DynaCache. DynaCache then uses this information to construct cache IDs and dependency IDs to be used for cache invalidation. For a list of the request attributes set up for session information, see the *WebSphere Commerce Administration Guide*, "Dynamic caching" chapter.

The cache filter creates the store relationships information by calling the getStorePath() and getStoresForRelatedStore() methods from the StoreAccessBean. The corresponding information is listed in the following table:

Table 3.

Store Relationship Type	**Store Relationship Identifier**	**Request Attributes Name for getStorePath()**	**Request Attributes Name for getStoresFor RelatedStore()**
IBM commerce businessPolicy	-1	DC_busN	DC_bus_RS_N
IBM commerce business campaigns	-3	DC_campN	DC_camp_RS_N
IBM commerce business catalog	-4	DC_catN	DC_cat_RS_N
IBM commerce business command	-5	DC_cmdN	DC_cmd_RS_N
IBM commerce hosted store	-6	DC_hostN	DC_host_RS_N
IBM commerce price	-7	DC_prcN	DC_prc_RS_N
IBM commerce referral	-8	DC_refN	DC_ref_RS_N
IBM commerce segmentation	-9	DC_segN	DC_seg_RS_N
IBM commerce URL	-10	DC_urlN	DC_url_RS_N
IBM commerce view	-11	DC_viewN	DC_view_RS_N
IBM commerce inventory	-13	DC_invN	DC_inv_RS_N
IBM commerce base item	-14	DC_baseItemN	DC_baseItem_RS_N
IBM commerce channel store	-15	DC_chsN	DC_chs_RS_N
IBM commerce currency conversion	-17	DC_currConvN	DC_currConv_RS_N
IBM commerce currency format	-18	DC_currFmtN	DC_currFmt_RS_N
IBM commerce supported currency	-19	DC_supCurrN	DC_supCurr_RS_N
IBM commerce counter value currency	-20	DC_cterCurrN	DC_cterCurr_RS_N
IBM commerce measurement format	-21	DC_meaFmtN	DC_meaFmt_RS_X

Note: The cache filter sets up multiple request attributes when multiple store IDs are returned as DynaCache does not support an array of request attributes.

For example, if getStorePath() returns an array [10051, 10002] for the resource id -4 (IBM commerce business catalog), then the request attributes set up will be

- DC_cat0 is 10051
- DC_cat1 is 10002

Store relationship caching example

Business To understand how caching pages that use a store relationship works, consider the following example.

Publishing the sample composite store archive Business DemandChain.sar and then creating a hosted store (for example, ResellerOne) in that site creates the following stores.

Table 4.

Store ID	Directory	Store Type
10001	CommercePlaza	channel hub
10002	CommercePlazaCatalog	catalog asset store
10003	CommercePlaza	distributor proxy
10004	ConsumerDirectResellerProfile	hosted storefront asset store
10051	ResellerOne	reseller hosted store

ResellerOne (10051), the reseller hosted store, uses the assets defined in the hosted storefront asset store (10004) and the catalog asset store (1002).

In order to set up the caching relationship, the cache filter gets the following information:

Table 5.

Store ID	Relationship Type	getStorePath()	getStoresFor RelatedStore()
10001	-1 (business policy) -4 (catalog) -7 (price) -17 (currency format) -19 (currency supported)	10002	not applicable
10001	-6 (hosted store)	10051	not applicable
10051	-1 (business policy) -14 (base item)	10051, 10002, 10004	10051
10051	-3 (campaigns) -5 (command) -10 (URL) -11 (view)	10051, 10004	10051

Table 5. (continued)

10051	-4 (catalog) -7 (price) -17 (currency conversion) -18 (currency format) -19 (currency supported) -20 (counter value currency) -21 (measurement format)	10051, 10002	10051

Then the cache filter sets up the following request attributes:

Table 6.

Store Relationship	Store ID 10051	store ID 10051	store ID 10001

-1 (business policy)	DC_bus0=10051
	DC_bus1=10002
	DC_bus2=10004
		DC_bus_RS_0=10051	DC_bus0=10002

-2 (tax)	DC_tax0=10051
	DC_tax1=10004
		DC_tax_RS_0=10051	

-4 (catalog)	DC_cat0=10051
	DC_cat1=10002
		DC_cat_RS_0=10051	DC_cat0=10002

-6 (hosted store)	DC_host0=10051	DC_host_RS_0=10001	DC_host0=10051

Whenever the catalog of the catalog asset store (10002) is changed, the catalog pages of the ResellerOne store (10051) must also be invalidated before it can use the information from the catalog asset store (10002). In order for the pages in 10051 to be invalidated, extra dependency IDs must be set up for this store relationship. Setting up the extra dependency IDs for StoreCatalogDisplay is illustrated in the following example:

```
<!-- Start Store Relationship Dependency Ids -->
<!-- DC_cat1 is the catalog Profile Store ID -->
<dependency-id>storeId
<component id="DC_cat1" type="attribute">
<required>true</required>
</component>
</dependency-id>

<dependency-id>storeId:catalogId
<component id="DC_cat1" type="attribute">
<required>true</required>
</component>
<component id="catalogId" type="attribute">
<required>true</required>
</component>
</dependency-id>

<dependency-id>StoreCatalogDisplay:storeId
<component id="DC_cat1" type="attribute">
<required>true</required>
</component>
</dependency-id>
<!-- Ends Store Relationship Dependency Ids -->
```

The extra dependency IDs created are as follows:

- storeId:10002
- storeId:catalogId:10002:10051
- StoreCatalogDisplay:storeId:10002

Once these extra dependency IDs are defined, whenever there changes to the catalog asset store 10002 that cause the catalog asset store pages to be invalidated, the hosted store (10051) pages will also be invalidated.

Replacing the cache command functions with dynamic caching

Previous versions of WebSphere Commerce used the CacheCommand (com.ibm.commerce.cache.commands.CacheCommandImpl) to implement more advanced caching configurations, for example, caching pages by the user's state and type determined from a customer profile.

In Version 5.5, using dynamic caching you can cache the servlet or JSP file result as you would using the cache command, by adding the cache command logic to a JSP file.

Consider the following example:

The StoreCatalogDisplay command can display different headers based on the user's state and type attributes. To cache the header JSP file, create a new JSP file, `CacheParametersSetup.jsp` that includes the user's state and type attributes. For example:

```
<%@ page import="com.ibm.commerce.command.CommandContext" %>
<%
    String userState = null;
    String userType = null;

    CommandContext cmdcontext = (CommandContext) request.getAttribute
(ECConstants.EC_COMMANDCONTEXT);
    if (cmdContext != null) {
        userState = cmdcontext.getUser().getState();
        userType = cmdcontext.getUser().getRegisterType();
    }
%>
```

Then the `StoreCatalogDisplay.jsp` statically includes the `CacheParametersSetup.jsp` and dynamically includes the CachedHeaderDisplay.jsp using the userState and userType as input parameters:

```
<%@ include file="CacheParametersSetup.jsp"%>

<jsp:include page="CachedHeaderDisplay.jsp" flush="true">
  <jsp:param name="storeId" value="<%= storeId %>" />
  <jsp:param name="catalogId" value="<%= catalogId %>" />
  <jsp:param name="langId" value="<%= languageId %>" />
  <jsp:param name="userState" value="<%= userState %>" />
  <jsp:param name="userType" value="<%= userType %>" />
 </jsp:include>
```

The `CachedHeaderDisplay.jsp` file contains the logic to display different information based on the input parameters.

```
<%
    if (userType.equals("G")) {
%>
        <table cellpadding="0" cellspacing="0" border="0" width="100%" height="28">
        . . .
        </table>
<%
```

```
    }
    else {
%>
        <table cellpadding="0" cellspacing="0" border="0" width="100%" height="28">
        . . .
        </table>
<%
    }
%>
```

In order to complete caching, the input parameters must be identified by a cache-ID rule.

```
<cache-entry>
        <class>servlet</class>
        <name>.../CachedHeaderDisplay.jsp</name>
        <property name="save-attributes">false</property>

        <cache-id>
            <component       id="storeId" type="parameter">
                <required>true/required>
            </component>
            <component       id="catalogId" type="parameter">
                <required>true</required>
            </component>
            <component       id="userState" type="parameter">
                <required>true</required>
            </component>
            <component       id="userType" type="parameter">
                <required>true</required>
            </component>
       < /cache-id>
        . . .
    </cache-entry>
```

Part 5. Store data overview

Chapter 10. 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).

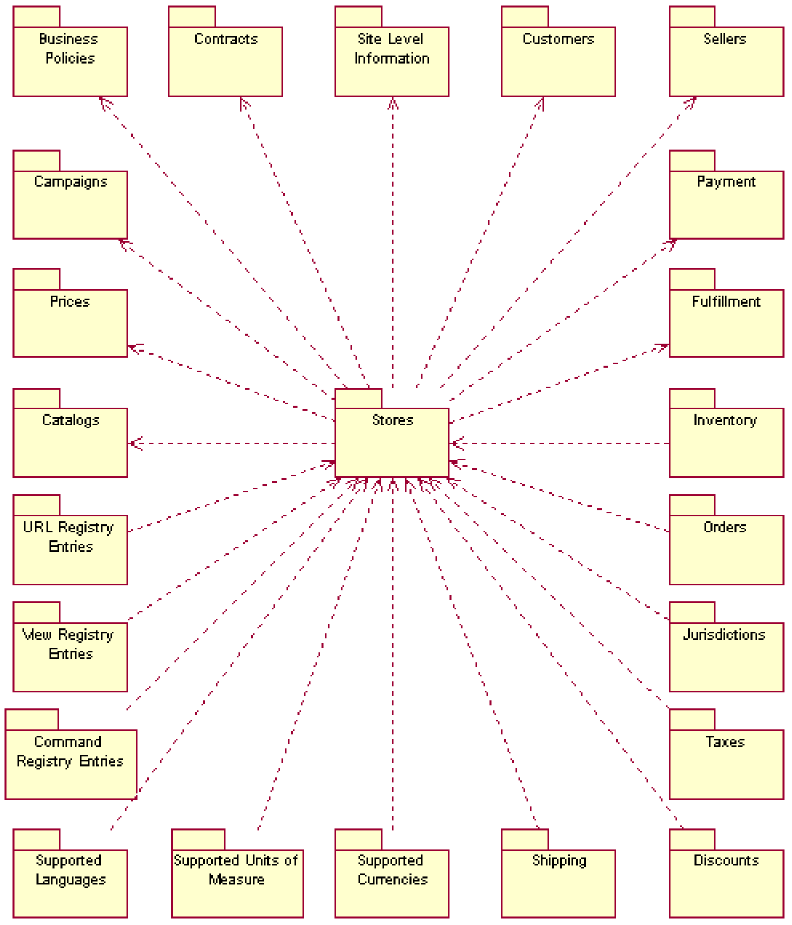
The store data information model

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 models. The information model highlights the most important features of the data 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 model that contain entity-relationship data (such as foreign key pairs) are not represented 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 differs from the data model in that in the data model each entity represents a table while in the information model any of the objects depicted may be mapped to the same database table, or a single object may map to several database tables. 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). Finally, unlike the data model, the information model may also illustrate concepts of inheritance. For more information on entity-relationship data and detail extensions, see the data model in the WebSphere Commerce Production and Development online help.

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

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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

Each of the data assets illustrated in the above diagrams is discussed in more detail in the chapters in Part 6, "Developing your store data," on page 107

Note:

In the UML notation, a dotted line with an arrow extending from an object and pointing to another object indicates that the first object has a dependency on the second object. In this diagram, the objects shown are referred to as packages. Notice that data in some packages, such as lists of Supported Currencies, are specific to a particular Store, and thus that package is shown as dependent on the Store package. Other packages, such as Catalogs, are not specific to any particular Store, but rather each Store may use Catalogs, and thus the Store object is shown as dependent on the Catalogs package. As a result, the lists of Supported Currencies form *part of* a Store, while a Store *uses* Catalogs.

Business One *part of* a Store that is of particular interest is its Store Relationships with other Stores. Each store relationship indicates that a Store depends on another Store to provide some service or information. Relationships can be defined to facilitate the use of one Store's data, such as its list of Supported Currencies, by another Store. In this scenario, the first Store acts as a provider, or container, of data which is used by the second, client, Store. As more client Stores are created, they can also define relationships that indicate they obtain certain data from certain other Stores. In this way Store relationships facilitate data sharing; the data can be created and maintained once by the provider Store, and used by several client Stores. For more information on store relationships, see Chapter 14, "Relationships between stores," on page 129.

The data in the information model can be categorized in the following ways:
- by subsystem
- by data type

Store data information model viewed by subsystem

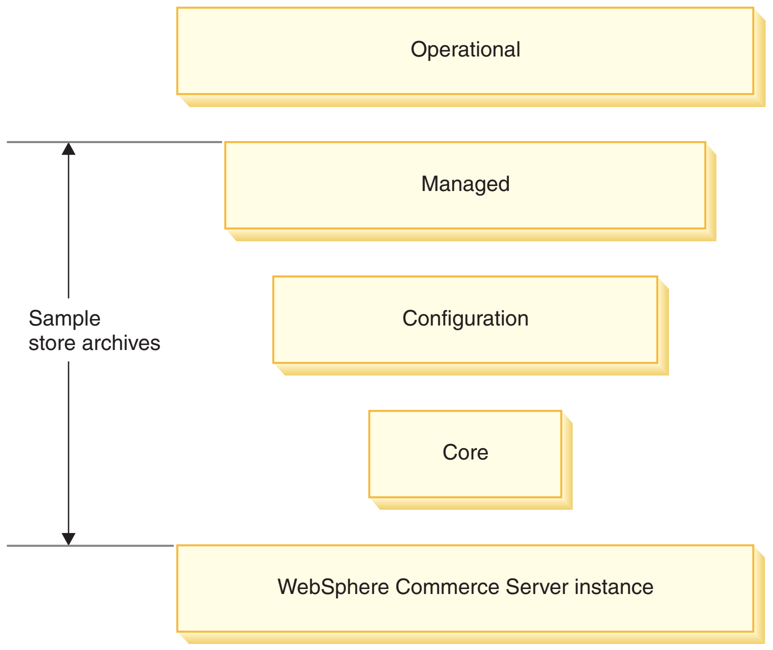
Each of the data assets in the store data information model can be grouped into the following functional areas:

Table 7.

Merchandising	**Marketing**	**Trading**	**Order Management**	**Catalog**	**Member**	**Run-time**
Discounts	Campaigns	Contracts	Shipping	Catalogs	Organizations	Organizations
Vendors	Customer profiles	Accounts	Taxes	Prices	Groups	URL, commands, and view registry
Auctions	E-mail activity	Business RFQs	Jurisdictions		Users	Supported Languages
	Coupons		Orders			Supported Units of measure
			Inventory			Supported Currencies
			Fulfillment			Site
			Payment			Store
						Store relationships
						Business policies

Store data information model viewed by data type

Data in WebSphere Commerce stores conforms to the types depicted in the following diagram. Each of the store data assets illustrated in the diagram in "Store data information model viewed by subsystem," 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, roles and addresses
- The default administrative ID, WCSADMIN
- The default commands, views and URLs
- The default business policies
- The default access groups and access control policies
- The languages and currencies supported by the instance
- The default quantity units and quantity unit conversions
- The default scheduled jobs and statecodes
- The default terms and conditions
- The default organization, which can be used as the store owner
- The default site organization
- The default store group
- The default information for staging

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 information model viewed by subsystem" on page 99.

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. Core data is divided into two levels:

- Organization
- Store

The organization core data creates the minimum data for a business model specific environment, including:

- The organization structure.
- Predefined user roles.
- Necessary access control policies.

Organization core data is available in both the sample composite store archives, and the sample organization structure component store archives.

The core data creates the minimum data for a store within that environment, 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.

Store core data is available in both the sample composite store archives and the sample component store archives.

If you published any of the sample store archives indicated above using the publish utility in the Administration Console, this information was created for you. The publish utility 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 Organization Administration Console. If you did not publish a sample composite store archive to use as the basis of your store, you will have to load this information into the database using the Loader package, or edit the database directly. For more information on using the Loader package, see Chapter 37, "Overview of loading store data," on page 335.

The Stores data in the diagram in "Store data information model viewed by subsystem" on page 99 is core data.

Configuration data

Configuration data controls the commerce server runtime. The commerce server runtime provides a framework in which the commerce applications are deployed and executed. The framework consists of command execution, exception handling, transaction control, data access, and persistence. The commerce server runtime 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 information model viewed by subsystem" on page 99 are classified as Configuration data:

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

Managed data

Managed data is 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 information model viewed by subsystem" on page 99 that are classified as managed data:

- Business policies
- Campaigns
- Catalogs
- Contracts
- Coupons
- Currencies
- Customer profiles
- Discounts
- E-mail activity
- Fulfillment centers
- Inventory (configuration information for catalog items)
- Jurisdictions
- Languages
- Members
- Payment
- Prices
- Sellers
- Shipping
- Tax
- Units of measure
- Vendors

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, this data is not managed using a content management system.

The following data assets identified in the diagram in "Store data information model viewed by subsystem" on page 99 are classified as operational data:

- Auctions
- Contracts
- Customers
- E-mail activity
- Fulfillment
- Inventory (receipts, expected receipts, inventory allocation)
- Orders
- Business Request for Quotes (RFQ)

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 value chain 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. Another example is e-mail activity. E-mail activity information and templates are considered managed data, but the actual e-mail activities generated from the templates and sent to customers are considered operation data, as are any as are any of the events resulting from the mailing, such as a customer opening the e-mail, or clicking on any of the clickable contents of the e-mail.

Store data types and the sample businesses

The sample businesses 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 the publish utility in the Administration Console, 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 10, “Publishing your store,” on page 319.

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. The Administration Console also allows you to identify which notification and messaging types will be available in your store. The Administration Console contains the publish utility, which allows you to publish sample business and stores.

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. For more information, see the “Tool and store data summary chart.”

Organizational Administration Console

The Organizational Administration Console allows you to create and manage the organizations that access your site or store. The Organizational Administration Console also allows the buyer 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 124.	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 137.	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. However, selected customer data may be loaded using the Loader package.
Administration Console	When you publish a store archive using the Administration Console, the core data is created for you. For more information on using publish, see the WebSphere Commerce Production online help.	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, the store creation wizard, or publish a store archive to create a default contract in the database). • Jurisdictions • Taxes • Shipping • Currency • Languages	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 (receipts and expected receipts).

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 add or change product information. For a different view of your merchandise and services, create a sales catalog by changing the XML source. Use the Loader packageor publish a store archive to create a master catalog in the database.) • Prices	Not applicable.
Organizational Administration Console	Use the Organizational Administration Console to create and manage organizations.	Not applicable.	Not applicable.	Customers and buyers are created when they enter the store. However, with the Organizational Administration Console, you can also manage users and approve buyers, or create new ones.

Part 6. 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:

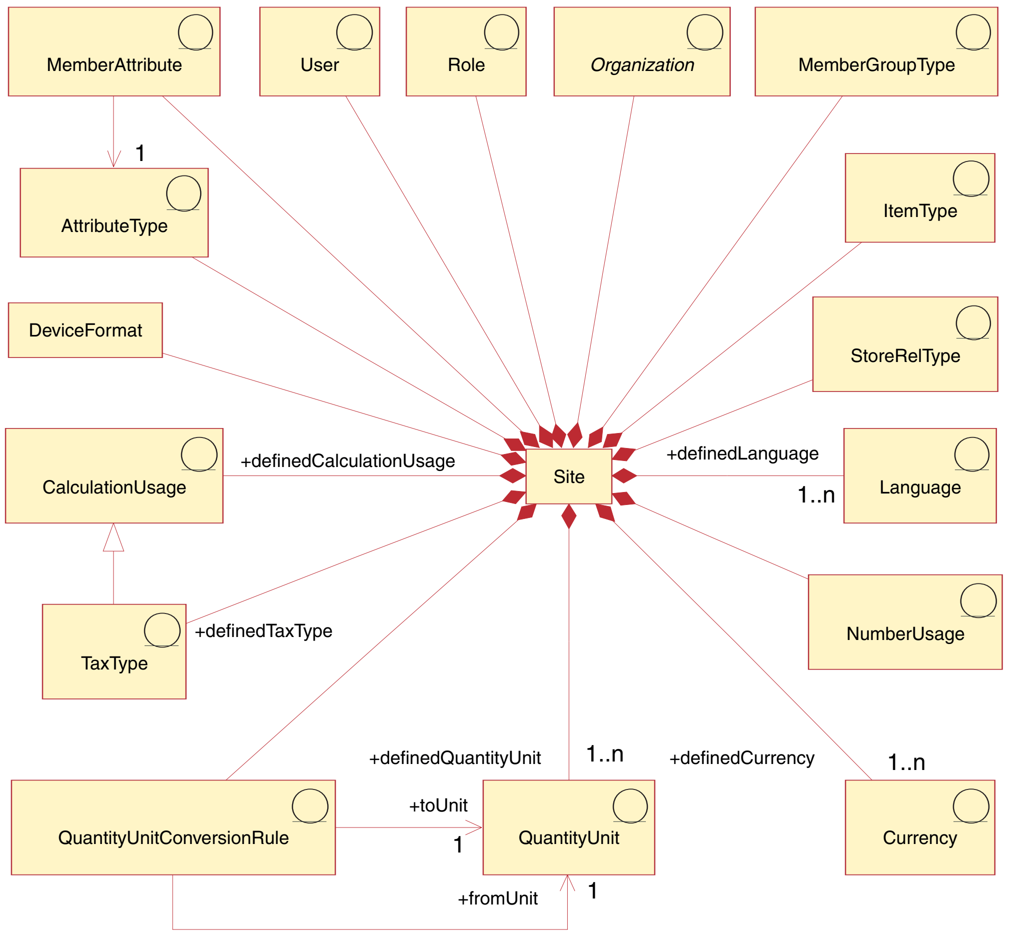
- WebSphere Commerce Server instance
  - Site
- Core data
  - Organization
  - Store
  - Relationships between stores
- Configuration data
  - Command registry
  - View Rregistry
  - URL registry
- Managed data
  - Catalog
  - Prices
  - Contracts (including Business Policies)
  - Fulfillment
  - Campaigns
  - Payment
  - Supported languages
  - Supported currencies
  - Supported units of measure
  - Jurisdictions
  - Shipping
  - Taxation
  - Discounts
- Operational data
  - Inventory
  - Orders
  - Customers
  - Auctions
  - Business RFQ

Chapter 11. 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. Some tables contain information that represents WebSphere Commerce site level capabilities available for use by all stores in the instance. All of this information is 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 Production and Development online help. For more information on store-specific asset information see Chapter 13, "Store assets," on page 123.

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 437. 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 97.

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 are 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 B2B direct buying organization are recorded as being placed on behalf of the buying organization. Stores, catalogs, and fulfillment centers are owned by organizations that are 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.

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.

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 29, “Inventory assets,” on page 265.

Device formats

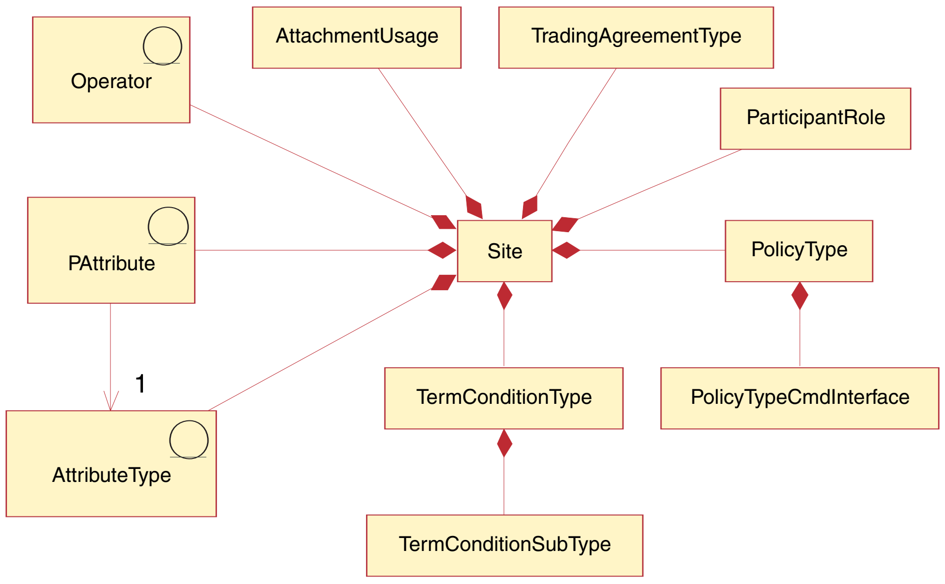
Device formats are stored in DEVICEFMT table and represent the many device formats a site uses such as browsers, I_MODE, e-mail, XMLMQ, and XMLHTTP. All these device types allow users to interact with the site through various media.

Store relationship types

Business A *store relationship type* (StoreRelType) defines the type of relationship between two stores. Each type of store relationship defines its own relationship, that is, what roles each partner in the relationship will play and what the relationship between the two is. A store relationship type is defined in the STRELTYP table, and described in the STRELTYPDS table.

Site level trading agreement data

The following diagram illustrates the types of trading agreement 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 437. 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 97.

Trading agreement type

WebSphere Commerce provides a number of trading mechanisms governing the interactions between buyers and sellers. 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. WebSphere Commerce supports several *trading agreement types*, including account, contract, RFQs, exchange, and auctions. The trading agreement types are defined in the TRDTYPE table. For more information on trading agreements, see Chapter 18, “Contract assets,” on page 179.

Participant role

Participants in trading agreements take on specific roles within each trading agreement. WebSphere Commerce supports several *participant roles*, including creator, seller, buyer, supplier, approver, administrator, distributor, service provider, reseller, host and recipient. Participant roles are defined in the PARTROLE table.

Policy type

WebSphere Commerce supports several types of business policies, including price, product set, shipping mode, shipping charge, payment and several others. Policy types are defined in the POLICYTYPE table. For more information on business policies, see Chapter 18, “Contract assets,” on page 179.

Policy type command interface

The *policy type command interface* is the Java command interface for the business policy object. The command for each policy instance must implement this interface. There can be zero or more commands for each business policy object.

Terms and conditions type

Terms and conditions define the behavior and properties of a trading agreement. WebSphere Commerce supports several *terms and conditions types*, including pricing, payment, and shipping. Terms and conditions types are defined in the TCTYPE table. For more information on terms and conditions, see Chapter 18, “Contract assets,” on page 179.

Terms and conditions sub type

Each terms and conditions type can contain several *terms and conditions sub types*. Terms and conditions sub types are defined in the TCSUBTYPE.

Personalization attribute

The *personalization attribute* allows you to create attributes for products. The personalization attribute is defined in the PATTRIBUTE table. Each personalization attribute has one and only one attribute type.

Attribute type

The *attribute type* defines the type of the attribute. Attribute types are defined in the ATTRTYPE table.

Operator

The *operators* used in the site include simple operator (allows a single value), compound operator (range - continuous), and compound operator (set). Operators are defined in the OPERATOR table.

Attachment usage

An attachment is a supporting document for a trading document. For example, it can be a specification of a product, or a price list spreadsheet. *Attachment usage* describe how and where attachments will be used. Attachment usage is defined in the ATTACHUSG table.

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 *WebSphere Commerce Installation Guide*, "Creating a WebSphere Commerce instance."

Chapter 12. Member assets

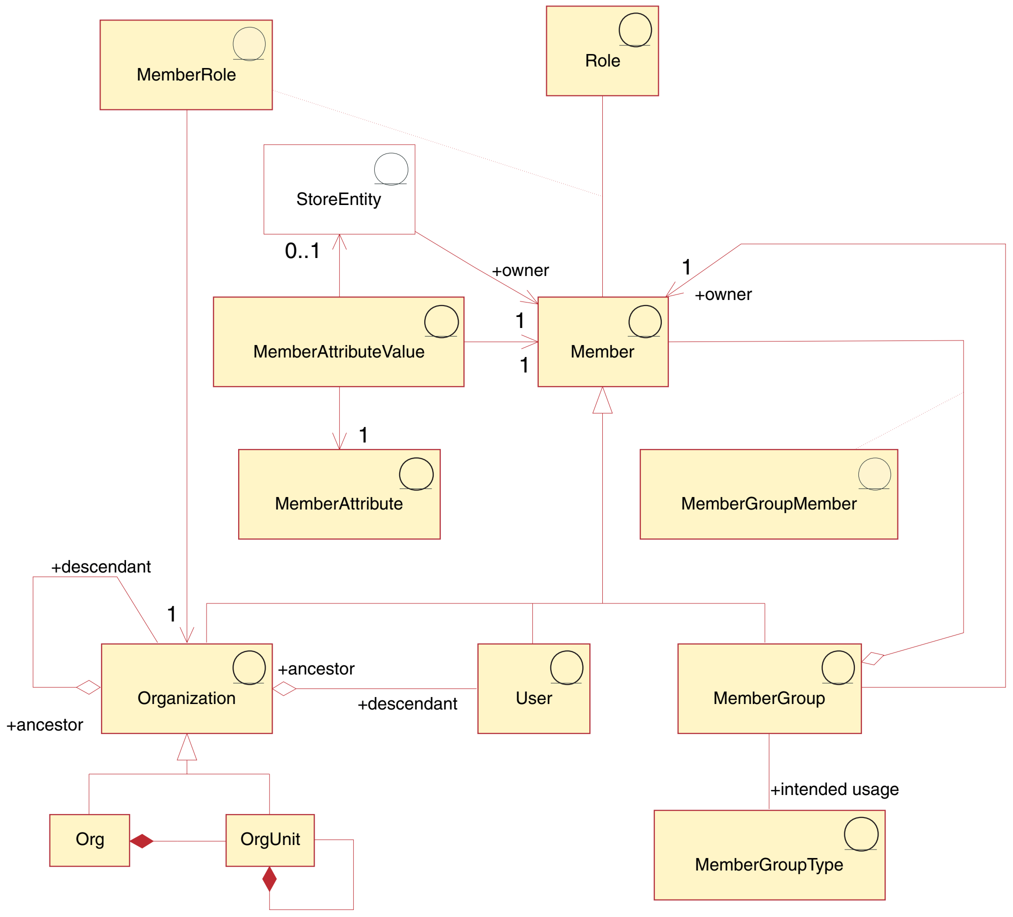
This chapter first explains the WebSphere Commerce Member subsystem, then describes the three types of members that are relevant to store developers: customers, Sellers, and administrators. Note that WebSphere Commerce provides a Member subsystem, which includes members or users, and organizations.

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 development 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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

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 Business B2B direct customer) and type of 'C' denotes a retail user (or a consumer direct customer). For more information about registered and non-registered users, refer to "Members" in the WebSphere Commerce development 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 the 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 supported for users and organizational entities, but not member groups.

Roles

Each user can perform one or more roles in an organization. A Site Administrator assigns a role or roles to each 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:
- Business relationship roles
- Customer service roles
- Marketing roles
- Operational roles
- Organizational management roles
- Product management and merchandising roles
- Technical operations roles

For details about each of these roles, refer to the WebSphere Commerce development 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 an organizational entity. This can be any organizational entity; it does not have to be one of the user's ancestors. However, since roles are inherited, the user will play the assigned role in any descendant of the organization for which the role is assigned. For example, if a user is given a role in the Root Organization, then the user will play that role for all organizational entities.

WebSphere Commerce roles can be assigned manually through the Organization Administration Console, and automatically through the registration and session management commands. This automated role assignment is based on the configuration specified in the MemberRegistrationAttributes.xml file. WebSphere Commerce 5.5 provides the MemberRegistrationAttributes.xml file, which can be modified to suit particular registration requirements. For more information on automated role assignment, and the MemberRegistrationAttributes.xml file, refer to the WebSphere Commerce development online help topic ″MemberRegistrationAttributes XML and DTD files″.

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

Understanding customer assets in WebSphere Commerce

A *customer* is a user within WebSphere Commerce. A customer can browse the store's online catalog, places an order, create an interest list, set up addresses (such as for general contact, billing, and shipping purposes), and purchase from the store or the Seller. A customer is also a user. The following diagram illustrates the assets that a customer requires to place an order from a store.

As shown in the preceding diagram, the WebSphere Commerce system contains members. Each user 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 116 for more details.

In this case, the user 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:

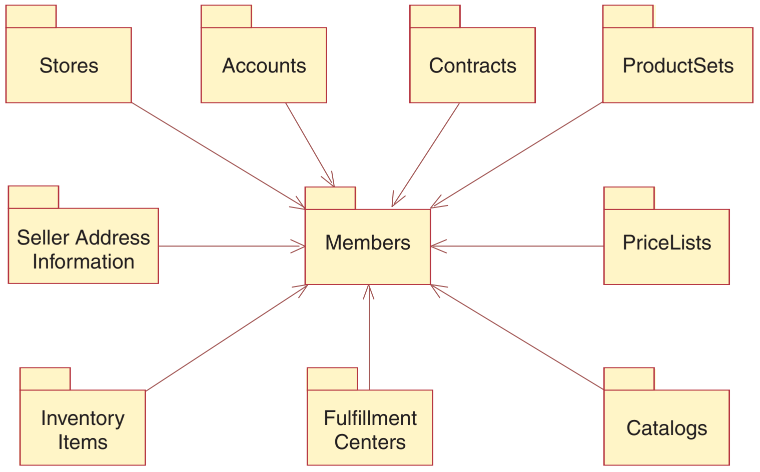
- A contact address is used to notify the customer for various purposes, such as regarding the status or changes to an order, and notices about upcoming store events (such as promotions or store maintenance). The customer's contact address includes the street name and number, city, state or province, 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* is a user within WebSphere Commerce. The 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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

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 Receiver at a warehouse. The Seller role can maintain the following assets in order to sell to customers:

- Stores
- Business 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 for 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 13, “Store assets,” on page 123 and Part 6, “Developing your store data,” on page 107.

Accounts

Business

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 “Accounts (business accounts)” on page 180 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.

The Buyer in a contract can be a user, an organization, or a member group. In the case of the user, the Buyer is considered the customer. In the case of an organization, which is defined as a Buyer in a contract, then any child of this organization can act as a Buyer for the contract. In the case of a member group, any user in the member group can act as a Buyer for the contract.

For more information about WebSphere Commerce contracts and the default contract a Seller can use, refer to “Contracts” on page 181.

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 144.

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 17, “Pricing assets,” on page 171.

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 16, “Catalog assets,” on page 141.

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 19, “Fulfillment assets,” on page 197 and Chapter 26, “Shipping assets,” on page 229.

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 development online help and Chapter 29, "Inventory assets," on page 265.

Understanding administrator assets in WebSphere Commerce

Administrators are simply users or members with assigned roles that allow them to perform certain administrative activities. Refer to "Understanding member assets in WebSphere Commerce" on page 115 for more details on the assets which can be associated with an administrator.

Creating member assets in WebSphere Commerce

To create a Seller (an organization that acts as the store owner) and to maintain information about the Seller, use the WebSphere Commerce Administration Console. For more information, see the WebSphere Commerce development online help topic ″Creating an organization″.

To create an administrator, use the WebSphere Commerce Administration Console to create the user, then assign the desired roles to this user. For more information, see the WebSphere Commerce development online help topics ″Creating a user″ and ″Assigning roles by user distinguish name″.

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.

The sample stores provided with WebSphere Commerce each contain their own versions of the MemberRegistrationAttributes.xml file, which is used for configuring the automated role assignment for registration and session management commands. If you choose to modify the organization structure, or have particular requirements on role assignment, then you will have to modify this file. Refer to the WebSphere Commerce development online help topic ″MemberRegistrationAttributes XML and DTD files″ for more information on this file and how to configure it to suit your needs.

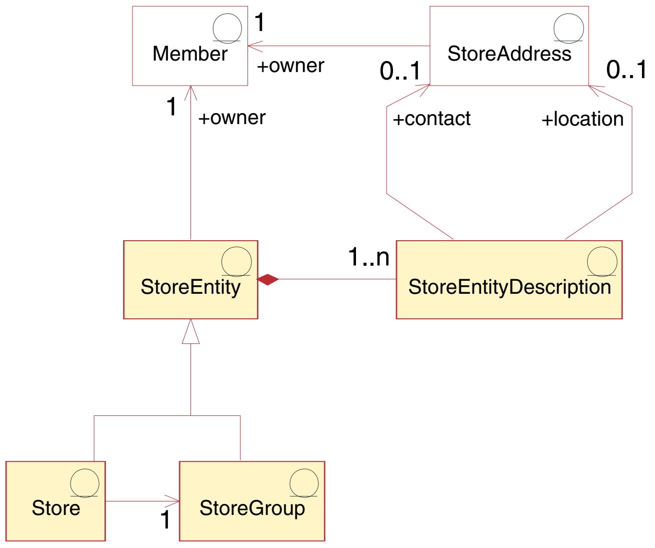
Chapter 13. Store assets

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 97. For more information on the conventions used in this diagram, see Appendix A, “UML legend,” on page 437.

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 “Understanding member assets in WebSphere Commerce” on page 115.

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 tools in WebSphere Commerce Accelerator 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

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.
- 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 creating store assets in the form of an XML file, see “Creating store data assets in an XML file.” For more information on editing the store using theWebSphere Commerce Accelerator, see the WebSphere Commerce Production online help.

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 globalized 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 10, “Publishing your store,” on page 319.

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 information in Chapter 38, "Loading WebSphere Commerce database asset groups," on page 383.
2. 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:
   - *WC_installdir*/samplestores

   Note: The *WebSphere Commerce Sample Store Guide* contains information about each of the data assets contained in the sample stores.

   Each sample store includes several `store.xml` files, which include the store information by language. Since the sample stores are translated into multiple languages, there will be multiple `store.xml` files in each store. 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.
3. Review the information in Appendix B, "Creating your data," on page 439.
4. 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 wcs.dtd file. The DTD file is located in the following directory:
   - *WC_installdir*/schema/xml
5. 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.
6. 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 globalized store, 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.

7. 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 globalized store, 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.

8. 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 directory is located under the document root of the Store.war Web module.
- 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 = B2B direct
  - B2C = Business-to-Consumer (consumer direct)
  - CHS = Reseller Hub (Commerce Plaza)
  - CPS = Master Catalog Profile Store (catalog asset store)
  - RHS = Reseller Hosted Store
  - RPS = Reseller Profile Store (reseller storefront asset store)
  - DPS = Distributor Profile Store (distributor asset store)
  - DPX = Distributor Proxy Store
  - HCP = Commerce Hosting Hub (hosting hub)

– PBS = Store Directory
– MPS = Merchant Profile Store (hosting storefront asset store)
– MHS = Merchant Hosted Store
– SCP = Supplier Hub
– SPS = Supplier Profile Store (supplier asset store)
– SHS = Supplier Hosted Store

Note: The names denoted in brackets are the names of the corresponding samples provided with WebSphere Commerce.

9. 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.

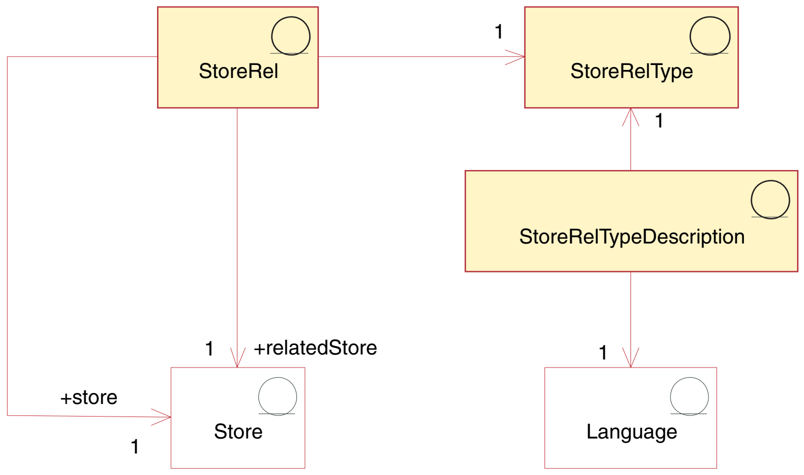
For more information about the use of @ and **&** see Appendix B, "Creating your data," on page 439.

Chapter 14. Relationships between stores

Business WebSphere Commerce supports several types of relationships between stores in a site. For example, one store may provide hosting services for another store, or a store may use the catalog or currency assets provided by another store.

Understanding relationships between stores in WebSphere Commerce

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



Store relationships

A *store relationship* (captured in the StoreRel table) is the relationship between two stores. All store relationships are directional, that is in each store relationship one store provides the services and the second store in the relationship uses those services. For example, store A uses the catalogs provided by store B.

Each store relationship has one store relationship type (StoreRelType).

Store relationship types

A *store relationship type* (StoreRelType) defines the type of relationship between two stores. Each type of store relationship defines its own relationship, that is, what roles each partner in the relationship will play and what the relationship between the two is.

Store relationship types supported by WebSphere Commerce

WebSphere Commerce supports several relationship types between stores. The default relationship types provided by WebSphere Commerce can be loosely grouped into two categories:

- Relationships in which one store provides data assets to another store. For example, store A provides the catalog data that is used in store B.
- Relationships in which one store has a ″business relationship″ with another store, that is a store may host another store, or a store may transfer a shopping cart to another store.

Relationships in which one store provides data assets to another store:

Table 8.

Relationship Type	Description	For more information, see
com.ibm.commerce. businessPolicy	One store uses business policies defined in another store.	Chapter 18, "Contract assets," on page 179
com.ibm.commerce. campaigns	One store uses campaigns defined in another store.	Chapter 20, "Campaign assets," on page 203
com.ibm.commerce.catalog	One store uses catalog data defined in another store.	Chapter 16, "Catalog assets," on page 141
com.ibm.commerce.command	One store uses commands defined in another store.	Chapter 15, "Command, view, and URL registry data," on page 135
com.ibm.commerce.price	One store uses price data defined in another store.	Chapter 17, "Pricing assets," on page 171
com.ibm.commerce. segmentation	One store uses customer profile data defined in another store.	Chapter 32, "Customer profiles," on page 281
com.ibm.commerce.URL	One store uses URLs defined in another store.	Chapter 15, "Command, view, and URL registry data," on page 135
com.ibm.commerce.view	One store uses views defined in another store.	Chapter 15, "Command, view, and URL registry data," on page 135
com.ibm.commerce.storeitem	One store uses items defined in another store.	Chapter 29, "Inventory assets," on page 265
com.ibm.commerce. propertyFiles	One store uses properties files defined in another store.	
com.ibm.commerce.currency. conversion	One store uses currency conversion rates defined in another store.	Chapter 23, "Currency assets," on page 217
com.ibm.commerce.currency. supported	One store uses currencies supported in another store.	Chapter 23, "Currency assets," on page 217
com.ibm.commerce.currency. format	One store uses currency formats defined in another store.	Chapter 23, "Currency assets," on page 217
com.ibm.commerce.currency. countervalue	One store uses currency countervalues defined in another store.	Chapter 23, "Currency assets," on page 217
com.ibm.commerce. measurement.format	One store uses units of measurement defined in another store.	Chapter 24, "Units of measure assets," on page 223

One store may have relationships with multiple stores. That is, store A may want to use the catalog resources from stores B, C, and D. In order to facilitate such relationships between multiple stores, you must provide a sequence order for the stores from which a store is using assets. Sequencing in relationships between stores works in the following ways:

- Override: If the store relationship follows the override method of sequencing, the store relationship with the lowest sequence number that is the store relationship used. The following store relationships use the override method:
  - command

  - currency
  - measurement
  - price
  - property files
  - storeitem
  - URL
  - views
- Merge: If the store relationship follows the merge method of sequencing, WebSphere Commerce looks for all store relationships associated with that store, and merges the data from all of the associated stores. The following store relationships use the merge method:
  - business policies
  - campaigns
  - catalog
  - segmentation

All of the default store relationship types are designated as using either the override or merge method of sequencing.

Note: Although a store relationship type does not exist for contracts, a single contract can be deployed to multiple stores. For more information, see the WebSphere Commerce Production and Development online help.

Relationships in which one store has a ″business relationship″ with another store:

Table 9.

Relationship Type	Description
com.ibm.commerce. hostedStore	The hub store hosts the reseller, supplier or hosted stores.
com.ibm.commerce. referral	The hub store has referral relationships with distributors. The hub store may transfer a shopping cart to a distributor store. Usually the store receiving the shopping cart is a proxy store for an external system.
com.ibm.commerce. channelStore	One store acts as the hub store for another store. This relationship defines the relationship between the store directory and the Hosting hub.

Store relationship type description

A *store relationship type description* describes the type of relationship. Each store relationship type description describes only one relationship type. The store relationship type description may be available in more than one language.

Creating store relationships in WebSphere Commerce

Create your store relationships 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 10, “Publishing your store,” on page 319.

Note: If you use the Store Creation wizard to create hosted stores (for more information, see "The Store Creation wizard" on page 68) or service agreements to create distributor proxy stores (for more information, see "Creating proxy stores" on page 68) many of these store relationships are created for you.

To create store relationship assets, do the following:

1. Review the information in Chapter 38, "Loading WebSphere Commerce database asset groups," on page 383.
2. 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:

   - *WC_installdir*/samplestores

   Note: The *WebSphere Commerce Sample Store Guide* 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 439.
4. Create a `storerelation.xml` file, either by copying one of the `storerelation.xml` files in the sample store archives, or by creating a new one. For more information, see the wcs.dtd file. The DTD file is located in the following directory:
   - *WC_installdir*/schema/xml
5. Create a store relationship.
   a. Using the following example as your guide, define a store entity in your XML file for the STOREREL table.

      ```
      <storerel
       store_id="@storeent_id_1"
       relatedstore_id="@storeent_id_2"
       streltype="-4"
       sequence="0"
       state="1"
      />
      ```

      where

      - `store_id` is the primary store that uses the services of the related store.
      - `relatedstore_id` is the store that provides the service used by the primary store.
      - `streltype` is the type of relationship. The default relationship types are as follows:
        - -1 com.ibm.commerce.businessPolicy
        - -3 com.ibm.commerce.campaigns
        - -4 com.ibm.commerce.catalog
        - -5 com.ibm.commerce.command
        - -6 com.ibm.commerce.hostedStore
        - -7 com.ibm.commerce.price
        - -8 com.ibm.commerce.referral
        - -9 com.ibm.commerce.segmentation
        - -10 com.ibm.commerce.URL
        - -11 com.ibm.commerce.view
        - -13 com.ibm.commerce.inventory
        - -14 com.ibm.commerce.storeitem

  – -15 com.ibm.commerce.channelStore
  – -16 com.ibm.commerce.propertyFiles
  – -17 com.ibm.commerce.currency.conversion
  – -18 com.ibm.commerce.currency.format
  – -19 com.ibm.commerce.currency.supported
  – -20 com.ibm.commerce.currency.countervalue
  – -21 com.ibm.commerce.measurement.format
- `sequence` defines the selection sequence when more than one related store is defined for the same relationship type. Default is 0.
- `state` is the state of the relationship (0 = inactive, 1 = active). Default is 1.

For more information about the use of **@** and **&** see Appendix B, "Creating your data," on page 439.

Chapter 15. Command, view, and URL registry data

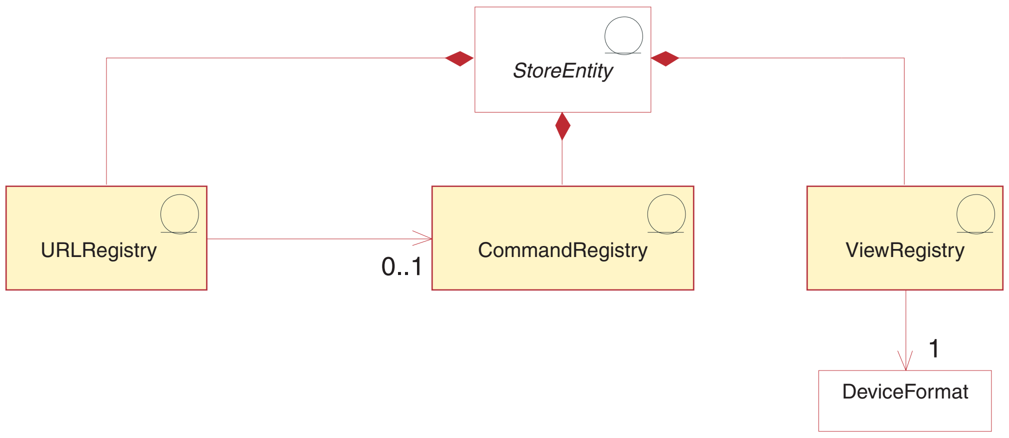
The command, view, and URL registries are part of the WebSphere Commerce command framework, which is described in more detail in *WebSphere Commerce Programming Guide and Tutorials*, chapters one, "Overview", two "Design patterns" and six "Command implementation". In order to understand how the command, view, and URL registries fit into the information model, a brief overview is provided here.

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.

Understanding command, view and URL registries in WebSphere Commerce

The WebSphere Commerce command framework determines how a command will execute and then returns a response based on the view returned by the executed command. The command execution and response is store dependent, which means that the same command can be implemented differently for each store, as well as return different responses for each store.

The following diagram illustrates the command, view, and URL registry structure in the WebSphere Commerce Server.



URL registry

The URL registry maps a command name to the actual interface of the command to be executed. Each URL registry entry is store sensitive, that is, each store can define a different interface for the same URL value. If the store version of the URL registry cannot be found, then the URL registry defined for the site (store 0) is used. By default, all URL registries are defined for the site.

Business URLs defined and registered in one store may be used by other stores. In order for one store to use URLs defined in another store a store relationship of type com.ibm.commerce.URL must be created between the stores. For more information, see Chapter 14, “Relationships between stores,” on page 129.

Command registry

Every command, whether it is a controller or task command, can be defined in the command registry. If a command is defined in the command registry, that definition will be used as the command implementation when the command is executed. If the command is not defined in the command registry, a default implementation will be used instead. Every command interface is assigned a default implementation that is used if the command is not defined the command registry.

If a command is defined in the command registry as a site level command (store 0), the site level implementation is used, except when the command is executed for a store that has defined a different implementation of the command.

The command registry allows different stores to use the same commands but to extend part or all of the implementations without changing the original flow of the command.

Business Commands defined and registered in one store may be used by other stores. In order for one store to use commands defined in another store a store relationship of type com.ibm.commerce.command must be created between the stores. For more information, see Chapter 14, “Relationships between stores,” on page 129.

View registry

After a command is executed, in most cases, the requestor of the command requires a response to be returned. When determining the response, the command framework considers the following factors:

- The view found in the response properties after the command is executed.
- The store on whose behalf the command was executed.
- The device format of the request when the request was made.

Every view that returns a response must be defined in the view registry, either per store, or by default, by site. Each store will normally define the view for each possible device format of the incoming request. However, if a view is not defined by a store, the default view for the site will be used. The adapter handling the request will decide which device format and the default device format to use when determining which view to call. There is no one generic device format, so depending on the different types of requests that can be accepted by WebSphere Commerce, there may be a view defined for each device format.

Business Views defined and registered in one store may be used by other stores. In order for one store to use views defined in another store a store relationship of type com.ibm.commerce.view must be created between the stores. For more information, see Chapter 14, “Relationships between stores,” on page 129.

Creating new commands, views, and URLs

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.

Note: When creating new views, ensure that you associate access control policies with each new view. For more information, see "Adding access control to your store" on page 289.

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

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 the publish utility in the Administration Console. For more information on the Loader package, see Part 10, "Publishing your store," on page 319.

Note: Before creating an XML file to load new or customized commands, refer to the *WebSphere Commerce Programming Guide and Tutorials* 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 information in Chapter 38, "Loading WebSphere Commerce database asset groups," on page 383.
2. 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:
   - *WC_installdir*/samplestores

   Note: The *WebSphere Commerce Sample Store Guide* 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.
3. Review the information in Appendix B, "Creating your data," on page 439.
4. 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 wcs.dtd file. The DTD files are located in the following directory:
   - *WC_installdir*/schema/xml
5. 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 439.
- `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.

6. 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 439.

- `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.

7. 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 439.
- `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 439.

Chapter 16. 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:

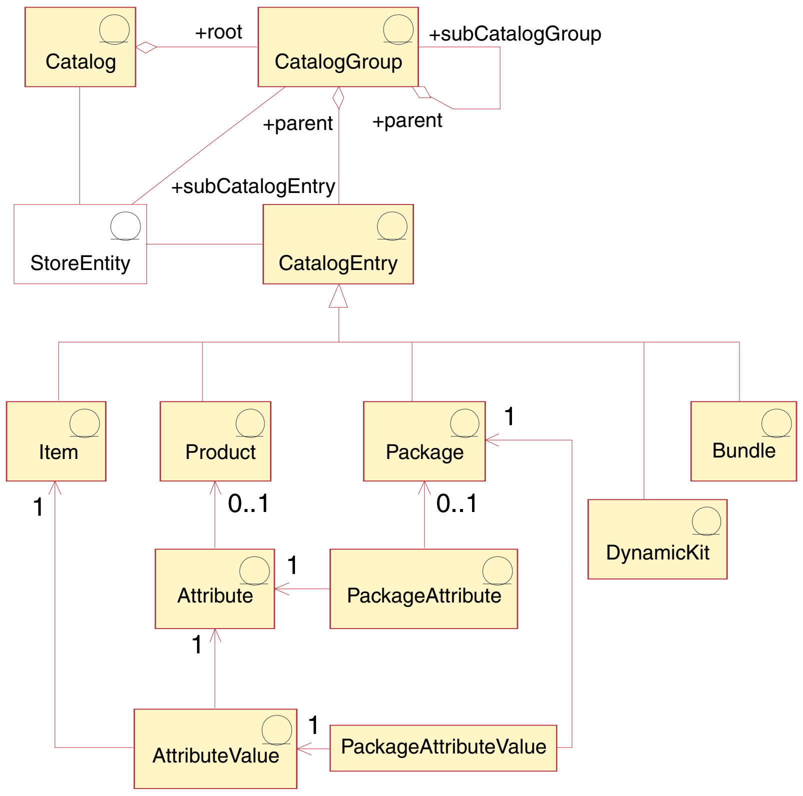
- What you are selling, including
  - Prices, which are almost always included in an online catalog.
  - Product data, such as descriptions and images of your merchandise.
  - Categories, as most, but not all catalogs divide merchandise into categories, to facilitate navigation for customers.
- A display 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*, also referred as simply a 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 *sales catalogs* for display purposes. A sales catalog can contain a subset or the same catalog entries as the master catalog, but will have a much more flexible category structure for customer display purposes. While there is only one master catalog, you can create as many sales catalogs as you want. However, since you need to use the master catalog to manage your online merchandise, we recommend that you also use the master catalog as your sales catalog to minimize maintenance overhead.

If you are creating a new master catalog for a WebSphere Commerce store, or if you are modifying an existing master catalog available from a WebSphere Commerce sample store, such as ToolTech, you will have to ensure that your catalog meets these requirements. The following diagram outlines the basic structure of a master 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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

Catalog

The *catalog* is the starting point in 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 displayed and available for purchase in 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.

Catalog groups

Catalog groups are generic groupings of your catalog entries, created for 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 general catalog groups (called root categories, or top categories), 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. As an 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, bundle, package, static kit, or dynamic kit. If necessary, you can create new catalog entry types that do not fit into one of the six 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, the terms 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.

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.

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.

Note: For WebSphere Commerce Accelerator users, packages and *prebuilt kits* are considered synonymous. When using the Product Management tools in the WebSphere Commerce Accelerator, a package is known as a prebuilt kit. In the WebSphere Commerce database schema, this particular type of catalog entry is called an package.

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 controlled by an external product configurator through a set of predefined rules and user interaction, and supplied at order entry time. 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.

Static kits

A *static kit* is a group of products that are ordered as a unit. The information about the products contained in a static kit is predefined and controlled within WebSphere Commerce. The individual components within the order cannot be modified and must be fulfilled together. A static kit will backorder if any of its components are unavailable.

A static kit is first created as a package, then configured by an administrator.

Product sets

Product sets are associated with published catalog entries. 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 and manage contracts and entitlement filtering rules on the master catalog.

Attributes

Attributes are properties of products in an online store. There are two types of attributes:

- Defining attributes are properties, such as color or size. Attribute values are the property of an attribute such as a specific color (blue or yellow) or size (medium). You must predefine attribute values before assigning them to items. Attribute values are implicitly related to their attributes. Each possible combination of attributes and attribute values equals a new item. After creating attributes and their values, you can update information such as name, description, and type (text, whole numbers, or decimal numbers). Defining attributes are used for SKU resolution, where each possible combination of attributes and attribute values defines an item.

- In contrast, descriptive attributes simply provide additional descriptions. For example, some pieces of clothing should only be dry cleaned, never washed, and a descriptive attribute can specify this dry clean only condition. Note that descriptive attributes are not used for SKU resolution and are meant to enhance product descriptions, or to provide easy customization for your business specific information.

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 defining attributes values defines an item.

Package attributes

Package attributes must be created from the attributes of the products that are contained within packages. A package containing only items will have no package attributes.

Package attribute values

Package attribute values are the values assigned to package attributes. Package attribute values must be created from the attribute values of the products that are contained within packages.

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 globalized 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, bundles, packages, static kits, 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 10, “Publishing your store,” on page 319.
3. Review the information in Appendix B, “Creating your data,” on page 439.
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 globalized 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 directory:
   - *WC_installdir*/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:
   - *WC_installdir*/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="@seller_b2b_mbr_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.
   - `identifier` 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="@seller_b2b_mbr_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 groups 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="@seller_b2b_mbr_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.
  - STKT = static 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="@seller_b2b_mbr_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="@seller_b2b_mbr_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="@seller_b2b_mbr_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 sales 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 = do 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"
usage="1"
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
- `usage` is the type of attribute:
  - 1 identifies a defining attribute used for SKU resolution.
  - 0 (or another value) identifies a descriptive 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. As an example, use the following code sample to create 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="@seller_b2b_mbr_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.
   - As an example, use the following code sample to 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 the catalog entry.
  - 1 = display the catalog entry.

• As an example, use the following code sample to 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 sales 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. As an example, use the following code sample to 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.
   - REPLACEMENT = replacement.
- `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:
   - REQUIRES
   - COMES_WITH
   - TEMP
   - NONE
- `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 store-catalog.xml file 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 globalized 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

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 248.

Part 13: Associating shipping methods to your catalog

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 231.

Part 14: Associating a fulfillment center to your catalog

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 200.

Part 15: Creating prices for your catalog entries

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 175. 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

After you have created your data, load the XML file into the database by either using the Loader package or through the publish utility. For more information on the Loader package, see Part 10, "Publishing your store," on page 319.

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

- *WC_installdir*/samplestores

The store-catalog.dtd file is located in the following directory:

- *WC_installdir*/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 store-catalog.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="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 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, static kits, 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="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 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 sales catalog

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

In particular, sales 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 task creates a sales catalog by modifying the FashionFlow sample store catalog. The resulting catalog can no longer be classified as a master catalog since the following steps introduce the categorization of some products into multiple categories. A classic sales 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 FashionFlow, you can follow these basic steps with your own master catalog, making the appropriate adjustments to match your catalog information, structure, and designs.

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 FashionFlow sample store master catalog to a sales catalog by adding a product to a second category, do the following:

1. Publish the FashionFlow store archive to create the FashionFlow sample store. FashionFlow is available in US English and one of the nine national languages shipped with WebSphere Commerce. Choose one of the `FashionFlow_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:
   - *WC_installdir*/samplestores/FashionFlow/*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 FashionFlow sample store. In this example, `@catgroup_id_18` is the **Women's Sleepwear** category.
   - `catalog_id` is the internal reference number of the catalog as defined by the FashionFlow sample store.
   - `catentry_id` is the catalog entry internal reference number as defined by the FashionFlow 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 FashionFlow 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 FashionFlow

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 FashionFlow sample store. In this example, `@catgroup_id_18` is the **Women's Sleepwear** category.
- `catalog_id` is the internal reference number of the catalog as defined by the FashionFlow sample store.
- `catentry_id` is the catalog entry internal reference number as defined by the FashionFlow 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 FashionFlow 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 FashionFlow store archive from the Administration Console or load the catalog.xml file with the Loader package as instructed in "Loading database asset groups" on page 390.

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:

- *WC_installdir*/samplestores

Note: These examples originate from the FashionFlow 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, static kit, 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="FashionFlow"
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, StaticBean, 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, you can use the Product Management tools.

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 also use the Product Management dynamic table, which allows you to update your catalog entry information directly. 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 an administrator to use this command.

Loader package

You can also maintain your catalog using the Loader package, formerly known as part of 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.

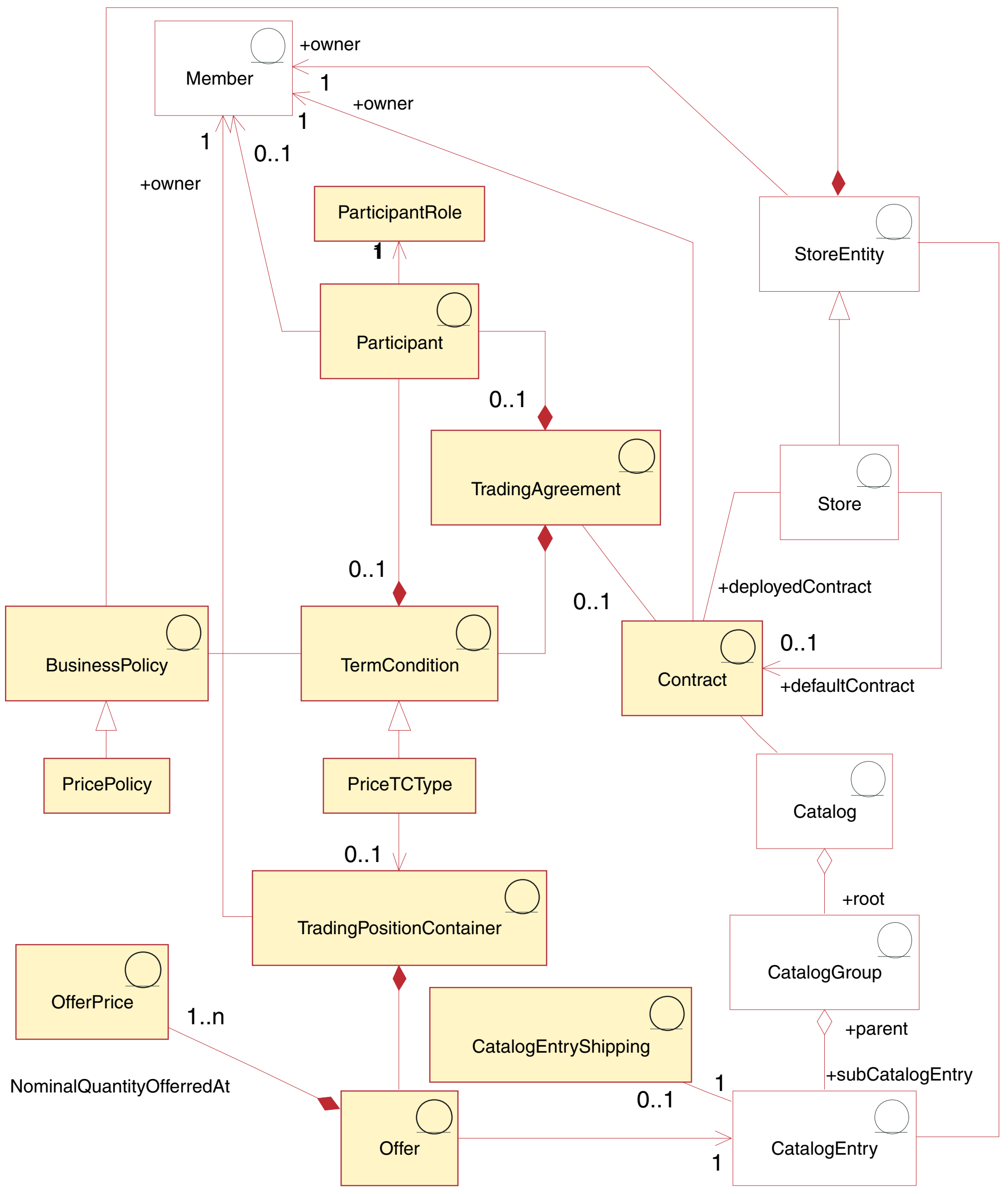
Refer to the WebSphere Commerce Production online help for more information.

Chapter 17. 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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

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

▶Professional ▶Business *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: The following pricing terms and conditions are available in WebSphere Commerce:

Customized price list
This term 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.

Entire catalog with adjustment
This 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.

Price list with adjustment
This 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.

Price list with selective adjustment
This 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. For information on the differences between the types of product sets, refer to ’Contract terms and conditions’ topic in the WebSphere Commerce online help.

Catalog with filtering
This 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. This term also offers all of the products available in a category, or a list of specify products and items, for sale with a percentage adjustment (mark-up or discount) from the base price as defined in the price list referenced by this term. This term can also state which categories, products and items are for sale or are not for sale in a contract. Category product sets will behave as a product set business policy. Item product sets will be customized products sets.

Trading agreement

Business 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 *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 the Administration Console. 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 10, "Publishing your store," on page 319.

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:

   - *WC_installdir*/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 439.
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:
   - *WC_installdir*/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="@seller_b2b_mbr_id"
   markfordelete="0"
   name="ToolTech"
   precedence="0"

   />
   ```

   where

   - `tradeposcn_id` is a generated unique key

- `@seller_b2b_mbr_id` is the owner of the trading position container. For the FashionFlow sample store, replace this with `@Member_ID;`.
- `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 162.
   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 162.

```
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 162.

```
<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 220.

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 439.

Chapter 18. Contract assets

In WebSphere Commerce, all 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 and WebSphere Commerce - Express support only the store default contract.

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

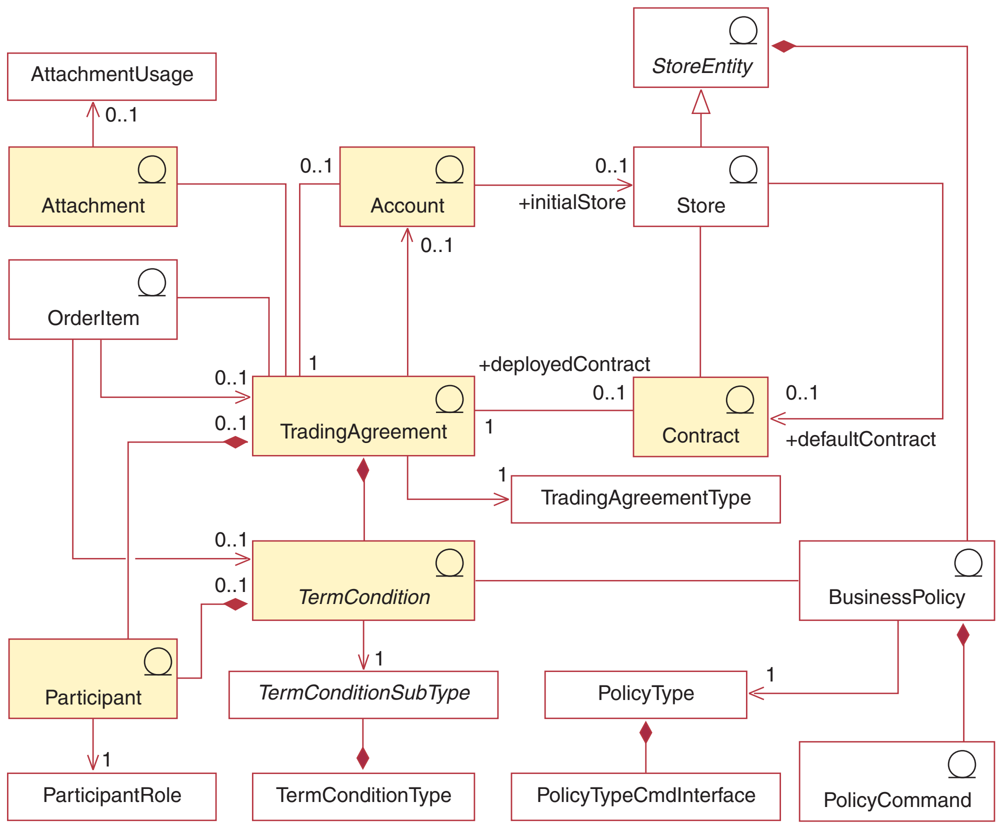
To allow all customers to shop at a store, a store created with WebSphere Commerce must include the following:

- 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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

Accounts (business accounts)

Business

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 agreements" on page 181.

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

Contracts

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 and only one default contract may be defined for a store. For more information on contracts, refer to the WebSphere Commerce Production and Development online help information. For information on creating a default contract asset, see "Creating a default contract asset in WebSphere Commerce" on page 187.

A typical contract consists of the following elements:

Profile
The contract profile contains the identifying information for the contract. This information includes a unique name for the contract, a short description, and a time period for which the contract is valid.

Participants
Contract participants are the organizations that take part in the contract. There is a buyer organization, a seller organization and contacts at both organizations.

Terms and conditions
Contract terms and conditions are the rules that cover the actual implementation of the contract. Contract terms and conditions cover such information as product pricing, returns and refunds, payment, shipping, and order approval.

Attachments
Contract attachments cover any information not covered by the previous elements such as file attachments that provide additional information about the contract and any general remarks about the contract. WebSphere Commerce stores Universal Resource Identifiers (URIs) for contract attachments, not the actual attachments.

Reference
A contract can refer to another contract to share its terms and conditions. For example, contract A can refer to contract B. Thus, a buyer who is entitled to contract A will be entitled to all the terms and conditions from contract A, as well as to all the terms and conditions in contract B.

Trading agreements

A contract is a type of 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 Business accounts
- Contracts (see restrictions discussed previously in this chapter)
- Business Request for quotes (RFQs)

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 Business 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 Business 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; Business RFQ stores RFQ-specific information; and ACCOUNT stores business account-specific information.

Participants

Contract participants take on specific roles within each contract. Participants can be a contact from a buyer organization and from a seller organization. If a contract specifies the buyer participant to be null, then all users, including guests, are entitled to the contract. Any contract may specify a null buyer participant.

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 returns are handled; how orders are approved; and from where orders are shipped.

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. WebSphere Commerce supports the following terms and conditions (terms and conditions that reference business policies are indicated with an asterisk (*)).

Fulfillment center
: This optional term allows you to specify the list of fulfillment centers from which orders placed under the contract must be filled. This list must be a subset of the fulfillment centers defined for the store. Fulfillment center precedence is defined by the store and cannot be overridden by the terms and conditions of a contract.

Order approval
: This term specifies if orders must be approved by the customer organization before filling the orders. You can specify an optional amount,

that includes taxes and shipping, that would allow orders with a value below the amount to be filled without approval from the customer organization. If an order total is over this amount, approval is required. If a buyer is placing an order with order items under multiple contracts and one item in the order has a contract specifying this term, the entire order is subject to the order approval term that applies to the item.

Payment method*

This optional term specifies the payment methods that will be accepted for orders made under the contract. The payment method could be as general as a payment type, such as a credit card type, or as specific as a credit card number to be used for payment. If no payment method term is specified in a contract, payment in all methods accepted by the store will be accepted for orders made under the contract.

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 pricing term is required in a contract. The following pricing terms and conditions are available in WebSphere Commerce:

Customized price list

This term 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.

Entire catalog with adjustment

This 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.

Price list with adjustment*

This 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.

Price list with selective adjustment*

This 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. For information on the differences between the types of product sets, see the WebSphere Commerce Development online help.

Catalog with filtering *

This 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. This term also offers all of the products available in a category, or a list of specify products and items, for sale with a percentage adjustment (mark-up or discount) from the base price as defined in the price list referenced by this term. This term can also state which categories, products and items are for sale or are not for sale in a contract. Category product sets will behave as a product set business policy. Item product sets will be customized products sets.

Product constraint terms and conditions

Product constraint terms and conditions control what products are included or excluded for sale under a contract. Product constraint terms are optional. If no product constraint terms and conditions are specified in a contract, all products specified in the contract's price terms and conditions are available for sale under the contract. The following product constraint terms and conditions are available in WebSphere Commerce:

Customized product set exclusion

This term states the items in a customized product set are not for sale in a contract.

Customized product set inclusion

This term states that items in a customized product set are for sale in a contract.

Product set exclusion*

This term states the items in a product set business policy are not for sale in a contract.

Product set inclusion*

This term states that items in a product set business policy are for sale in a contract.

Exclusion terms have precedence over inclusion terms. This means that if a product appears both an inclusion term and an exclusion term in the contract, the product could not be purchased under the contract. For information on the differences between a customized product set and product set business policy, see the WebSphere Commerce Development online help.

Returns terms and conditions

Returns terms and conditions specify how returns are handled under this contract. If no returns terms and conditions are specified then returns can not be created. If returns terms and conditions are specified they should only be one set that applies to the entire contract. The following returns terms and conditions are available in WebSphere Commerce:

Refund payment method*

This term specifies the payment method used to pay refunds to a customer. If a return charge term is specified, at least one refund payment method term must be specified as well. This term may not be specified if returns are not allowed under the contract.

Return charge*

This term specifies how returns are automatically approved and any deductions from the refund made for handling the return, for example, restocking charges.

Right to buy amount

This term places a limit on the combined value of all orders, including taxes and shipping, placed under a contract. The value of all orders made under the contract must be less or equal to a specified amount. If this limit is exceeded when placing an order, payment authorization for the order will fail.

Shipping terms and conditions

Shipping terms and conditions specify how orders will be shipped, where they will be shipped to and who will pay for the shipping. The following shipping terms and conditions are available in WebSphere Commerce:

Shipping mode*

This optional term defines how orders created under a contract are shipped. If this term is not specified in a contract, orders can be shipped by any mode available in a store. A shipping mode is also known as a shipping provider. A shipping provider is the combination of a shipping carrier and its shipping service. For example, XYZ Courier, Overnight is a shipping provider.

Ship-to address

This optional term specifies where products purchased under a contract are shipped. Specifying this term and condition allows you to limit the locations where orders can be shipped. If the ship-to address term and condition is not specified, a ship-to address must be specified each time an order is made under a contract. If this term is specified, the buyer can not specify a new ship-to address when placing an order, but must select a ship-to address from a list of ship-to addresses.

Shipping charge type*

This term defines who pays for shipping orders. The following types of shipping charges are supported:

- Shipping charges are paid by the buyer to the seller. The seller calculates the shipping charges when the order is captured and the shipping costs become part of the order total.
- Shipping charges are paid by the buyer to the shipping carrier. The carrier calculates the shipping cost and assumes the responsibility of collecting payment from the buyer. Shipping costs are not calculated when the order is captured.

Referral Interface *

This term specifies the relationship between a store and a remote store. It defines the functions supported by the remote store and the parameters to be used in messages sent to the remote store.

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.

Business Business policies are a sharable resource. When you list business policies that can be used in a contract, the business policies listed are the ones owned by the store in which the contract is being created, and the business policies owned by any store with which there is a com.ibm.commerce.businessPolicy store relationship. For more information about sharing assets across stores within a site, refer to Chapter 14, "Relationships between stores," on page 129.

The following categories of business policies are provided in WebSphere Commerce:

Catalog business policies
: Catalog business policies define the scope and characteristics of the catalog of products for sale in a store including prices and the categorization of products in a store's catalog.

Payment business policies
: Invoicing, payment, and refund business policies define how a store accepts payments, pays refunds, and the format of a store's invoices.

Returns business policies
: Returns business policies define if refunds are accepted, the time period they are accepted for, and any re-stocking fees applied to returns.

Shipping business policies
: Shipping business policies define the shipping providers a store can use and the charges associated with each type.

Referral interface business policies
: Referral interface business policies define the relationship between a proxy store and a remote store.

Many contract terms and conditions reference business policies. This provides a measure of control over the nature of contracts a store enters into while still providing flexibility in creating the contract terms and conditions. For more information on business policies, refer to the *WebSphere Commerce Programming Guide and Tutorials*.

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 Business 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 or item 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 Development 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 WebSphere Commerce Development online help.

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 *WC_installdir*/samplestores directory.

   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 439.
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.” 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 *WC_installdir*/xml/sar directory.
6. Load the businesspolicy.xml file using the Loader package. For more information on the Loader package, see Part 10, “Publishing your store,” on page 319. 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 file” on page 189. If you want to create a more complex default contract, review the B2BTrading.dtd or Package.xsd file which defines the structure of a contract file. The B2BTrading.dtd file is located in the *WC_installdir*/xml/trading/dtd directory; the Package.xsd file is located in the *WC_installdir*/xml/trading/xsd directory.
8. Publish the contract using the ContractImportApprovedVersion command. For more information, see Chapter 39, “Publishing business accounts and contracts,” on page 395. Information on the ContractImportApprovedVersion command is also available in the WebSphere Commerce Development online help.

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 WebSphere Commerce Production online help.

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 the *WebSphere Commerce Programming Guide and Tutorials*.

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=&STORE_IDENTIFIER;&orgentity_dn=ORGANIZATION_DN
/>
```

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
  - ReferralInterface
- `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.RetrievePricesCmdImpl"
/>
```

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 439.

Creating a default contract 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.

You can create a default contract in XML, based on one of two formats: XSD or DTD. Refer to the sections below for details on how to create each type.

Creating a default contract file in XSD

To create a default contract in XSD format, 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"?>
<Package xmlns="http://www.ibm.com/WebSphereCommerce"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.ibm.com/WebSphereCommerce Package.xsd">
<BuyerContract state="Active" contractUsage="Default" comment="">
<ContractUniqueKey majorVersionNumber="1" minorVersionNumber="0"
 name="&STORE_IDENTIFIER; Default Contract" origin="Manual">
<ContractOwner>
<OrganizationRef distinguishName="ou=&ORGENTITYNAME;,&ORGANIZATION_DN;" />
</ContractOwner>
</ContractUniqueKey>
```

   Note that line breaks in the `Package` and `ContractUniqueKey` elements are for display purposes only.
2. Define the contract participants in your contract XML file. Use the following example as a guide:

```
<Participant role="Buyer">
</Participant>
<Participant role="Seller">
  <ParticipantMember>
    <OrganizationRef distinguishName=""ou=&ORGENTITYNAME;,&ORGANIZATION_DN;"/>
  </ParticipantMember>
</Participant>
```

   where `distinguishName` is the name of the user that is the seller for this contract in LDAP distinguished name format. For example, `uid=johnsmith,ou=People,o=ibm,o=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.
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." locale="en_US"/>
```

   where
   - `title` is a text description of the contract.
   - `locale` is the locale for the language that the title is in. The following values are predefined for `locale`:
     - en_US (English – US)
     - fr_FR (French)
     - de_DE (German)
     - it_IT (Italian)
     - es_ES (Spanish)
     - pt_BR (Brazilian Portuguese)
     - zh_CN (Simplified Chinese)
     - zh_TW (Traditional Chinese)
     - ko_KR (Korean)
     - ja_JP (Japanese)

     Additional values can be defined for `locale` by updating the language assets for your store. For more information on language assets, see Chapter 22, "Language assets," on page 215.

4. 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 `Package.xsd` 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.

   policy references
   The type of business policy that the term and condition references. Valid values are:
   - PricePolicyRef
   - ProductSetPolicyRef
   - InvoiceFormatPolicyRef
   - PaymentPolicyRef
   - ReturnApprovalPolicyRef
   - ReturnChargePolicyRef
   - ReturnPaymentPolicyRef
   - ShippingChargePolicyRef
   - ShippingModePolicyRef

   storeRef
   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:
     ```
     <PriceTCMasterCatalogWithOptionalAdjustment>
     </PriceTCMasterCatalogWithOptionalAdjustment>
     ```
   - Buyers pay any shipping charges to the seller:
     ```
     <ShippingTCShippingCharge>
         <ShippingChargePolicyRef policyName="StandardShippingChargeBySeller">
             <StoreRef name="&STORE_IDENTIFIER;">
              <Owner>
               <OrganizationRef distinguishName="ou=&ORGENTITYNAME;,
                &ORGANIZATION_DN;" />
              </Owner>
            </StoreRef>
       </ShippingChargePolicyRef>
     </ShippingTCShippingCharge>
     ```
   - Buyers can return products without any return charges. The products must be returned within the number of days defined in the `ApprovalByDays` business policy:
     ```
     <ReturnTCReturnCharge>
        <ReturnChargePolicyRef policyName="NoCharges">
           <StoreRef name="&STORE_IDENTIFIER;">
               <Owner>
                 <OrganizationRef distinguishName="ou=&ORGENTITYNAME;,
                 &ORGANIZATION_DN;" />
     ```

```
            </Owner>
        </StoreRef>
     </ReturnChargePolicyRef>
     <ReturnApprovalPolicyRef policyName="ApprovalByDays">
        <StoreRef name="&STORE_IDENTIFIER;">
            <Owner>
              <OrganizationRef distinguishName="ou=&ORGENTITYNAME;,
              &ORGANIZATION_DN;" />
            </Owner>
        </StoreRef>
      </ReturnApprovalPolicyRef>
  </ReturnTCReturnCharge>
```

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:

```
<ReturnTCRefundPaymentMethod>
   <ReturnPaymentPolicyRef policyName="UseOriginalPayment">
   <StoreRef name="&STORE_IDENTIFIER;">
        <Owner>
          <OrganizationRef distinguishName="ou=&ORGENTITYNAME;,
           &ORGANIZATION_DN;" />
        </Owner>
      </StoreRef>
    </ReturnPaymentPolicyRef>
  </ReturnTCRefundPaymentMethod>
```

5. Close the XML file as follows:

```
</BuyerContract>
</Package>
```

For more information about the use of @ and **&** see Appendix B, “Creating your data,” on page 439.

Creating a default contract file in DTD format

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 in DTD format, 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="ou=&ORGENTITYNAME;,&ORGANIZATION_DN;" />
     </Member>
   </ContractOwner>
   ```

   where `distinguishName` is the name of the user owning the contract in LDAP distinguished name format. For example, `uid=johnsmith,ou=People,o=ibm,o=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 22, "Language assets," on page 215.

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="ou=&ORGENTITYNAME;,&ORGANIZATION_DN;"/>
     </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,o=ibm,o=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>
               <Organization distinguishName="ou=&ORGENTITYNAME;,
                &ORGANIZATION_DN;" />
             </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="ou=&ORGENTITYNAME;,
              &ORGANIZATION_DN;" />
            </Member>
          </PolicyReference>
        </ReturnChargePolicyReference>
        <ReturnApprovalPolicyReference>
          <PolicyReference policyName="ApprovalByDays"
           policyType="ReturnApproval"
           storeIdentity="&STORE_IDENTIFIER;">
            <Member>
              <Organization distinguishName="ou=&ORGENTITYNAME;,
              &ORGANIZATION_DN;" />
            </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>
            <Organization distinguishName="ou=&ORGENTITYNAME;,
             &ORGANIZATION_DN;" />
          </Member>
        </PolicyReference>
      </ReturnTCRefundPaymentMethod>
    </ReturnTC>
  </TermCondition>
  ```

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

6. Close the XML file as follows:

```
</Contract>
</Trading>
```

For more information about the use of @ and **&** see Appendix B, "Creating your data," on page 439.

Chapter 19. Fulfillment assets

Fulfillment centers are used by stores as both inventory warehouses and shipping and receiving centers. A fulfillment center represents a place from which products are shipped to customers. Inventory counts are maintained separately for each fulfillment center. 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.

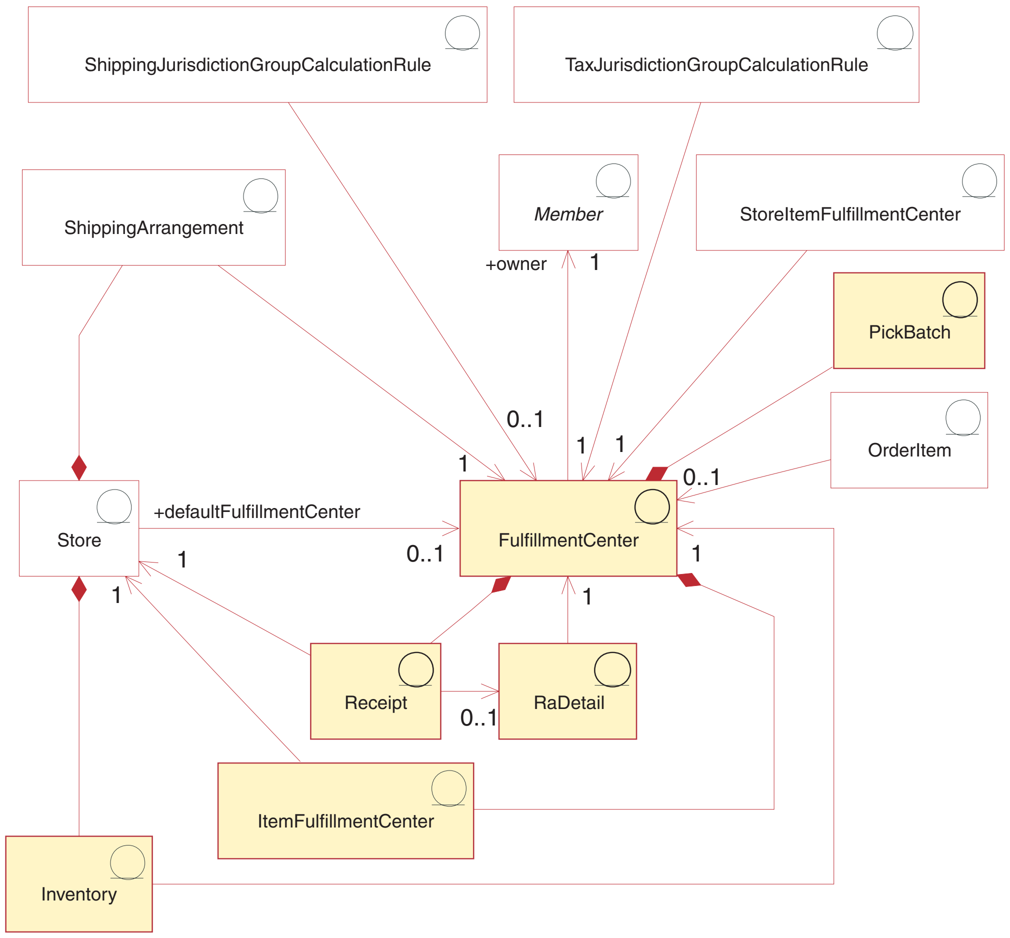
Products are configured for fulfillment with the Product wizard and the Product notebook. Product configuration provides options to track inventory, allow backorders, force backorders, release the product separately, and specify that the product should not be returned.

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 fulfillment 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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

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 can be associated with multiple fulfillment centers, and 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 123.

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 268.

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 29, “Inventory assets,” on page 265.

Shipping arrangements

A *shipping arrangement* is a relationship that enables a store to use a fulfillment center. 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 240.

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 ORDERITEMS table. For more information on order assets, see Chapter 30, “Order assets,” on page 273.

Like other entities, a fulfillment center has rules which govern some of its actions. Each fulfillment center 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 26, “Shipping assets,” on page 229, and “Understanding tax assets in WebSphere Commerce” on page 245.

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 10, "Publishing your store," on page 319.

Before creating assets, you should also be familiar with the material covered in Chapter 38, "Loading WebSphere Commerce database asset groups," on page 383.

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:

   - `*WC_installdir*/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 439.
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:
   - *WC_installdir*`/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="@seller_b2b_mbr_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. (For more information about support for different languages, see Chapter 34, "Globalization," on page 295.)
- `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 439.

Creating store fulfillment assets (non-ATP)

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 10, "Publishing your store," on page 319.

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 124. 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 162.
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:

   - *WC_installdir*/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 439.
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 `wcs.dtd` file in the *`WC_installdir`*`/schema/xml` directory or the DTD included in the store archive.
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.

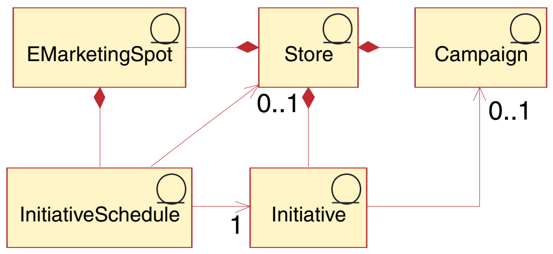
For more information about the use of @ and & see Appendix B, "Creating your data," on page 439.

Chapter 20. 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

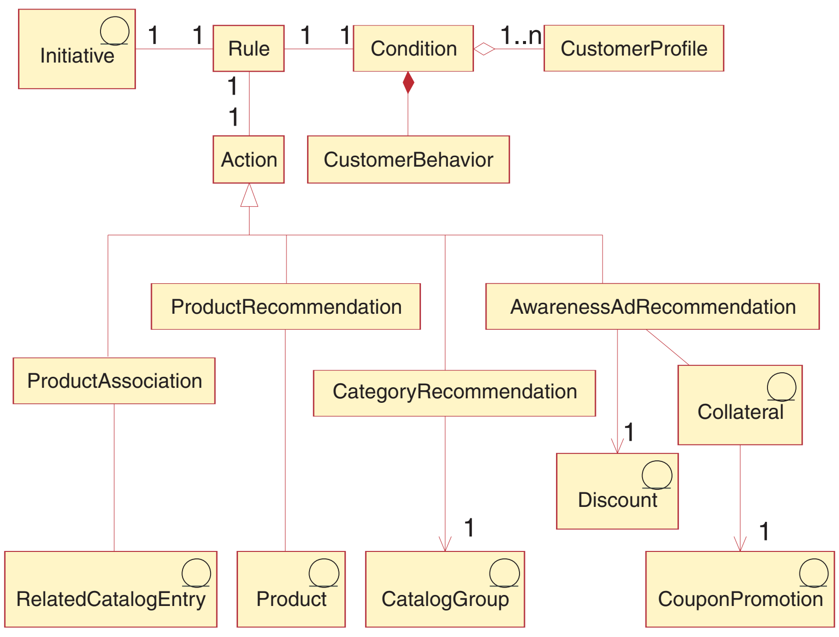
The following diagram illustrates the campaign assets in the WebSphere Commerce Server:



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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

Campaigns, and their associated assets are scoped to stores.

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. The campaign initiative information model is displayed below:



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.

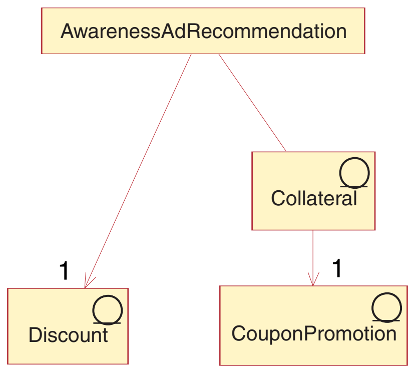
Campaign initiatives are capable of displaying four types of dynamic content:

- Suggestive selling initiative
- Collaborative filtering-based recommendation
- Awareness advertisement
- Merchandising association

Suggestive selling content provides rule-based category and product recommendations, targeted at a specific customer audience, based on a customer's profile, and other customers' behaviors. The targetable customer behaviors include the total value of the shopping cart, the contents of the shopping cart, and the contents of the customer's purchase history.

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

Awareness advertisements 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. Awareness advertisements follow the information model shown here:



Merchandising associations create up-sell and cross-sell opportunities, based on the static associations defined in the catalog. In order to create an initiative of this type, a method of selecting the source product in the association must be defined, so that the proper source is used when the e-Marketing Spot is invoked, to return the target products. The method can select the source based on either the content of the current page, the contents of the shopping cart, or the contents of the shopper's purchase history, as sources of the association.

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 42, "Adding e-Marketing Spots to your store," on page 429.

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. For more information about customer profiles, see Chapter 32, "Customer profiles," on page 281.

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 42, "Adding e-Marketing Spots to your store," on page 429.

Chapter 21. Payments instruments

WebSphere Commerce provides an optional component called WebSphere Commerce Payments (formerly known as IBM Payment Manager).

If you wish to use WebSphere Commerce Payments with your store, you must include a payment asset file in your store archive. Before publishing your store archive, ensure that your Payments instance is started, then, when the store archive is published, the payment asset file (included as part of the sample store archives) sets up the following information in WebSphere Commerce Payments:

- The merchant_ID in the WebSphere Commerce Payments database.
- The type of cassette used in the store.
- An account in the WebSphere Commerce Payments database for each currency specified as supported by the store in the payment asset file. If your store does not support the currency specified in the payment asset file, the account will not be created.
- A brand, or brands, for each account.

To set up a payment asset file and set up your store to use WebSphere Commerce Payments, do the following:

- Create payment data in the form of an XML file (`paymentinfo.xml`) that is loaded during store publish using the Administration Console. This configures WebSphere Commerce Payments with the merchant and the brand types specified for the store being published. For more information, see "Create payment assets using an XML file" on page 208.

  Note: `paymentinfo.xml` does not populate tables in the WebSphere Commerce Server database. It configures WebSphere Commerce Payments. `paymentinfo.xml` is only applicable if you are using offline credit card as the payment method.

  After the store archive has been published, you can place orders using the payment information set up in the sample store archive. If you want to add new brands, you must configure WebSphere Commerce Payments to work with each brand.
- If you will use an IBM payment cassette other than the OfflineCard or CustomOffline cassette, modify the sample store Web assets as described in "Customize environment for a payment cassette" on page 209.
- Complete the set up of WebSphere Commerce Payments for your store using the Administration Console or the WebSphere Commerce Payments user interface. If you use the Administration Console, menu items appear on the Payments menu. If you use the WebSphere Commerce Payments user interface, menu items appear under Administration in the navigation frame. For more information on setup tasks, see the topic *"Setting up WebSphere Commerce Payments for your store"* in the WebSphere Commerce Production online help.

If you intend to use a payment mechanism other than WebSphere Commerce Payments, the steps to follow to use your payment mechanism are similar to the following procedures.

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:

   - `WC_installdir/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 the `paymentinfo.xml`. The DTD file is located in the following directory: `WC_installdir/xml/sar`.
3. Enable or disable WebSphere Commerce Payments.
   a. Using the following example as your guide, in your XML file enable or disable WebSphere Commerce Payments 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 WebSphere Commerce Payments is enabled or disabled. When WebSphere Commerce Payments is disabled, your store will not be able to process payment transactions through the Payments component, although the Payments user interface will still function. If you disable the Payments component, there is no need to specify the other elements in the paymentinfo element.
   - `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.

Customize environment for a payment cassette

WebSphere Commerce provides sample stores that can use the OfflineCard or CustomOffline cassette as the payment cassette for handling payment transactions. These cassettes are automatically configured as being available for use as a payment method for the sample stores. The sample store Web assets need to be modified to use any other payment cassette. The instructions in this section describe how to customize your environment to use other IBM payment cassettes provided with WebSphere Commerce.

For a store to use an IBM payment cassette, you must first have selected the WebSphere Commerce Payments component for installation. Installation instructions are provided in the *WebSphere Commerce Installation Guide* for your platform. The WebSphere Commerce installation program installs both the Payments framework and the cassette software. You must then use the WebSphere Commerce Configuration Manager to perform necessary post-installation tasks, such as creating a Payments instance and adding a cassette to an instance. Refer to the *WebSphere Commerce Installation Guide* and the Configuration Manager online help for instructions on configuring a Payments instance.

After adding a payment cassette to a Payments instance, check the following customization steps to ensure that your WebSphere Commerce sample store can process payments through the payment cassette you have selected:

1. Modify the store .jsp file to specify the payment cassette.
2. Check the cassette's Cashier profile.
3. Check the cassette's .jsp file that supports the clerk order (guest order) placement page.
4. Configure the Merchant Settings.

These steps are described in the following sections.

Modify the store .jsp file

If you are not going to use the OfflineCard or CustomOffline cassette with the sample store, you must modify the store's .jsp file. By default, the store's .jsp file is set up to use the OfflineCard cassette; therefore, you must modify the file to use any other cassette. The FashionFlow store also uses the CustomOffline cassette.

For a list of .jsp files to review for possible modification, see Table 10 on page 211.

To modify the .jsp file, follow these steps:

1. Create a store in WebSphere Commerce using a sample store such as FashionFlow.
2. Go to the following directory:

   WAS_installdir/installedApps/*cell_name*/*WC_instance_name*.ear/Stores.war/

   ▶ 400 For iSeries, the path is:

   WAS_userdir/*WAS_instance_name*/installedApps/*cell_name*/*WC_instance_name*.ear/Stores.war/.The store you created has its own directory within the war directory.
3. From your store's directory, open the `OrderSubmitForm.jsp` file in a text editor.

Business The Contract Tools in the WebSphere Commerce Accelerator supports all payment cassettes. The `OrderSubmitForm.jsp` file must honor the payment terms and conditions of the contract that was set up between the Buyer organization and the Seller.

4. Search for the following text in the `OrderSubmitForm.jsp` file:

   ```
    if (info[i].getPolicyName().trim().equals("OfflineCard"))
   ```

   Change the name of the payment policy from `OfflineCard` to one of the following as appropriate:

   ```
   CustomOffline
   BankServACH
   Paymentech
   VisaNet
   VisaNet_PCard
   ```

   For more information about policies, see the POLICY database table in the WebSphere Commerce Development online help.

   The policy for CustomOffline supports processing of custom payment transactions such as cash on delivery or COD, Bill-Me-Later or coupons that are often executed outside of WebSphere Commerce Payments.

   The policy for BankServACH supports processing of online electronic check payments using the BankServ payment gateway that interfaces with the Automated Clearing House Network (ACH).

   The policy for Paymentech supports online authorization and settlement of credit card and non-PIN based debit card payments.

   The policy for VisaNet supports processing of credit card transactions using the Vital Processing Services or First Horizon Merchant Services (FHMS) financial network.

   Note: If you are using the Cassette for VisaNet with purchasing card support, select VisaNet_PCard rather than VisaNet.

   For more information about these cassettes, refer to the cassette supplements.

   If your store uses the Quick Checkout function, you should also change the name of the payment policy in these other files:

   ```
   ShoppingArea\CheckoutSection\QuickCheckoutSubsection\QuickCheckoutForm.jsp
   UserArea\AccountSection\QuickCheckoutProfileSubsection\QuickCheckoutProfileForm.jsp
   ```

5. (Optional) If you are using the credit card method of payment and need to add additional fields in the user interface to collect additional information from the user, review the `StandardCreditCard.jsp` file also for possible modifications. See Table 10 on page 211 for path information.

   To enable credit card brands to be displayed when doing a purchase with a particular payment method that involves the use of credit cards, be sure that an option value exists in the .jsp file for the payment method. For example, to enable credit card brands to be displayed when doing a purchase with the Paymentech payment method, search for `<select name="cardBrand">`. Add a new line underneath this text and add the following:

   ```
   <option value="Paymentech">Paymentech</option>.
   ```

Table 10. Store .jsp files to review

JSP file	Business model/sample	Purpose of change
/ShoppingArea/CheckoutSection/StandardCheckoutSubsection/OrderSubmitForm.jsp	Consumer direct (Fashion Flow or Express), hosted reseller	change payment policy name from OfflineCard
/ShoppingArea/CheckoutSection/QuickCheckoutSubsection/QuickCheckoutForm.jsp /UserArea/AccountSection/QuickCheckoutProfileSubsection/QuickCheckoutProfileForm.jsp	Consumer direct (Fashion Flow or Express)	change payment policy name from OfflineCard
/ShoppingArea/CheckoutSection/StandardCheckoutSubsection/StandardCreditCard.jsp	Consumer direct (Fashion Flow or Express)	enable credit card brand to display
/ShoppingArea/CheckoutSection/StandardCheckoutSubsection/StandardCreditCardDisplay.jsp	B2B direct (ToolTech), value chain - supply	enable credit card brand to display

Check the Cashier profile for the cassette

A WebSphere Commerce Payments Cashier profile should be available for any IBM payment cassette provided with WebSphere Commerce. The Cashier profile is used to create orders in the Payments component.

You may want to edit the Cashier profile to set certain parameters, such as the APPROVEFLAG and DEPOSITFLAG parameters. Since not all cassette parameters are alike, refer to the cassette supplement for more information about setting parameters:

- *WebSphere Commerce Payments CustomOffline Cassette Supplement*
- *WebSphere Commerce Payments OfflineCard Cassette Supplement*
- *WebSphere Commerce Payments Cassette for BankServACH Supplement*
- *WebSphere Commerce Payments Cassette for Paymentech Supplement*
- *WebSphere Commerce Payments Cassette for VisaNet Supplement*

Cashier profiles associated with IBM-provided payment cassettes include the following:

```
WC51_BankServACH.profile
WC51_CustomOffline_BillMe.profile
WC51_CustomOffline_COD.profile
WC51_OfflineCard.profile
WC51_VisaNet.profile
WC51_VisaNet_PCard.profile
WC_Paymentech.profile
```

The cassette profile should be stored in the WebSphere Commerce instances profile directory.

To locate the directory in which the profile is stored, look for the WebSphere Commerce configuration file for the instance you created. If you used the default instance name of ’demo’, the configuration file is:

WC_installdir/instances/demo/xml/demo.xml

400 For iSeries, the path would be:

WC_userdir/instances/demo/xml/demo.xml

Then, look at the directory specified by the ProfilePath attribute of the Payment Manager element in the configuration file. This attribute specifies where the profile should be located. If you used the default instance name of ’demo’, the directory path in which to store the profile would be the following:

WC_installdir/instances/demo/xml/payment

400 For iSeries, the path is:

WC_userdir/instances/demo/xml/payment

If you edit the cassette’s Cashier profile to set parameters, be sure to save the profile in the `*WC_installdir*/instances/*instance_name*/xml/payment` directory, where *`instance_name`* is the name of the instance you are using.

400 For iSeries, the path is:

WC_userdir/instances/instance_name/xml/payment

The actual cashier profile used by a payment business policy is specified by the profileName property value in the Properties field of the payment business policy. Refer to the POLICY database table in the online help for more information about business policies.

Check the cassette .jsp file

Payments are processed through the WebSphere Commerce Accelerator if an order clerk places a guest order on behalf of a customer. The payment data for the cassette is gathered using the cassette’s .jsp file.

The cassette .jsp file is called the “payment attribute page” in WebSphere Commerce. The actual page used is specified by the attrPageName property value in the Properties field of the payment business policy. For more information see the POLICY database table in the WebSphere Commerce Development online help. Both the store flow and WebSphere Commerce Accelerator flow should make use of the payments attribute page.

The cassette’s .jsp file should already be located in the following directory:

WC_installdir/wc.ear/CommerceAccelerator.war/tools/order/buyPages/
WAS_installdir/installedApps/*cell_name*/WC_demo.ear/CommerceAccelerator.war/tools/order/buypages

400 For iSeries, the path is:

QIBM/userdata/webas5/base/*WAS_instance_name*/installedApps/*cell_name*/WC_demo.ear/CommerceAccelerator.war/tools/order/buypages

If you wish to customize the “buy page” information, modify the .jsp file accordingly.

Configure Merchant Settings in WebSphere Commerce Payments

To configure the merchant for the IBM payment cassette, follow the instructions provided in the supplement for the cassette. Merchant settings can be modified through the WebSphere Commerce Administration Console or the Payments user interface (http://*host_name:port*/webapp/PaymentManager). You must have Payment Administrator or Merchant Administrator authority in WebSphere Commerce Payments to configure merchant settings.

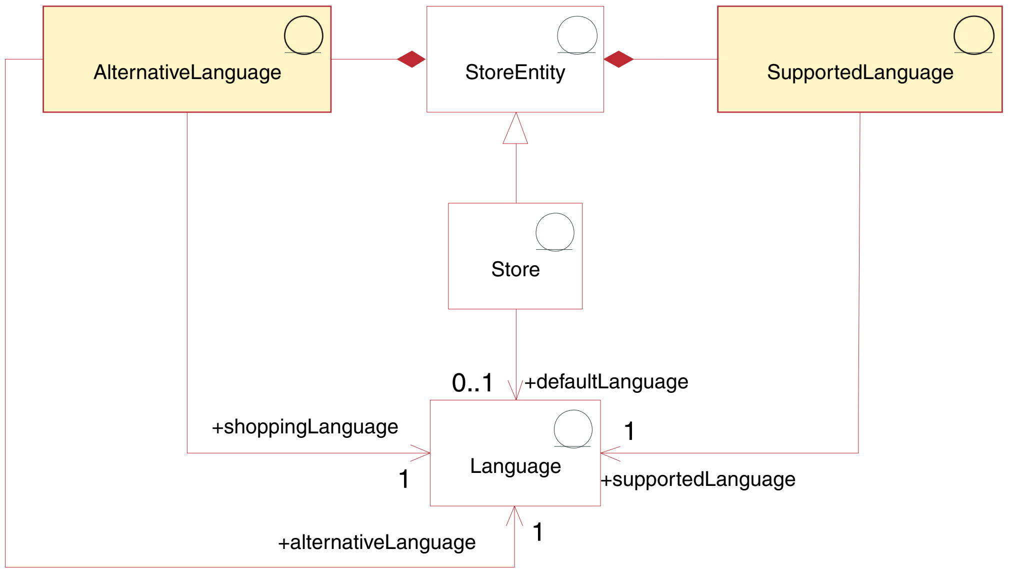
Chapter 22. Language assets

In WebSphere Commerce, your site can define many languages which can be used within it. At instance creation, the LANGUAGE table can have 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 language,
- Supported language,
- Alternative language, and
- Shopping language.

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 its 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.

For more information on adding a supported language, see "Adding a language to a store" on page 307.

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 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 store tools in WebSphere Commerce Accelerator
- In an XML file that will be loaded by the Loader package, or by the publishing tool in the Administration Console
- Editing the database directly using SQL inserts
- Using SQL edits and updates

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

For more information on defining store supported languages using store tools, 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 124.

Chapter 23. Currency assets

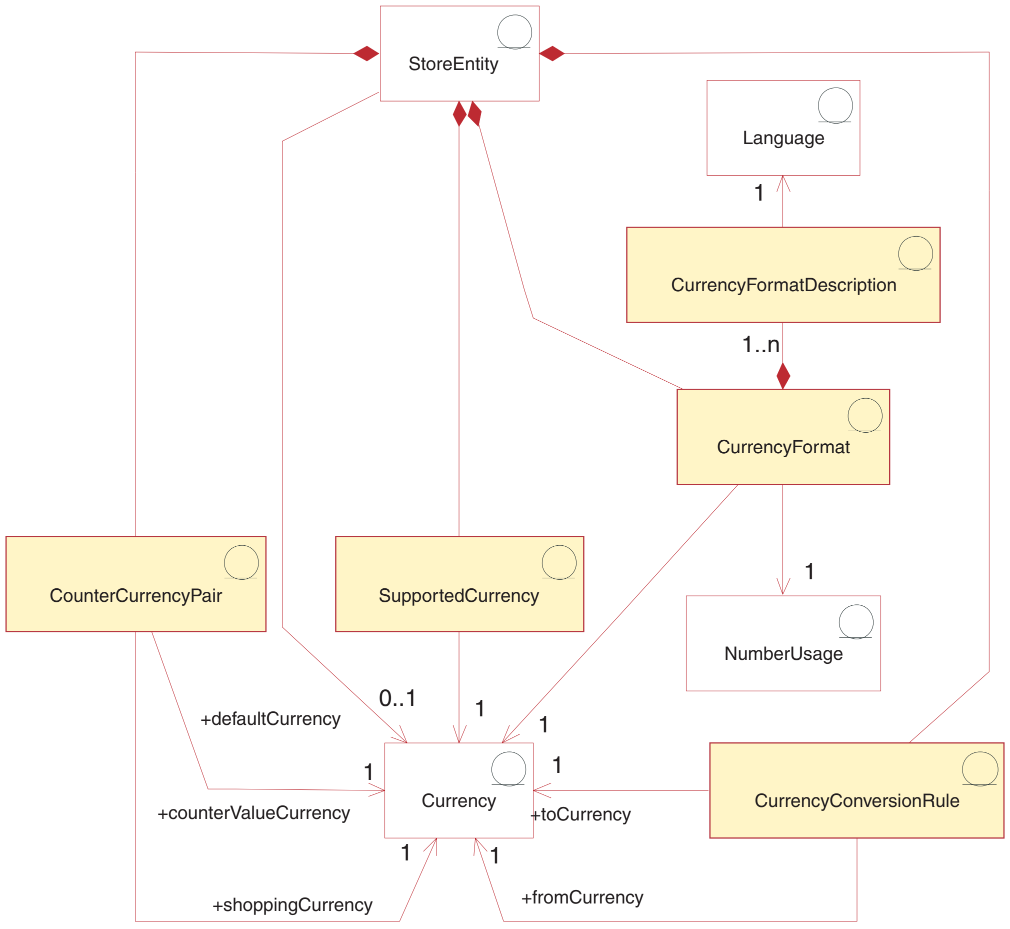
You can display prices in your site in one currency, or you can display multiple currencies by following the instructions provided for the euro (see "Counter currency" on page 219). For a site with multiple stores, you can use different currencies for the stores, or you can assign currencies to the store 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.

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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

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.

The currency format asset can be used by other stores as described in Chapter 14, "Relationships between stores," on page 129.

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 22, “Language assets,” on page 215. For more information about support for globalization, see Chapter 34, “Globalization,” on page 295. 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.

The supported currency asset can be used by other stores as described in Chapter 14, “Relationships between stores,” on page 129.

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.

The supported currency conversion rule asset can be used by other stores as described in Chapter 14, “Relationships between stores,” on page 129.

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.

The currency countervalue asset can be used by other stores as described in Chapter 14, “Relationships between stores,” on page 129.

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 Administration Console in WebSphere Commerce allows 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 Administration Console, see the WebSphere Commerce online help topic *"Changing store database assets."*

Note: The Administration Console works 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 working with currencies, see the WebSphere Commerce Development 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 10, "Publishing your store," on page 319.

Before creating assets, you should be familiar with the material covered in Chapter 38, "Loading WebSphere Commerce database asset groups," on page 383.

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:

   - *WC_installdir*/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 439.
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 `wcs.dtd` file in the `*WC_installdir*/schema/xml` directory or the DTD included in the store archive.
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 124.

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 175.

   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 439.

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

see the WebSphere Commerce Development online help.

Chapter 24. 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.

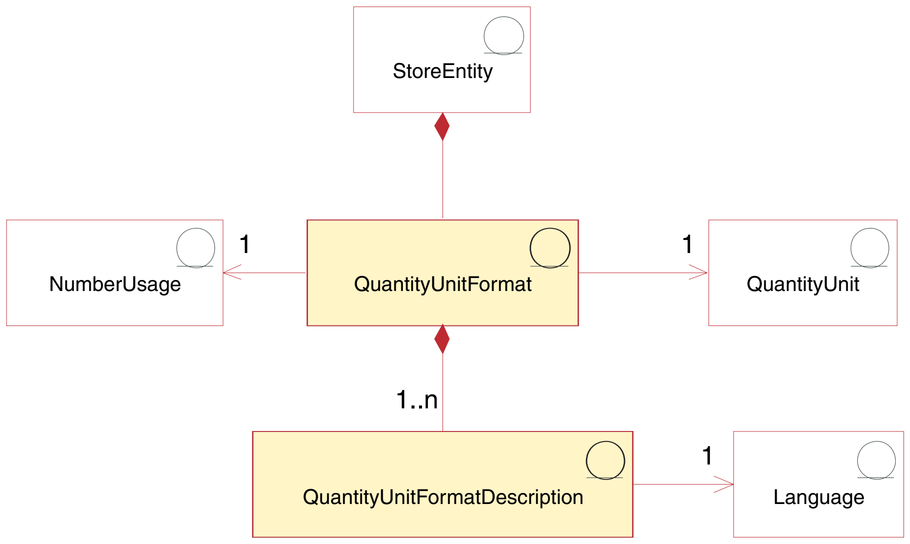
When Available to Promise (ATP) inventory is enabled (refer to the ALLOCATIONGOODFOR column of the STORE table), the inventory quantity unit is defined in the QUANTITYMEASURE column of the BASEITEM table. Otherwise, it is defined in the QUANTITYMEASURE column of the INVENTORY table.

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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

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.

Business Quantity units defined in one store may be used by other stores. In order for one store to use quantity units defined in another store a store relationship of type com.ibm.commerce.measurement.format must be created between the stores. For more information, see Chapter 14, "Relationships between stores," on page 129.

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 11, “Site assets,” on page 109.

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

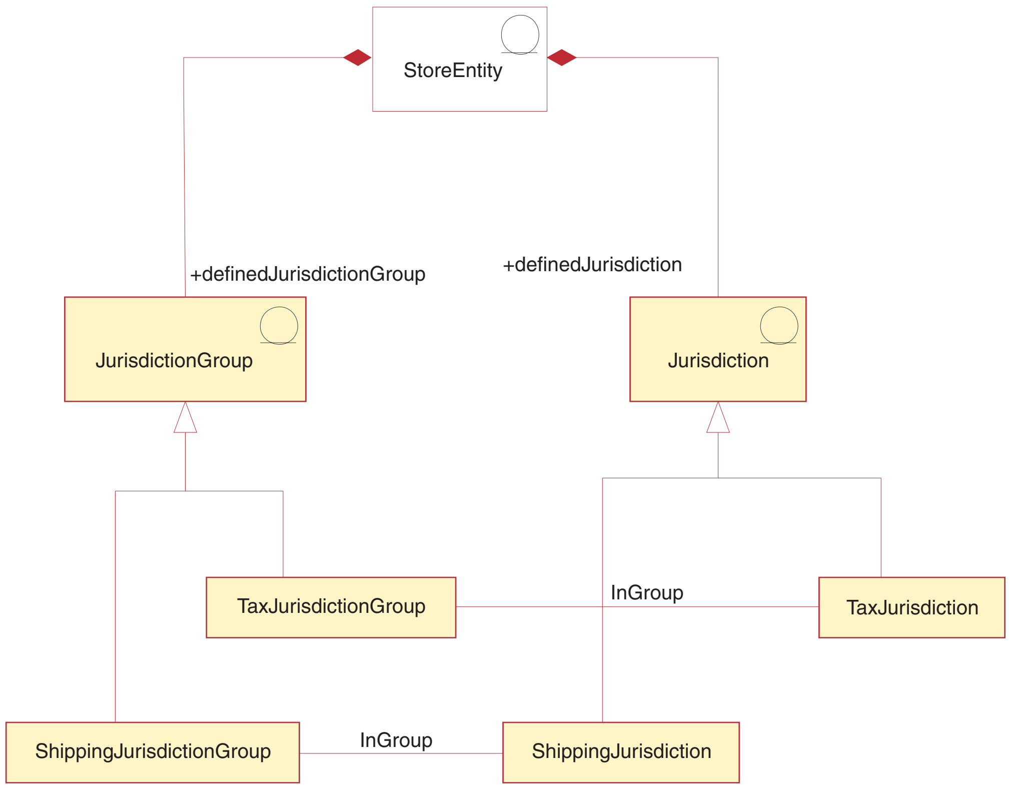
Chapter 25. Jurisdiction assets

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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

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 248 or “Creating shipping assets in WebSphere Commerce” on page 231.

Once jurisdictions have been created for your store, you can edit them or create new ones, using the Tax and Shipping notebooks in the store tools on the WebSphere Commerce Accelerator.

Note: A jurisdiction group is automatically created for every jurisdiction created. Jurisdictions are created for stores, but not for store groups.

Chapter 26. Shipping assets

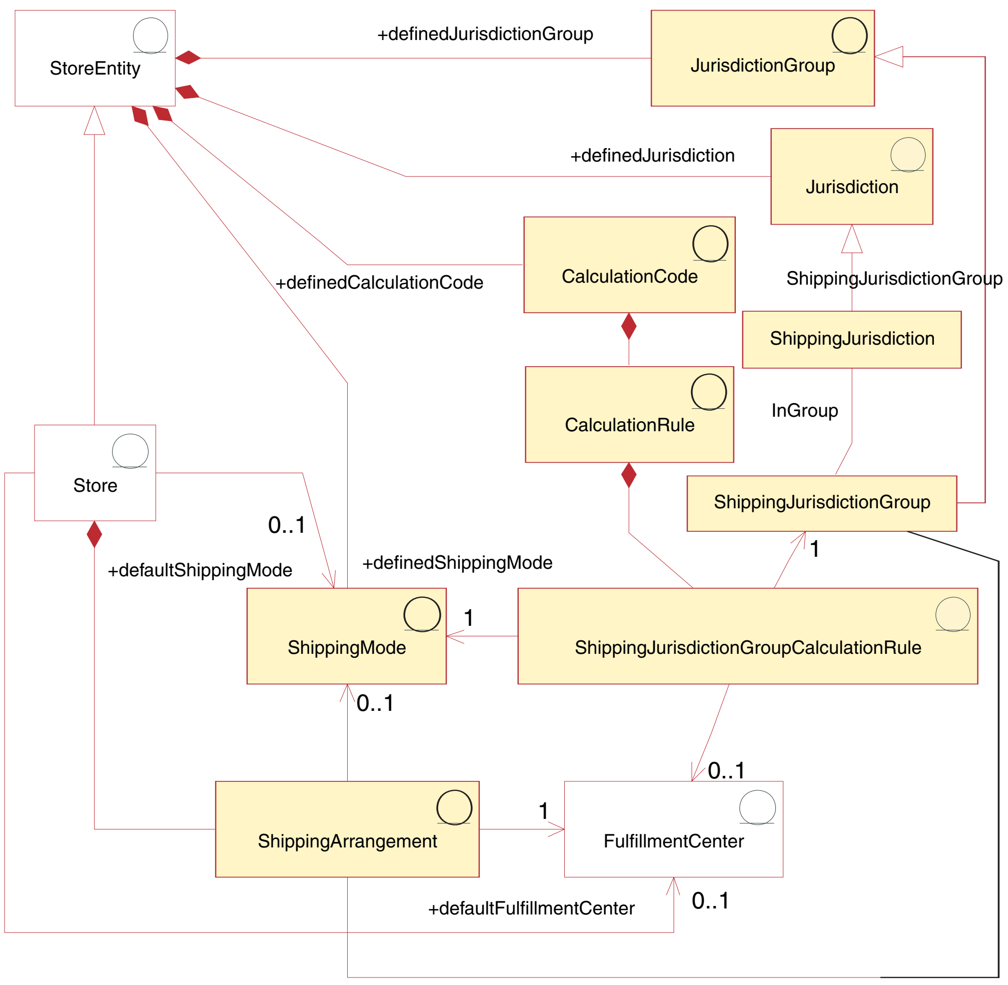
Shipping is how a store handles physically delivering goods to customers. In most cases, goods are shipped from a fulfillment center, a separate agency that is responsible for warehousing the store's goods.

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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

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 *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 codes, see the *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 shipping data models in the WebSphere Commerce online help.

Creating shipping assets in WebSphere Commerce

The shipping tools in WebSphere Commerce Accelerator allow you to create and edit certain shipping assets (for example shipping modes and jurisdictions), but not all shipping assets. The following list details the database tables that can be edited by the shipping tools:

- CALCODE
- CALCODEDSC
- CALRULE
- SHPJCRULE
- CRULESCALE
- CALSCALE
- CALSCALEDS
- CALRANGE
- CALRLOOKUP
- SHIPMODE

- SHPMODEDSC
- SHPARRANGE
- SHPARJURGP
- JURST
- JURSTGROUP
- JURSTGPREL
- CATENCALCD
- CATGPCALCD

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:

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

For information on creating or editing shipping assets using the WebSphere Commerce Accelerator, see the WebSphere Commerce Production online help. For information on creating new shipping assets in the form of an XML file, see "Creating shipping assets using an XML file."

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 10, "Publishing your store," on page 319. If you are creating a globalized 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 creating globalized stores, see Chapter 34, "Globalization," on page 295.

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 *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 information in Chapter 38, "Loading WebSphere Commerce database asset groups," on page 383.
3. 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:
   - *WC_installdir*/samplestores

   Note: The *WebSphere Commerce Sample Store Guide* 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.

4. Review the information in Appendix B, "Creating your data," on page 439.
5. 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 wcs.dtd file. The DTD file is located in the following directory:
   - *WC_installdir*/schema/xml
6. 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.

7. 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.

8. 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 globalized 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.

9. 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.

10. 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:
    - *WC_installdir*/schema
  - 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, FashionFlow creates a cost per order scale and a cost per item scale.

11. 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:

    - *`WC_installdir`*`/schema`

  – 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.

12. 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.

13. 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 439.

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 200.

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*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 information in Chapter 38, "Loading WebSphere Commerce database asset groups," on page 383.
3. 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:

   - *WC_installdir*/samplestores

   Note: The WebSphere Commerce online help contains information about each of the data assets contained in the sample stores.
4. Review the information in Appendix B, "Creating your data," on page 439.

5. 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 wcs.dtd file. The DTD files are located in the following directory:
   - *WC_installdir*/schema/xml
6. 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.
7. Repeat step 3 for each jurisdiction group, fulfillment center and rule association in your store.
8. 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.

9. 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 439.

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 124. For more information on creating catalog assets, see "Displaying store catalog assets" on page 162.

To create store-catalog-shipping assets, do the following:

1. Review the *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 information in Chapter 38, "Loading WebSphere Commerce database asset groups," on page 383.
3. 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:
   - *WC_installdir*/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.
4. Review the information in Appendix B, "Creating your data," on page 439.
5. 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 wcs.dtd file . The DTD files are located in the following directory:
   - *WC_installdir*/schema/xml
6. 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 439.

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 information in Chapter 38, "Loading WebSphere Commerce database asset groups," on page 383.
2. 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:

   - *WC_installdir*/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.
3. Review the information in Appendix B, "Creating your data," on page 439.
4. 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 wcs.dtd file. The DTD files are located in the following directory:
   - *WC_installdir*/schema/xml
5. 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 439.

Chapter 27. Tax assets

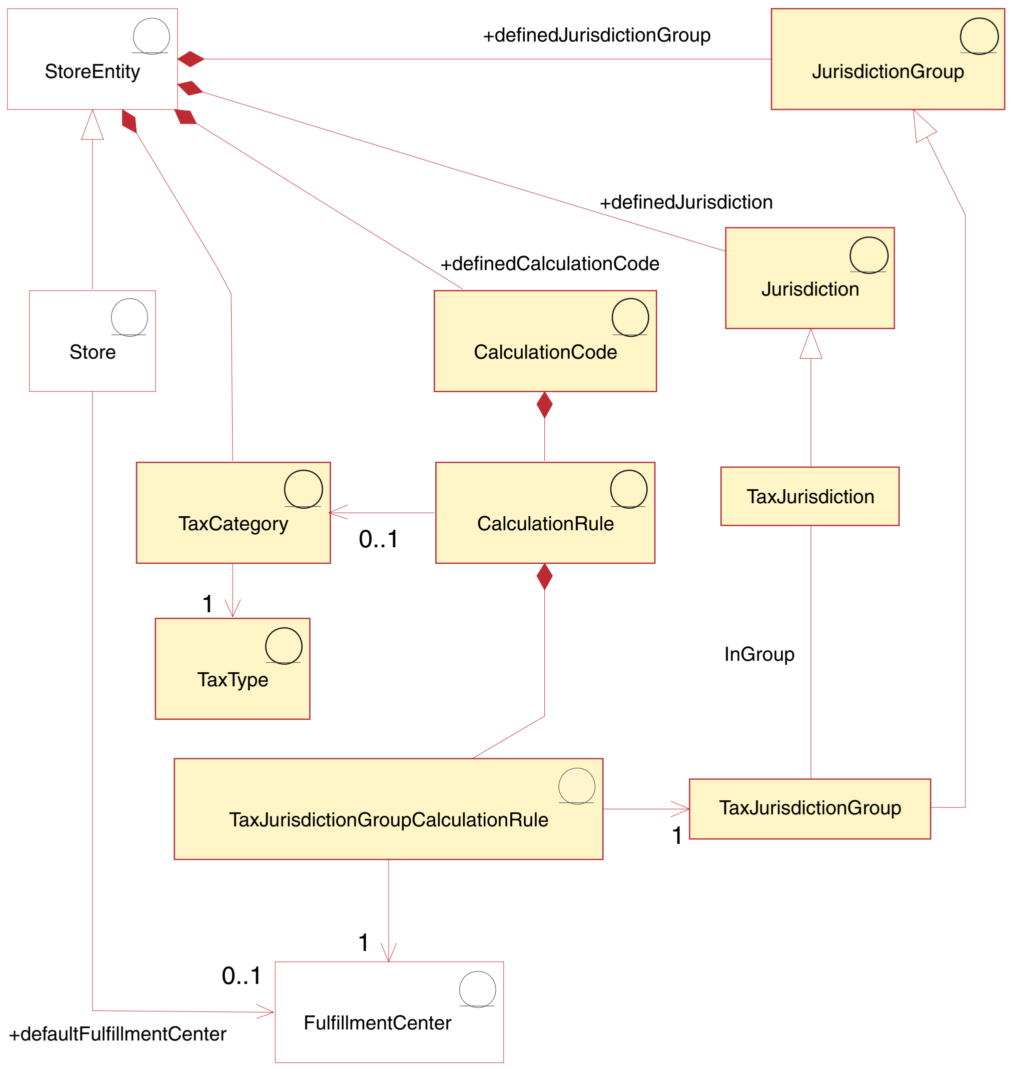
In order to charge and collect taxes on the goods and services your store provides, a store created with WebSphere Commerce must include the following:

- 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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

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 data models in the WebSphere Commerce online help.

Creating tax assets in WebSphere Commerce

The tax tools in the WebSphere Commerce Accelerator allow you to create and edit certain tax assets (for example tax categories and jurisdictions), but not all tax assets.

The following list details the database tables that can be edited by the tax tools:
- CALCODE
- CALCODEDSC
- CALRULE
- TAXJCRULE
- CRULESCALE
- CALSCALE
- CALSCALEDS
- CALRANGE
- CALRLOOKUP
- TAXCGRY
- TAXCGRYDS
- JURST
- JURSTGROUP
- JURSTGPREL
- CATENCALCD
- CATGPCALCD

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:
- Create new or edit the existing tax assets from one of the sample stores provided with WebSphere Commerce.
- Create new tax assets in the form of an XML file.

For information on editing the tax assets in an existing store archive, or general tax information, 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” on page 249

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 10, "Publishing your store," on page 319. If you are creating a globalized 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 information in Chapter 38, "Loading WebSphere Commerce database asset groups," on page 383. 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:
   - *WC_installdir*/samplestores
3. Review the information in Appendix B, "Creating your data," on page 439.
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 file is located in the following directory:
   - *WC_installdir*/schema
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 Jursticton 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:
    - *WC_installdir*/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:
    - *WC_installdir*/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:
      - *WC_installdir*/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:
        - *WC_installdir*/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 439.

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 200.

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:
   - *WC_installdir*/samplestores

   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 439.
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:
   - *WC_installdir*/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 439.

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 124. For more information on creating catalog assets, see “Displaying store catalog assets” on page 162.

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:
   - *WC_installdir*/samplestores
3. Review the information in Appendix B, “Creating your data,” on page 439.
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:
   - *WC_installdir*/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.

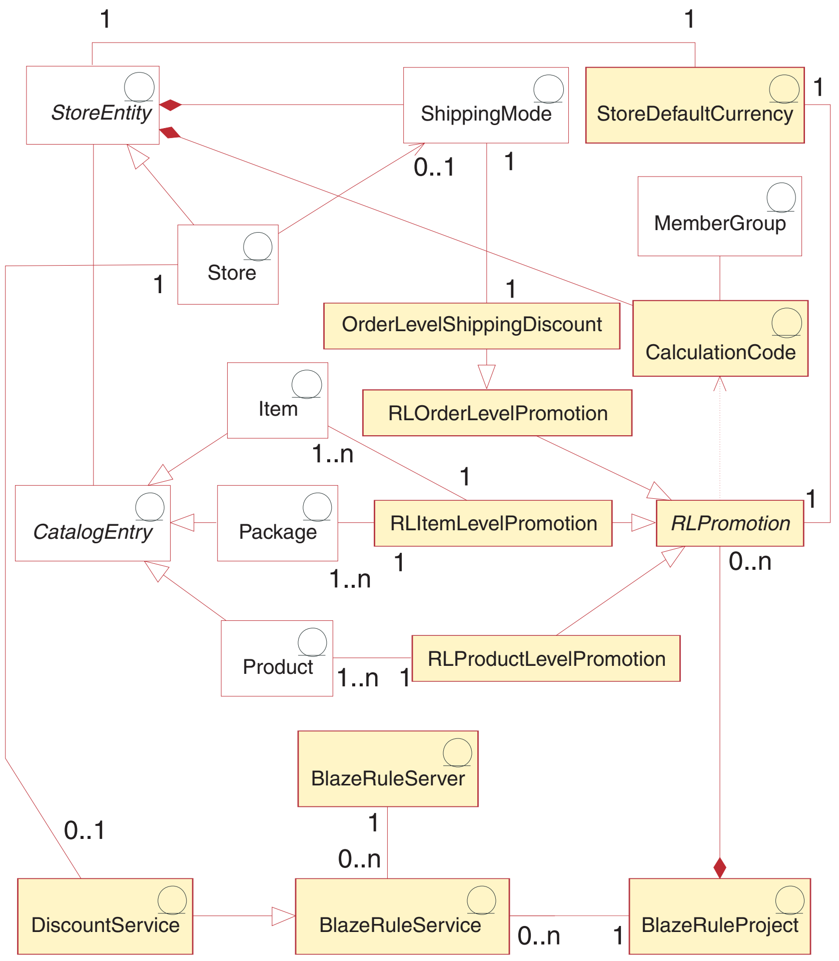
For more information about the use of @ and **&** see Appendix B, “Creating your data,” on page 439.

Chapter 28. Discount assets

Discounts allow you to offer customers price incentives to drive sales, or promote a product. WebSphere Commerce has two available discount implementations; rule-based discounts, and schema-based discounts. Either implementation enables 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. Rule-based discounts go beyond these discount types to offer shipping discounts, and free gifts with qualifying purchases.

Understanding rule-based discounts in WebSphere Commerce

The following diagram illustrates the rule-based 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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

Store default currency

This is the default currency for the store as defined in the STOREENT table. Rule-based discounts are defined in this currency, but may be evaluated on demand, using any currency supported by the store. For more information on currency usage, see Chapter 23, "Currency assets," on page 217.

Calculation code

A *discount* is represented in the calculation framework by a discount *calculation code*. A discount calculation code indicates how the discount is calculated for order items by the corresponding calculation rule.

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.

RLPromotion

This is the parent object for rule-based discounts. While RLPromotion is the object type name, it should be understood that it corresponds to a rule-based discount. Each rule-based discount has a name, various descriptions which display in different circumstances, a priority, a target segment (predefined customer profile), and an execution schedule governing both dates and times.

The priority attribute should be further clarified. The priority attribute exists to help resolve conflicts when there are multiple discounts that can be applied concurrently. Applicable discounts are applied in the order defined by their respective priority values, in descending order. That is, the discount with the highest priority value is applied first.

All of the child objects listed below further categorize the rule-based discount types, and introduce values specific to the discount type where required. Each of these objects also contains the appropriate logic to manipulate the domain XML file, which defines the discount.

RLProduct level promotion

These objects represent product level rule-based discounts. This class is derived from the RLPromotion class. This class requires an additional attribute, `SKU`, which identifies the target product.

RLItem level promotion

These objects represent item level rule-based discounts. This class is derived from the RLPromotion class. This class requires an additional attribute, `catEntryID`, which identifies the target product. These item level promotions are also used to target prebuilt kits with rule-based discounts since they are separately orderable, with their own `catEntryID` and price. Bundles and dynamic kits are not targetable by rule-based discounts.

RLOrder level promotion

These objects represent order level rule-based discounts. This is a derivative of the RLPromotion. This class requires an additional attribute, `inCombineWithProductLevelDiscount`, which determines whether the order level discount can be applied at the same time as a product level promotion.

Order level shipping discount: This class is derived from the RLOrder level promotion class. This class requires additional attributes which define the shipping method to use, and the discounted rate.

Blaze rule project

The Blaze rule project contains all of the currently defined discounts for a store, generated from the domain XML file. The rule project resides in the file system, and is used to populate the discount service.

Blaze rule service

This is an interface WebSphere Commerce uses to communicate with the Blaze rule server.

Discount service

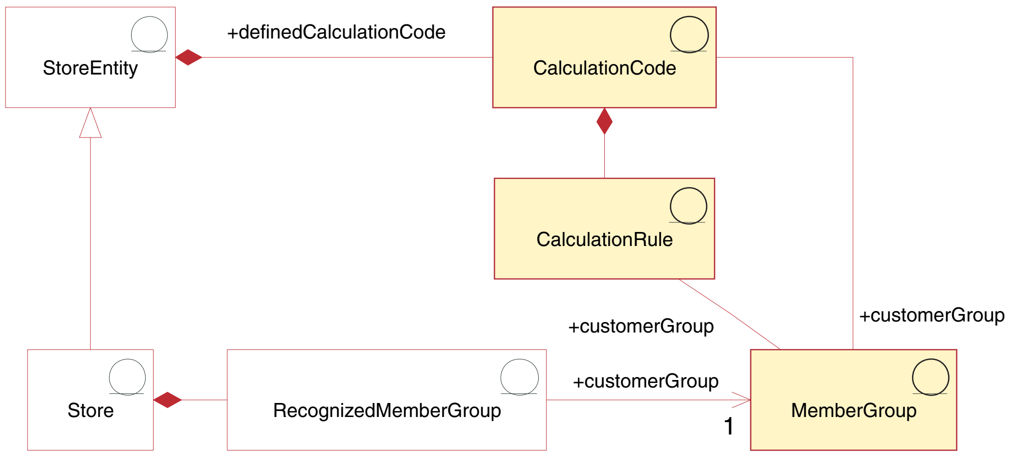
Extends from the Blaze rule service, and contains an instance of the rule project wherein evaluation and calculation take place. This service passes the discount context, an object containing information about the applicable discounts, to the WbeSphere Commerce calculation framework..

Blaze rule server

Blaze software that evaluates the rule project whenever a request arrives from the order processing commands.

Understanding schema-based discounts in WebSphere Commerce

The following diagram illustrates the schema-based discount structure in the WebSphere Commerce Server.



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. 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 29. 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.

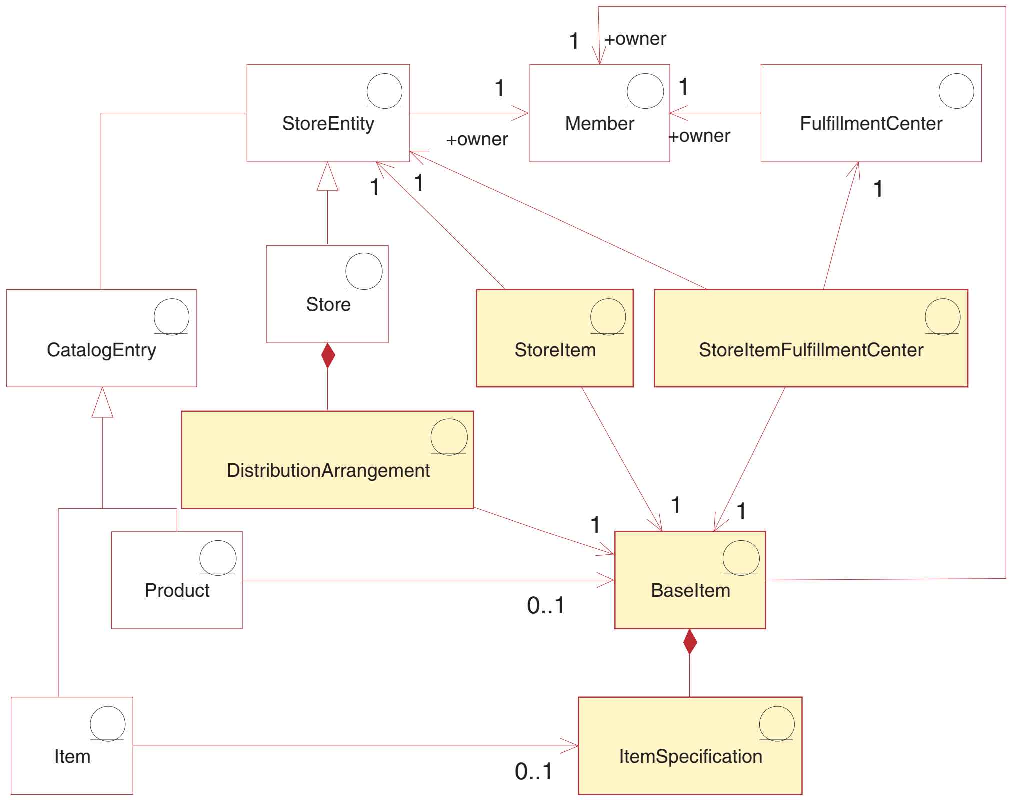
Ad hoc inventory receipts are created when inventory arrives at a fulfillment center without a corresponding expected inventory record. This could be due to an unexpected inventory arrival, or it could be the choice of the merchant or seller not to use expected inventory records to record inventory receipts.

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

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 available-to-promise (ATP) inventory and non-ATP inventory. A store may use either ATP or non-ATP inventory methods. A store is considered to be enabled for ATP if the ALLOCATIONGOODFOR column in the STORE database table contains a value greater than zero. 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 97. For more information on the conventions used in this diagram, see Appendix A, “UML legend,” on page 437.

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

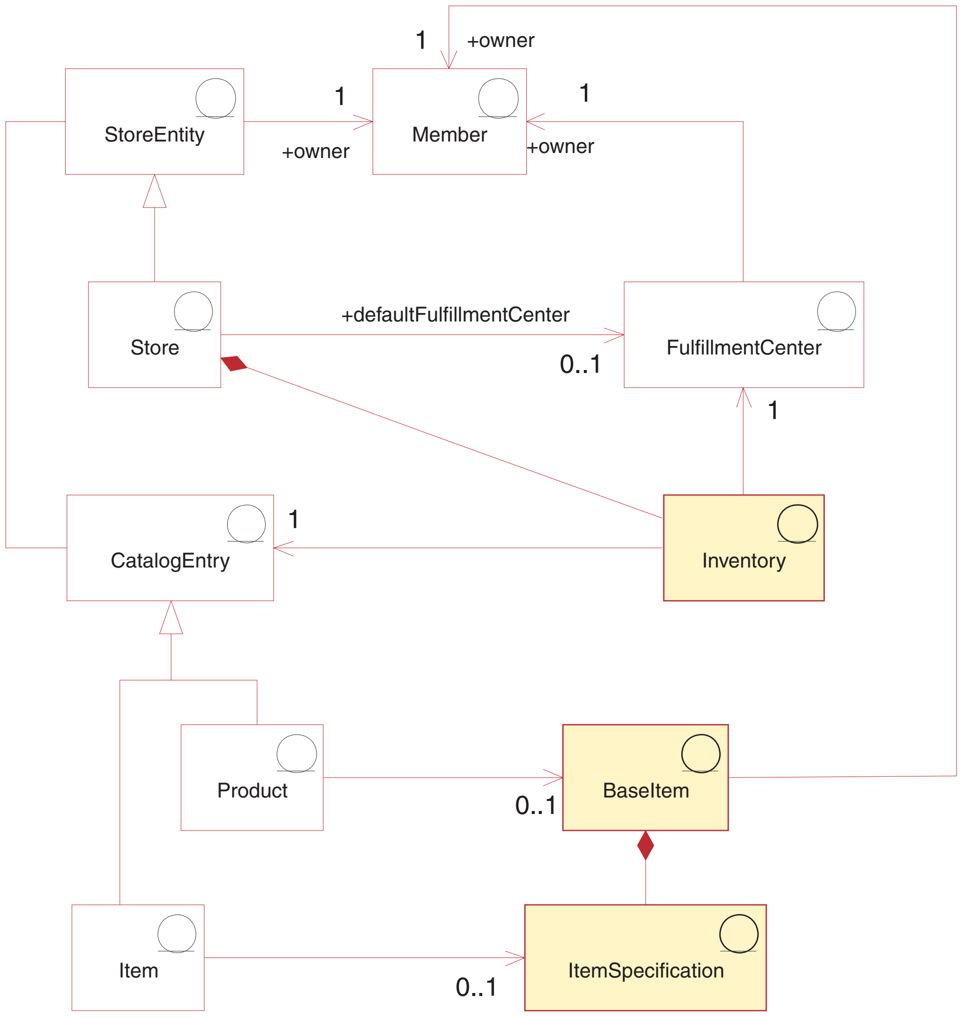
A *distribution arrangement* is associated with a base item, enabling a store to sell its own inventory. Distribution arrangements are stored in the DISTARRANG table.

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.

The store item asset can be used by other stores as described in Chapter 14, "Relationships between stores," on page 129.

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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

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. For more information on fulfillment assets, seeChapter 19, "Fulfillment assets," on page 197.

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.

Managing inventory adjustment codes

Inventory adjustment codes are provided by WebSphere Commerce to enable users to specify the reason why a particular adjustment to the inventory is being made. The following table lists the inventory adjustment codes that are initially provided with WebSphere Commerce. You can also view these codes by entering a SELECT * database query against the INVADJCODE and INVADJDESC database tables. As store developer, you can add, change, or delete these codes as necessary to fit the needs of your inventory environment. The codes described in the table below should be thought of as examples of the types of codes that you can use. You can achieve a more customized environment by adding your own adjustment codes or by modifying the codes that are preloaded.

Table 11. Preloaded inventory adjustment codes

Adjustment code	Description	Explanation
RTND	Returned	Use this code to specify that an item* was returned to inventory. For example, a customer may return an item from an order because it was the wrong color or size. This code is preloaded for use by all sample stores.
EXPD	Expired	Use this code to specify that an item had expired in the inventory. For example, prescription drugs or products with a short shelf life (such as dairy products) could expire. This code is preloaded for use by all sample stores.
DMGD	Damaged	Use this code to specify that an item is damaged (for example, it may be dented, scratched, broken, or defective). This code is preloaded for use by all sample stores.
LOST	Lost	Use this code to specify that an item was lost in or stolen from the fulfillment center or storage location. This code is preloaded for use by all sample stores.
MSCT	Miscount	Use this code to specify that the count made of the item was previously wrong. This code is preloaded for use by all sample stores.
PCNT	Physical Count	Use this code to specify the actual physical count of the item. This code is preloaded for Consumer direct and B2B direct sample stores but not hosted stores.
SPLG	Spoilage	Use this code to specify that an item has spoiled. This code is preloaded for Consumer direct and B2B direct sample stores but not hosted stores.
DISC	Discard	Use this code to indicate that an item has or will be discarded from inventory. This code is preloaded for Consumer direct and B2B direct sample stores but not hosted stores.
*The word ″item″ is used in the general sense here. Inventory goods may be items, products, SKUs, bundles, and packages.		

In this table, the value shown in the Adjustment code column represents the value found in the ADJUSTCODE column of the INVADJCODE database table. The value shown in the Description column represents the value found in the DESCRIPTION column in the INVADJDESC database table.

To add, change, or delete inventory adjustment codes, use the Load command (massloader) command. Details about the Load command are provided in “Load command” on page 349.

For more information about how data is organized in the sample stores, see Chapter 38, “Loading WebSphere Commerce database asset groups,” on page 383.

Before using the Load command, be sure that the `wcs.dtd` file is included in your path (*WC_installdir*`\schema\xml`). The `massloader.cmd` file must also be in your path (*WC_installdir*`\bin`).

Adding inventory adjustment codes

To add inventory adjustment codes for your store, do the following:

1. Create an XML file (with a name of your choice) similar to the following example, and place it in a location where the Loader can find it. In the XML file, specify values for the invadjcode and invadjdesc elements to add the new inventory adjustment code to two database tables: INVADJ and INVADJDESC. The invadjcode_id should be the same value in both tables for any given code.

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE import SYSTEM "wcs.dtd">
   <import>
   <invadjcode invadjcode_id="404" adjustcode="BRKN" storeent_id="-1"
   markfordelete="0" />
   <invadjdesc invadjcode_id="404" description="BROKEN" language_id="-1" />
   </import>
   ```

   where:

   - `invadjcode_id` is an adjustment code identifier you assign. The identifier is used internally and is not displayed to users in the WebSphere Commerce user interface.
   - `adjustcode` is a 4-character code that uniquely identifies the code, suitable for display in a user interface.
   - `storeent_id` is the identifier for the store entity or store group. To add this inventory adjustment code to all stores, enter a value of ″–1″. Otherwise, provide a specific store entity or store group identifier.
   - `description` is a text description of the inventory adjustment code, suitable for display in a user interface.
   - `language_id` is the default language for information displayed to customers shopping in the store. For more information about language support, see Chapter 34, "Globalization," on page 295.

   If you have a need to add more than one inventory adjustment code, you can specify multiple codes in the XML file.
2. Run the Load command against your XML file to load your data into the two target databases.

   Windows

   ```
   massload.cmd -dbname dbname -dbuser dbuser -dbpwd dbpwd -infile
   xml_file_name -method sqlimport
   ```

   AIX Solaris Linux Become *wasuser* first (the WebSphere Application Server user ID):

   ```
   su - wasuser
   ```

   Then, issue this command:

   ```
   ./massload.sh -dbname dbname -dbuser dbuser -dbpwd dbpwd -infile
   xml_file_name -method sqlimport
   ```

   400 Start a QShell session (STRQSH). Then, run the following command from the `WC_installdir/bin` directory:

   ```
   massload.sh -dbname dbname
   -dbuser dbuser -dbpwd dbpwd -infile
   xml_file_name -method sqlimport
   ```
3. Confirm the addition of the inventory adjustment codes by running a database query to view the new code values were added to both tables.

Changing inventory adjustment codes

To change the description of an inventory adjustment code, do the following:

1. Follow steps similar to adding a new code, but change the values for the adjustcode and description elements in the XML file for the particular invadjcode_id. For example:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE import SYSTEM "wcs.dtd">
   <import>
   <invadjcode invadjcode_id="404" adjustcode="DEFE" storeent_id="-1"
   markfordelete="0" />
   <invadjdesc invadjcode_id="404" description="DEFECTIVE" language_id="-1" />
   </import>
   ```

   In this example, the previous adjustment code and description (BRKN, BROKEN) are changed to the new values shown. If necessary you can change the invadjcode_id values also.

   To find the invadjcode_id you can issue the massextract.cmd (the Extract command). More information about the Extract command is available in the WebSphere Commerce Development online help. The massextract.cmd extracts the data from the database and puts it into an XML file. You can then browse the file to find the invadjcode_id value. As an alternative, you can also run a SELECT * query on the INVADJCODE and INVADJDESC database tables.
2. Run the same massload.cmd as for adding a new adjustment code:

   Windows

   ```
   massload.cmd -dbname dbname -dbuser dbuser -dbpwd dbpwd -infile
   xml_file_name -method sqlimport
   ```

   AIX Solaris Linux As *wasuser*, run this command:

   ```
   ./massload.sh -dbname dbname -dbuser dbuser -dbpwd dbpwd -infile
   xml_file_name -method sqlimport
   ```

   400 Start a QShell session (STRQSH). Then, run the following command from the `WC_installdir/bin` directory:

   ```
   massload.sh -dbname dbname
   -dbuser dbuser -dbpwd dbpwd -infile
   xml_file_name -method sqlimport
   ```
3. Confirm that the inventory adjustment codes were changed in the database tables.

Deleting inventory adjustment codes

To delete an inventory adjustment code from the WebSphere Commerce database tables, do the following:

1. Create an XML file containing the codes you want to delete. Refer to the sample XML file shown in "Adding inventory adjustment codes" on page 270 for an example.
2. Run the massload.cmd but be sure to specify the delete method as shown

   Windows

   ```
   massload.cmd -dbname dbname -dbuser dbuser -dbpwd dbpwd -infile
   xml_file_name -method delete
   ```

   AIX Solaris Linux As *wasuser*, run this command:

```
./massload.sh -dbname dbname -dbuser dbuser -dbpwd dbpwd -infile
xml_file_name -method delete
```

▶ 400 Start a QShell session (STRQSH). Then, run the following command from the `WC_installdir/bin` directory:

```
massload.sh -dbname dbname
-dbuser dbuser -dbpwd dbpwd -infile
xml_file_name -method delete
```

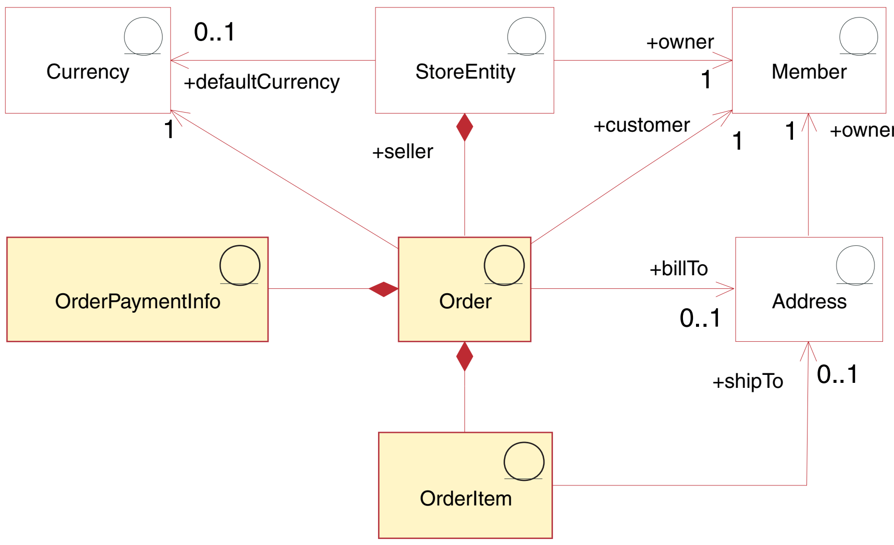
3. Confirm that the inventory adjustment codes were deleted from the database tables.

Chapter 30. 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 97.

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

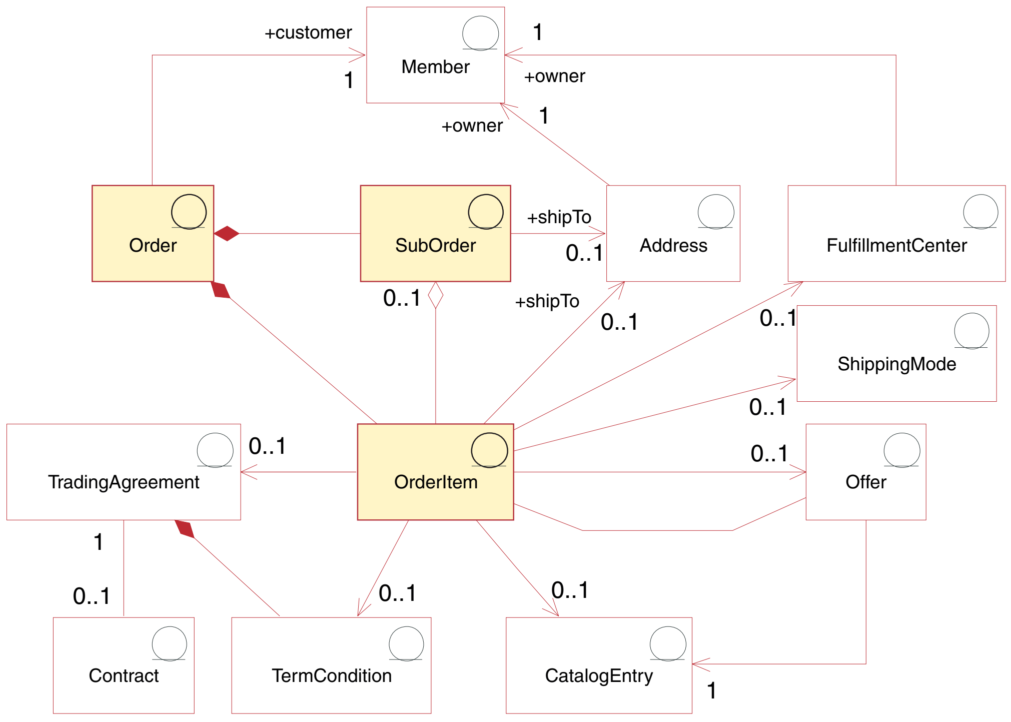
Once a customer has selected a preferred shopping currency, all payment will be processed in that currency. Depending on the store's payment support and policies, customers can pay for purchases using online payment (where a customer provides payment information over the Internet on the store's site) 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 WebSphere Commerce Payments for the store, you can set up the store to accept offline payment, or use other payment protocols 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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

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

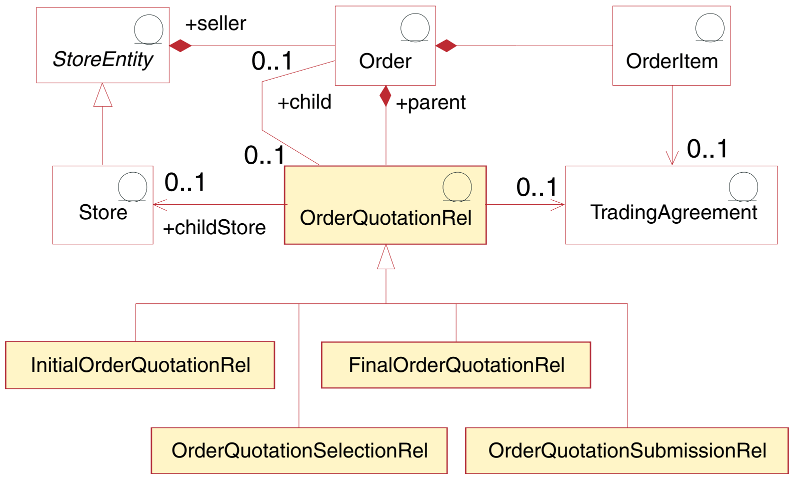
An order item can be associated with zero or one of each of the following objects.

- 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 Business 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.

Order quotation relationships

The following diagram describes order quotation relationships.



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.

An order quotation relationship (OrderQuotationRel) is a relationship between a parent shopping cart order and a child quotation-related order. In the Value chain (demand) business model, where selling is done through channels, channel area shoppers can use a shopping cart to obtain price quotations, from multiple stores, for products or services, select quantities from the resulting quotations, and submit orders to the stores that provided the quotations. Order quotation relationships indicate the store and contract (trading agreement) from which the quotation was requested.

A parent shopping cart can have the following types of child quotation-related orders for each quotation store and contract pair: an initial quotation, a current quotation selection, a final quotation, and one or more submitted orders.

The child order of an InitialOrderQuotationRel represents the initial quotation received for items in a parent shopping cart order. An initial quotation may include alternate and related products not explicitly mentioned in the parent order.

The child order for an OrderQuotationSelectionRel represents the items, and their quantities, currently selected from the initial or final quotation. An OrderQuotationSelectionRel child order can be submitted to its store for processing.

The child order for a FinalOrderQuotationRel represents the final quotation received for items in a selection order. A final quotation provides prices and quantities for only the items in the selection order when the request for a final quotation was sent.

The child order of an OrderQuotationSubmissionRel represents a submitted order. When a selection order is submitted, its OrderQuotationSelectionRel object is changed to be an OrderQuotationSubmissionRel object.

Creating order assets in WebSphere Commerce

A customer can place orders from a store, or request that a Customer Service Representative for the store help complete this task (using the WebSphere Commerce Accelerator). To create an order on behalf of a Consumer direct 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 direct customer, see the help topic ″Creating an order for a business user″.

Chapter 31. Vendor assets

A vendor represents a source for merchandise received at a fulfillment center, or expected to be received at a fulfillment center. Depending on the store model, vendors are defined by the Buyer, Product Manager, Seller (Merchant), or some other authorized role. Through the WebSphere Commerce Accelerator, you can view a list of all vendors, create a new vendor, change an existing vendor, and delete a vendor.

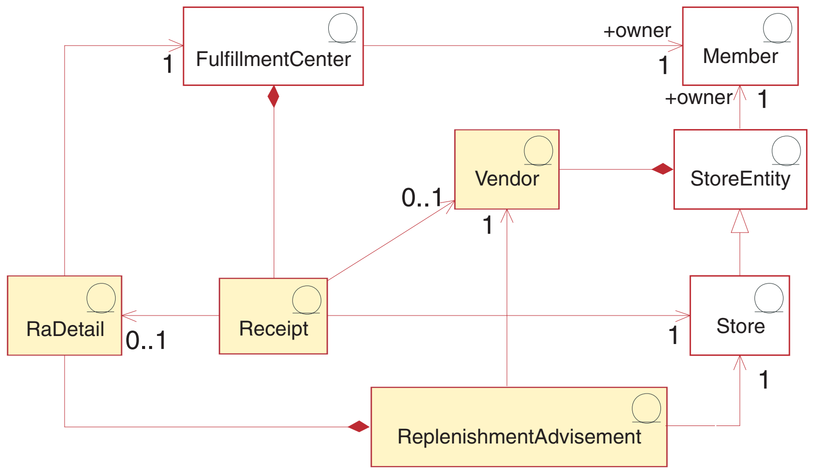
The vendor record includes information about the vendor, such as the name, address, and contact information. Vendors must be created before the store can create expected inventory records.

Expected inventory records are shown by vendor, External ID (usually a purchase order number), and order date on the Expected Inventory page.

For more information about managing vendor records, refer to the *Vendor information* topics in the WebSphere Commerce Production online help.

Understanding vendor assets in WebSphere Commerce

The following section describes the relationships that vendors have 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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

Vendor represents the source from which a store receives merchandise, and can be thought of as a supplier to the store.

Replenishment advisement is synonymous with expected inventory record.

Receipts refer to inventory receipts. Normally, a receipt results from an expected inventory record.

RaDetail represents detailed information about items on an expected inventory record, such as the date the inventory is expected, fulfillment center ID, and quantity ordered.

For more information about inventory receipts, see Chapter 29, “Inventory assets,” on page 265.

Creating vendor assets

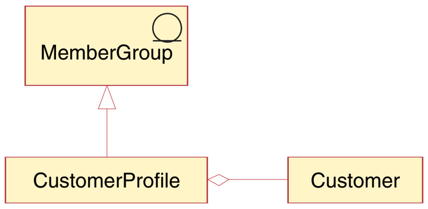
Vendor assets can be created using the WebSphere Commerce Accelerator. See the topic *"Creating a vendor"* in the WebSphere Commerce Production online help for instructions.

Chapter 32. Customer profiles

Customer profiles help organize your marketing efforts by grouping the targets of your marketing messages. Customer profiles are typically created by either a Merchant using the WebSphere Commerce Accelerator.

Understanding customer profiles in WebSphere Commerce

The following diagram illustrates the customer profile 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 97. For more information on the conventions used in this diagram, see Appendix A, "UML legend," on page 437.

A customer profile incorporates registration information, demographics, address information, customer culture, purchase history, and other miscellaneous attributes which define a dynamic group of customers or accounts. Customer profiles serve as targets for advertising, promotions, suggestive selling, discounts, and e-mail activities. You must create customer profiles before creating campaigns. Profiles are considered dynamic because customers belong to them based on their personal data, and purchase history, both of which may change. For example, you might create profiles based on a customer's registration status. If you create a profile that requires customers to be registered to qualify, an unregistered customer will be excluded. If that same customer registers at a later date, they would then become a member of that target profile, and would continue to be a member until the profile is deleted.

Customer profiles also support static criteria. You can explicitly include or exclude particular customers or accounts, which overrides any defined dynamic criteria. In this way for example, you can include a customer in a profile that they would otherwise not match, or exclude an account from a profile that it would otherwise match. In other words, if both static and dynamic criteria have been defined in the same customer profile, the dynamic criteria will be evaluated first, then the static criteria.

Furthermore, the customers or accounts that you can explicitly include or exclude can be based on the data mining results from WebSphere Commerce Analyzer. WebSphere Commerce integrates the advanced analytics from WebSphere Commerce Analyzer, allowing Merchants to easily create an explicit customer profile based on the segments which WebSphere Commerce Analyzer generates.

Part 7. Adding access control to your store

Chapter 33. 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 in WebSphere Commerce, see Chapter 4, “Access control in WebSphere Commerce,” on page 35. For greater detail on the access control model, see the *WebSphere Commerce Security Guide*.

Understanding access control in WebSphere Commerce

Access control in stores

All stores created in WebSphere Commerce are subject to the default access control policies that are subscribed by the organization that owns the store or inherited from that organization’s ancestors.

By default, the Root Organization subscribes to the management and administration policy group. If you create your store based on one of the sample stores provided with WebSphere Commerce, you will create store specific access control policies and policy groups that are owned and subscribed by the organization that owns the store. In addition to the store specific policy group, the owning organization may also subscribe to the management and administration, common shopping and B2C or B2B policy groups, depending on the nature of your store. For more information on subscribing policy groups, see the *WebSphere Commerce Security Guide*.

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 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 and policy group subscriptions 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 access control policy groups subscribed by the organization.

Access control in the samples stores

All of the sample stores contain the high level access control policy files, which define access control policies and policy groups created specifically for the stores. These access control data are owned by the organization that owns the store.

The high level access control policy files for the sample stores are as follows:

- Consumer direct

– FashionFlowAccessControl.xml

- Business B2B direct
  – ToolTechAccessControl.xml
- Business Demand chain
  – CommercePlazaAccessControl.xml (for the channel hub)
  – ResellerStoreFrontAssetStoreAccessControl.xml
- Business Supply chain
  – SupplierHubAccessControl.xml
  – SupplierAssetStoreAccessControl.xml
- Business Hosting
  – CommerceHostingHubAccessControl.xml
  – HostedStoreFrontAssetStoreAccessControl.xml
  – StoreDirectoryAccessControl.xml

These files are located in the following directory:

- *WC_installdir* /samples/stores/*businessmodel*

Note: Express The Express Store high level access control file is located in the Express Store store archve (ExpressStore.sar) . The high level access control file is called `AccessControl.xml`.

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 `ToolTechAccessControl.xml` file.

Defining actions: The first section of the Business `ToolTechAccessControl.xml` file defines the new actions in the store, which are not covered by bootstrap 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.

Defining action groups: The second 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 ToolTechRegisteredCustomerViews, 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 need to be redefined in the Action list discussed in "Defining actions" on page 286.

```
<!-- [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. The action group name is defined in the ACACTGRP table.
- `OwnerID` is the owner of the action group. The Root Organization is typically the owner of the action group. If any other organization is used, the orgentity id of that organization is used.
- `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 286. This action to action group relationship is stored in the ACACTACTGP table.

Defining policies: The next section defines the new policies used in the store.

```
<!-- [Start of Policy definitions] -->

 <!-- AllUsers for ToolTech can execute ToolTechAllUsersViews -->


<Policy Name="AllUsersForToolTechExecuteToolTechAllUsersViews"
   OwnerID="&seller_b2b_mbr_id;"
   UserGroup="AllUsers"
   UserGroupOwner="RootOrganization"
   ActionGroupName="ToolTechAllUsersViews"
   ResourceGroupName="ViewCommandResourceGroup"
   PolicyType="groupableStandard">
    </Policy>
 <!-- RegisteredApprovedUsers for ToolTech can execute
ToolTechRegisteredApprovedUsersViews -->

  <Policy Name="RegisteredCustomersForOrgForTool
TechExecuteToolTechRegisteredCustomerViews"
```

```
   OwnerID="&seller_b2b_mbr_id;"
   UserGroup="RegisteredCustomersForOrg"
   UserGroupOwner="RootOrganization"
   ActionGroupName="ToolTechRegisteredCustomerViews"
   ResourceGroupName="ViewCommandResourceGroup"
   PolicyType="groupableTemplate">
    </Policy>

 <!-- [End of of Policy definitions] -->
```

where

- `Policy Name` is the name of the policy being defined. The policy is defined in the ACPOLICY table.
- `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.In this example, the resource group, ViewCommandResourceGroup, is already defined in the bootstrap data. Hence, it is not required to be redefined in this high level xml file.
- `PolicyType` is the type of the policy. A policy can be either a groupableStandard or groupableTemplate type. For more information on policy types, see the *WebSphere Commerce Security Guide*.

Defining policy groups: The final section defines the specific policy group for the store.

```
<PolicyGroup Name="ToolTechPolicyGroup" OwnerID="&seller_b2b_mbr_id;">
 <PolicyGroupPolicy Name="AllUsersForToolTechExecuteToolTechAllUsersViews"/>
 <PolicyGroupPolicy Name="RegisteredCustomersForOrgForToolTechExecuteTool
TechRegisteredCustomerViews"/>
 <PolicyGroupSubscription OrganizationID="&seller_b2b_orgentity_id;"/>
</PolicyGroup
```

where

- `PolicyGroup Name` is the name of the policy group. It is defined in the ACPOLGRP table.
- `OwnerID` is the owner of the policy group. In this example, the owner of the policy group is the organization that owns the store.
- `PolicyGroupPolicy Name` is the name of the policy that belongs to this group. This policy to policy group relationship is stored in the ACPOLGPPOL table.
- `PolicyGroupSubscription OrganizationID` is the organization that subscribes this policy group. This subscription is stored in the ACPLGPSUBS table.

For more information on using XML files for access control, see the ″Customizing access control policies using XML″ chapter in the *WebSphere Commerce Security Guide*.

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 *WebSphere Commerce Security Guide*. Before continuing with the next steps outlined in the this guide, ensure you review the *WebSphere Commerce Security 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. 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. For detailed instructions for both scenarios, see "Creating or editing access control in your store."

Creating or editing 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 high level access control XML files and then transform and load them.

To create or edit 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` . These files are located in the following directory:
   - `*WC_installdir*/samples/stores/*businessmodel*`

   For the syntax of the high level access control xml files, see the DTD file that are referred to by the *samplestorename*`AccessPolicies.xml`. The DTD files are located in the following directory:
   - `*WC_installdir*/xml/policies/dtd`

   If you are changing a store based on a sample, edit these files. If you are adding access control to a store not based on the sample, create a new high level access control file for your store by copying one of the sample store files. After copying the file, just rename it and edit as appropriate. For more information on adding access control to your store, see the *WebSphere Commerce Security Guide*.
2. Copy the appropriate *samplestorename*`AccessPolicies.xml` file to the following directory:
   - `*WC_installdir*/xml/policies/xml`
3. Replace the placeholders identified in the following chart with the actual values. To find the values, do the following steps:
   a. Find the value for the distinguished name (DN) by using the following query. Use the DN values supplied in the chart below in the query:
      - `select orgentity_id from orgentity where dn= <the DN of the organization>`

      Example: In the `CommercePlazaAccessControl.xml` file, replace `&channel_mbr_id;` and `&channel_orgentity_id;` with the value you find using the above query, when the DN value is ou=Reseller Hub Organization, o=Demand Chain Management Organization, o=Root Organization.

Table 12.

File name	**Replace the following placeholders with the actual value**	**DN value**
Business CommercePlazaAccess Control.xml	&channel_mbr_id; &channel_orgentity_id;	ou=Reseller Hub Organization,o=Demand Chain Management Organization,o=Root Organization
	&proxy_orgentity_id;	ou=Distributor Proxy Organization,o=Demand Chain Management Organization,o=Root Organization
Business ResellerStoreFrontAsset StoreAccessControl.xml	&profile_mbr_id;	ou=Asset Store Organization,o=Demand Chain Management Organization,o=Root Organization
Business SupplierAssetStoreAccess Control.xml	&profile_orgentity_id &;profile_mbr_id;	ou=Asset Store Organization, o=Supply Chain Management Organization,o=Root Organization
Business SupplierHubAccessControl.xml	&channel_orgentity_id; &channel_orgentity_id;	ou=Supplier Hub Organization,o=Supply Chain Management Organization,o=Root Organization
FashionFlowAccessControl.xml	&seller_b2c_mbr_id; &seller_b2c_orgentity_id;	"ou=B2C,o=Seller Organization,o=Root Organization"
Business ToolTechAccessControl .xml	&seller_b2b_mbr_id; seller_b2b_orgentity_id;	ou=B2B,o=Seller Organization,o=Root Organization
CommerceHostingHubl AccessControl.xml	&channel_mbr_id; &channel_orgentity_id;	ou=Hosting Hub Organization,o=Hosting Organization,o=Root Organization
HostedStoreFrontAsset StoreAccessControl.xml	&profile_mbr_id;	ou=Asset Store Organization,o=Hosting Organization,o=Root Organization
StoreDirectoryAccess Control.xml	&public_mbr_id; &public_orgentity_id;	ou=Store Directory Organization,o=Hosting Organization,o=Root Organization

4. Add the appropriate access control information to the file. For more information see "Access control in the samples stores" on page 285 and the *WebSphere Commerce Security Guide*.
5. Run the `acpload` command to transform the file,*`samplestorename`*`AccessPolicies.xml`, and load it into the database.

Note: The database user ID must have the following permissions in order to execute the acpload command:

- read/write/execute authority to the directories, subdirectories and files of *WC_installdir*/xml/policies, *WC_installdir*/logs.
- read/execute authority to the *WC_installdir*/bin directory and its files.

a. At a command prompt, change the directory to the following:
   - `WC_installdir/bin`

b. Run the acpload command file.
   - 2000 **Syntax**: acpload.cmd *databasename database user database user password Policies XML file* [schema name] **Example**: acpload mall dbuser dbusrpwd ChannelHubAccessControl.xml
   - 400 AIX Solaris Linux **Syntax**: acpload.sh *databasename database user database user password Policies XML file* [schema name] **Example**: acpload.sh mall dbuser dbusrpwd ChannelHubAccessControl.xml

c. Check the following log file to ensure that the access control data has been loaded successfully:
   - *WC_installdir*/logs/acpload.log
   - 400 *WC_userdir*/instances/acpload.log

For more information about the use of @ and & see Appendix B, “Creating your data,” on page 439.

Part 8. Globalizing your store

Chapter 34. Globalization

Supporting globalization

WebSphere Commerce is translated to the following ten languages:

- United States English
- French
- German
- Italian
- Spanish
- Brazilian Portuguese
- Simplified Chinese
- Traditional Chinese
- Korean
- Japanese

This includes the software, its documentation, user interfaces, and the samples. You can add support for other languages. You can translate many of the features of your site, such as product descriptions, messages and text on the pages. This applies to pages on your store as well as on the browser based WebSphere Commerce tools, such as the WebSphere Commerce Accelerator and the Administration Console.

WebSphere Commerce includes several features that allow you to create a site that can be tailored to fit the needs of an international or culturally diverse customer base. Through the use of Java technology and a flexible database schema, some of the cultural characteristics of your site that you can vary, depending on location or customer preference, including:

Language
A store may be viewed in more than one language. For example, you may wish to allow users to choose the language in which they would like to view your site, or you may want to automatically select a default language, depending on the location of the store.

Currency
A store may display prices in more than one currency. In WebSphere Commerce, the selection of a specific currency is not related to the selection of a particular language.

Cultural format
Data may be displayed in various customizable formats. Customers from different cultures may have different expectations about how certain information should be displayed. For example, a decimal number may be indicated by using either a comma or a period, depending on the language, or country or region of the customer.

Address
Addresses may be entered, stored, and displayed in various formats, to conform to different international standards.

Taxation
Taxation rules may be defined for different jurisdictions. This includes rules for sales and other business taxes.

Shipping
Shipping rules and carriers may be defined for different jurisdictions.

Payment method
Payment methods may be defined for different jurisdictions.

Prices The price can be set in one currency and a conversion factor can convert from this currency to other currencies, or the price can be set per currency.

Online catalog content
Product and category descriptions, attributes, and images may be varied across locations. WebSphere Commerce allows a merchant to manage online catalog content for selectable display format by defining each format within the language table. You can also maintain a master product catalog that can be shared by a number of stores.

The WebSphere Commerce database has been designed to allow you the flexibility to create and maintain internationally recognizable data by using Unicode UTF-8 encoding, or 400 UCS-2, which can be converted into most international encoding formats when sent to the Web browser. For a working sample of a site globalized for many countries, see any of the sample stores.

The Administration Console and WebSphere Commerce Accelerator support globalization usage. These tools display in any of the ten national languages supported by WebSphere Commerce. They use a programming model that allows for additional languages to be added without affecting the general functioning or look-and-feel of the pages.

You can translate the tools into other languages, just as you would translate a store or other Web site. To translate the tools modify the appropriate properties files located under the following directory:
WAS_userdir/installedApps/*cell_name*/*WC_instance_name*.ear/properties /com/ibm/commerce/tools/

Sample stores

The sample stores and sites provides a foundation to create your store. All the sample stores demonstrate how you can create and maintain a globalized site.

The sample stores allow customers to select the language in which they would like to view the site. In FashionFlow and Commerce Plaza, you select the desired display format from a list. The list is displayed in a drop-down list on the left-hand frame throughout the site. Customers can navigate through the site, viewing it in the language of their choice.

Business In ToolTech, you switch languages by going to the personal information page to select your preferred language.

Business In Commerce Hosting Hub, and Commerce Supplier Hub, on the logon page, there is a drop-down list available from which you select your language.

The sample stores use the single template for all stores and languages programming model. Each supported language has its own property file, which contains the translated text and messages for that language. All store archives contain: publishNLS.properties, publishNLS_en_US.properties. These are just used in the publish wizard.

For FashionFlow and hosted stores based on FashionFlow:

- infashiontext_*locale*.properties
- infashiontext_dynamic_*locale*.properties
- infashiontext_dynamic_labels_*locale*.properties (only used for the Change pages GUI)
- AuctionSample_*locale*.properties

Business For ToolTech it is:

- tooltechtext_*locale*.properties

Business For Commerce Plaza, it is:

- pcdmarket_*locale*.properties

Business For Commerce Supplier Hub, it is:

- tooltechtext_*locale*.properties

Business For Commerce Hosting Hub, it is:

- b2cHostingChannel_*locale*.properties

Business For the Store Directory in the Hosting sample it is:

- b2cHostingPublic_*locale*.properties

Within the property file, a number of elements of the page are translated:

Text Textual page content.

Labels Form field labels.

Messages
Error, status, and confirmation messages.

Alternate text
For images, Java applets, and other embedded objects.

Determining the character set
The character set in which the text is displayed in a browser is defined in the property file using the ENCODESTATEMENT property. For example, the infashiontext_en_US.properties file contains the statement: ENCODESTATEMENT = text/html; charset=ISO_8859-1 Because encoding is specified within the property file instead of in the JSP template, the character set can be different for each language. The character-encoding of the generated JSP page is set using the following statement in the JSP template: <%response.setContentType(infashiontext.getString ("ENCODESTATEMENT")); %> At run time, each requested JavaServer Page includes the file EnvironmentSetup.jsp. Within this file, the command context is retrieved, and from that, the locale is used to retrieve the infashiontext Properties java object, which obtains its values from the infashiontext_locale.properties file in the appropriate locale-specific directory. The template then has access to each of the properties as needed, through the use of the getString() method of the ResourceBundle object.

Note: *This is for the value chain hosted stores created with the Store Creation Wizard only.* When a seller creates a hosted store, this is not a *regular* store publish, and additional store language assets beyond the store default language are not carried over into the hosted store. So, if a seller adds a supported language to a store, the store assets for that language are not available. If a supported language is going to be added to a hosted store, ensure that the translated assets (properties files) are available to the store or the store pages will not function correctly.

Displaying translated images

The locale and language are retrieved at run time to determine the correct folder in which to look for the image file. The template might look for the file FashionFlow/*language_Locale*/images/go_button.gif, where *language_Locale* is replaced by the display format from the command context. For example, the resulting page will display the image: FashionFlow/en_US/images/go_button.gif or FashionFlow/jp_JA/images/go_button.gif .

Displaying catalog content

The catalog contains multiple translations, one for each supported locale. At run time, the command context is sent through a data bean to determine which translation to retrieve from the database and display on the page.

Resource bundles and property files

Resource bundles and property files allow you to maintain collections of java objects that are locale-specific for your JavaServer Pages. When the page needs a locale-specific resource, such as a form field label, a graphical user interface message, or a value for a drop-down menu, the page can load it from the resource bundle or property file that is appropriate for the selected locale, allowing the customer to view the page in their own language. In this way, you can create JSP templates that are largely independent of the customer's locale, isolating all of the locale-specific information in the resource bundles or property files.

Although resource bundles and property files perform similar functions, there are some differences in the manner in which they are processed. The table below shows some of the more important differences between resource bundles and property files:

Table 13. ListResource bundles and property files

Property files	**ListResource bundles**
Text files	Compiled format
Slight performance degradation	Slight performance boost
If a property file is changed, WebSphere Application Server must be restarted to see the changes.	If a resource bundle is changed, it must be recompiled and WebSphere Application Server must be restarted to see the changes.
Language and locale dependent. One file should exist for each locale. You need to run native2ascii on non-ISO-8859-1 characters.	Language and locale dependent. One file should exist for each locale. You need to run native2ascii on non-ISO-8859-1 characters.

For an example of how property files can be used in a site globalized for many countries, refer to the sample stores. For more information on these topics, visit the Sun Microsystems Java Web site.

Data storage and transfer

A single store can display pages in multiple languages, even when the languages use different character sets. To accomplish this, data is stored in the WebSphere Commerce database in a universal format that can be applied to a wide number of languages. Since not all Web browsers support the same character sets, when the data is requested by a JavaServer Page, it is converted into an appropriate character set.

The following describes how data travels from the database to the browser:

1. Text data is stored in the WebSphere Commerce database using Unicode UTF-8 encoding or 400 UCS-2.
2. JDBC drivers load the data from the database, converting it from UTF-8 to Java's native 16-bit Unicode encoding.
3. The JSP files output the data using the Java 16-bit encoding.
4. WebSphere Application Server converts the JSP file output from 16-bit Unicode to the target encoding. The encoding can either be specified in the JSP file or in a property file. For example, to specify Shift-JIS encoding for a Japanese page, you could do the following:
   - JSP file <%@ page contentType="text/html; charset=Shift-JIS"%> .
   - Property file ENCODESTATEMENT = text/html; charset=Shift-JIS The character-encoding of the generated JSP file is set using the following statement in the JSP template: <%response.setContentType(fashionflowtext.getString ("ENCODESTATEMENT")); %>

   Since not all browsers can understand every encoding scheme, we recommend that you specify the more well known encoding schemes, such as UTF-8 and Shift-JIS.
5. The converted data is sent back to the browser.
6. The browser interprets the HTTP reply based on the encoding specified in the header.

The following describes how data travels from the browser to the database:

1. Data is entered into the browser. Multilingual data can be entered using an input method.
2. WebSphere Commerce converts the data coming from the browser into Java 16-bit encoding using the setCharacterEncoding() method. Each LANGUAGE_ID in the LANGUAGE table is mapped to an encoding value using the ENCODING column. This value is used to interpret the data coming from the browser.
3. The data is sent to the database where it is converted from Java 16-bit to UTF-8 encoding, which is how it is stored in the database.

Display formats

Display formats allow a single store to sell to a globalized, multilingual customer base. Each display format can be identified by three factors: a language, a region, and a variant you can define. You can design your site to display different content to groups that differ on any of these factors. For example, you could use language and locale to have a separate format for US English and Canadian English. These display formats could have the same text, but different currencies and units of measurement. You could add a display format for Canadian French. This display format could display the same currency and units of measurement as Canadian English, but the text would be in Canadian French. You could even use the third factor to have a separate format for specific audiences within a culture, such as teens, scientists, or technical professionals, tailoring your site to suit their expectations.

A customer can either choose the format in which they would like to view the site, or you can set a default value for them. Information regarding the format is passed through a URL parameter when you want to change the language. For example, if you pass in langId=-2, the session will set the current language to French. The langID is stored in the session. When the customer requests a page, the display format determines the Web assets and catalog information to retrieve.

Creating a new display format

To create a new display format, do the following:

1. Run the following command: select * from language This command returns the language IDs in use by currently available display formats. Choose the next available ID_VALUE.
2. Run the following command: insert into language (LANGUAGE_ID, ENCODING, LOCALENAME, LANGUAGE, COUNTRY) values (ID_Value, ENCODING_VALUE, ’x’, ’y’, ’z’) where:

   ID_VALUE
   The value you selected in step 2.

   LANGUAGE_ID (Required)
   An identifier to uniquely identify the display format.

   ENCODING_VALUE (Required)
   The character encoding value that the browser should use to display the page for this language. This should be the same encoding value used in your property files. ENCODESTATEMENT = text/html; charset=[ENCODING_VALUE]. A list of encoding values supported by the Sun JDK is available from the Sun Java site at www.java.sun.com.

   LOCALENAME (Required)
   A java locale used to represent a political, geographical, or cultural region that has a distinct language and customs for formatting. The localename is the two-letter ISO 639 language code, followed by the two-letter ISO 3166 country code, separated by an underscore.

   LANGUAGE (Optional)
   The name of the language.

   COUNTRY (Optional)
   The country or region for the display format.

   VARIANT (Optional)
   The variant column is an extra column that allows you to describe a subgroup within a particular culture, such as teen, technical, other any other classification.

   For example, to add a display format for Italian as spoken in the United States, you could execute the statement: insert into language (LANGUAGE_ID, ENCODING, LOCALENAME, LANGUAGE, COUNTRY) values (’333’, ’ISO8859-1’, ’it_US’, ’Italian’, ’United States’) You may also want to specify an alternative language, refer to *Example of creating a new display format*.
3. Add an entry to the LANGUAGEDS table. For an example, refer to *Example of creating a new display format*.
4. Add an entry to the LANGPAIR table. For an example, refer to *Example of creating a new display format*.
5. Add a language to a store. For details on how to add a language to your store, see “Adding a language to a store” on page 307

Example of creating a new display format

The following example shows how you could create a display format to view the FashionFlow sample store pages in Thai.

1. Translate the infashiontext_locale.properties file to Thai.
2. Ensure that the encoding statement in the properties file references a character set that target browsers will support. For Thai the encode statement is as follows: ENCODESTATEMENT = text/html; charset=MS874
3. Save the file as infashiontext_th_TH.properties
4. Open a DB2 command window.
5. Run the following command: select * from language This command returns the language IDs in use by currently available display formats. Choose the next available ID_VALUE. In this example, the ID_VALUE is for Thai is 3.
6. Run the following command: insert into language (LANGUAGE_ID, ENCODING, LOCALENAME, LANGUAGE, COUNTRY, MIMECHARSET) values (ID_Value, ENCODING_VALUE, ’w’, ’x’, ’y’, ’z’) Using the following values: insert into language (LANGUAGE_ID, ENCODING, LOCALENAME, LANGUAGE, COUNTRY) values (’3’, ’MS874’, ’th_TH’, ’Thai’, ’Thailand’, ’MS874’) where:

   ID_VALUE
   The value you selected in step 2.

   LANGUAGE_ID (Required)
   An identifier to uniquely identify the display format.

   ENCODING_VALUE (Required)
   The character encoding value that the browser should use to display the page for this language. A list of encoding values supported by the Sun JDK is available from the Sun Java site at www.java.sun.com. The encoding value must be supported by the Sun JDK.

   LOCALENAME (Required)
   A java locale used to represent a political, geographical, or cultural region that has a distinct language and customs for formatting. The localename is the two-letter ISO 639 language code, followed by the two-letter ISO 3166 country code, separated by an underscore. For ISO language codes refer to the following International Standards Organization Web site at www.iso.ch.

   LANGUAGE (Optional)
   The name of the language.

   COUNTRY (Optional)
   The country or region for the display format.

   MIMECHARSET (Optional)
   The charset used in MIME messaging.

   VARIANT (Optional)
   The variant column is an extra column that allows you to describe a subgroup within a particular culture, such as teen, technical, other any other classification.

7. Add an entry to the LANGUAGEDS table:
   languageds language_id=-1, description=`your_language_name_in_English`, language_id_desc=-11.
   For example: ″languageds language_id=-1, description=`French`, language_id_desc=-11, languageds language_id=-11, description=`Your_language_name_in_your_own_language`, language_id_desc=-11,

desc=-11.
For example: ″languageds language_id=-11, description=Francais, language_id_desc=-11.

8. Create an alternative language for Thai used if the data requested in Thai does not exist. This is useful if not all the data in the database is translated into the new language. To create an alternative language run the following command: insert into langpair(LANGUAGE_ID, LANGUAGE_ID_ALT, SEQUENCE, STOREENT_ID) values (ID_Value, ID_Value_ALT, ′x′, ′y′) Using the following values: insert into langpair(LANGUAGE_ID, LANGUAGE_ID_ALT, SEQUENCE , STOREENT_ID) values (′3′,′-1′, ′1′ ′12345′) where LANGUAGE_ID The requested Language. LANGUAGE_ID_ALT The alternative Language. SEQUENCE When the requested Language is supported as specified in the STORELANG table, but information is not available in that Language, each alternative Language is tried in ascending order of SEQUENCE. A store may override the SEQUENCE specified for its StoreGroup. STOREENT_ID The StoreEntity this relationship belongs to. The alternative Language relationships for a Store include the alternative Language relationships for its StoreGroup. The above insert statement will assign English (language id = -1) as the first alternative language to try for store with id ′12345′ in the event that no Thai data is found.
9. Convert any properties file from native to ascii:
Copy `infashiontext_th_TH.properties` to a temporary directory (for example, /tmp). Run the following command: *JDK_dir*/bin/native2ascii -encoding TIS620 /tmp/infashiontext_th_TH.properties /tmp/infashiontext_th_TH_new.properties. Copy this: /tmp/infashiontext_th_TH_new.properties to this directory: /*WAS_userdir*/installedApps/*cell_name*/WC_instance.ear/Stores.war/WEB-INF/classes/*storeDir*/infashiontext_th_TH.properties. Where *JDK_dir* is the path to your JDK.

Multilingual data entry

You can display multiple languages simultaneously in a browser, if the browser supports Unicode. Both Netscape Navigator and Internet Explorer Version 4 or higher support Unicode display. However, your operating system may not have the characters you need to enter text in some languages. To do this, you may need to use an input method. An input method is a a software component that converts key presses into text input which can not be typed directly. Input methods are normally used to input text for languages which have more characters than can fit on a standard keyboard, such as Japanese, Chinese and Korean, Thai and Hindi. A common input method tool is Microsoft® Global Input Method Editor which is available from the Microsoft Web site. The Global IME is appropriate for both customers to enter data on shopping pages, and for Administrators to enter data in the WebSphere Commerce Accelerator and Administration Console.

If you choose to not use an input method, you can also enter data in different languages provided that you have a number of machines setup, each using a different operating system language with a properly configured browser. A browser will automatically support the language that is native to the machine on which it is installed. For example, to enter Japanese and German data, you can setup two machines, one using a German operating system and one using a Japanese operating system, each with a browser capable of displaying data from that operating system. For more information on this issue, consult the documentation for your operating system or Web browser.

Unicode: WebSphere Commerce text data is encoded using the Unicode character set. Unicode can display characters used in the major languages, including

European, Middle Eastern and Asian languages. In WebSphere Commerce, the Unicode UTF-8 standard is used to store data in multiple languages in the same database instance. Although customers do not need to have a Unicode-enabled browser to view sites driven by WebSphere Commerce, administrators may need one if they want to view their site in more than one language on the same machine. If you want to view your site in a language other than English, you need a Unicode enabled browser. For more information on Unicode, visit the Unicode Web site.

Creating a globalized store

To create a globalized store, do the following

1. Create a store.
2. Manage your template.
3. Add a language to your store.
4. Create a globalized catalog.
5. Manage your globalized assets.
6. Translate your property files

Creating a store

You can either create a store by publishing one of the sample store archives and editing the resulting store, or you can create the storefront, business logic, or data assets separately.

- Storefront: The external portion of your store, or the portion that displays to your customers, is known as the storefront. The storefront is comprised of Web assets such as HTML pages, JSP files, style sheets, images, graphics and other multimedia file types. For more information, see Part 4, “Developing your storefront,” on page 73.
- Business logic: The portion of your store that processes customer requests, including the commands, customized code, is known as the business logic. For more detailed information on creating business logic or customized code see the *WebSphere Commerce Programming Guide and Tutorials*.
- 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. Your store must also have methods for processing and collecting payment. The concepts and tasks involved in creating store data are discussed in Part 6, “Developing your store data,” on page 107.

For more information on publishing a sample store archive, see the WebSphere Commerce Production online help, topic ’Publishing a store archive″.

Managing your template for a globalized site

To manage static pages and dynamic templates in a globalized site, it is necessary to store files in a directory structure that allows for the quick and easy identification of the files and which locale they belong to.

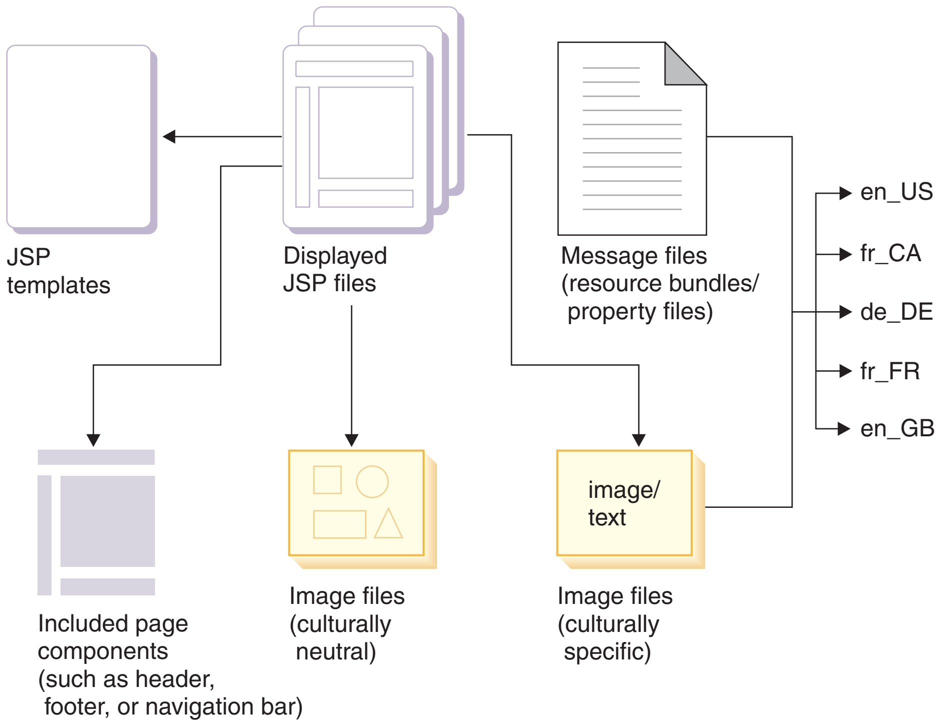
The file directory path is constructed based on the WebSphere Commerce instance, the Business store path contained within the store profile and also the registered file path. When you create a globalized site, you create multiple stores, each one representing a supported shipping jurisdiction of the site, and each one having a list of supported languages. Since the template files influence the look-and-feel of a site, the template files are stored under locale-specific directories so that they can be selected similar to the manner in which resource bundles are selected, using a locale value. When the system selects a template to use for a particular language format, the locale is used to determine the language format that will be used to determine the directory from which the file is retrieved.

There are three models for template storage in a globalized environment:

Table 14. Template storage in a globalized environment

	One template for all stores and languages	**One template per language**	**One template per store**
Customization	For most stores provides sufficient levels of customization between each store and each store language format.	Allows the maximum level of customization between each store and each language format.	Some level of customization between each store.
Look and feel of pages	Pages will look similar.	Pages can be very different.	Pages have the same general layout.
Maintenance	Allows for easy site-wide page-design changes since only one template needs to be changed. For most globalized sites, this model provides optimal levels of maintainability and scalability.	Must manage multiple copies of each template. Changes that affect all stores, or all language formats have to be made to every template.	Site-wide changes to the look of a JSP file will have to be made across multiple templates.
When to use	Use when the look and feel for each store and each language is very similar.	Use when page look and feel and content between languages are very different. In this case, there is not much that can be shared across languages and it is easier to develop separate pages for each language.	Use when stores differ in look and feel significantly, but the store's look and feel remains relatively the same regardless of the language.
When not to use	Do not use if site is meant to look very different between stores and languages	Do not use if pages are very similar between stores and between language formats.	Do not use if stores look and feel are very similar.
Property files	Required. Each supported language also has its own property file that is included when the page is generated.	Not required. Each store and locale combination has its own JavaServer Page template.	Required. To allow for template sharing between each language format.
Shopping flow	The shopping flow between languages and stores remains the same.	The shopping flow can change significantly between languages.	The shopping flow between languages and stores remains the same.

One template for all stores and languages programming model


JSP templates
Displayed JSP files
Message files (resource bundles/ property files)
en_US
fr_CA
de_DE
fr_FR
en_GB
image/ text
Included page components (such as header, footer, or navigation bar)
Image files (culturally neutral)
Image files (culturally specific)

In the one template for all stores and languages programming model, each page consists of a single JavaServer Page template, containing a basic page layout and culturally neutral data and images. This template is combined at run time with culturally sensitive components, based on the display format selected by the customer. Changes to the design of a page only need to be made once, regardless of the number of cultures supported. Adding or removing languages or cultures is simple, since the culturally sensitive is separate from other features of the page.

The following table shows how files may be organized. Note that *webapp* refers to the root directory of the site or application. Within that directory you might have a common directory and a directory for each display format, or language-locale combination supported. The exact structure is up to you. At run time, the template uses the language and locale information from the command context, and uses it to determine the appropriate folder from which to retrieve the property file, image files, and any other culturally specific content. For example, if the command context indicates an en_US display format, it will use the site_root/en_US/sensitivetext.properties file and will retrieve images from the site_root/en_US/images/ directory.

Table 15. One template for all stores and languages programming model

Templates	/webapp/common/web/template/template.jsp The same template is used for all display formats.
Included page components	/webapp/common/web/template/header.jsp /webapp/common/web/template/footer.jsp The common page components are in this directory.
Culturally neutral image files	/webapp/common/web/images/image.gif Images are in a common directory and used for all display formats.

Table 15. One template for all stores and languages programming model (continued)

Property files (select either of the following methods to store property files)	/webapp/language_LocaleA/web/sensitivetext.properties /webapp/language_LocaleB/web/sensitivetext.properties Each display format has a separate property file. The property files for different display formats have the same base name, with the locale suffix xx_XX appended to the name before the file extension. They are located in the same directory. The directory name is based on the *Language_Locale* combination as it appears in the LOCALENAME column of the LANGUAGE table. For an example of this method, see a sample store. **OR** you can use this method: /webapp/properties/sensitivetext_Language_LocaleA.properties /webapp/properties/sensitivetext_Language_LocaleB.properties Property files are stored in a single directory, but have locale-specific file names.
Culturally specific image files	/webapp/language_LocaleA/web/images/image.gif /webapp/language_LocaleB/web/images/image.gif A separate translated image is stored for each display format. The files have the same name, but they located in different directories, corresponding to the name of the display format to which they apply. The *Language_Locale* combination represents the display format, as it appears in the LOCALENAME column of the LANGUAGE table.

One template for all stores and languages directory structure

This model is suitable where the look and feel for each store and each language are very similar. You only need to maintain one set of JSP templates, but you must manage a series of property files. This method is the template management model to maintain and it allows for site-wide page-design changes since only one template needs to be changed.

For example, if you have two store locations, each displaying US English and Canadian French, you might organize your JSP templates as follows:
/webapp/common/web/template/abc.jsp

The path for the property files for this JSP template would be stored as follows:
/webapp/common/web/properties/en_US/abc.properties
/webapp/common/web/properties/fr_CA/abc.properties
In this case, when registering the JSP files, only the file type needs to be included in the file registry. Using this method, only one set of JSP files needs to be registered for all stores and all locales. Here, the property files must be stored separately because they contain culturally sensitive information, whereas the template itself, which is completely neutral, is stored in a common directory. For a working example of this template management strategy, refer to a sample store.

One template per language directory structure

To use this model, a separate directory must be created within each store's template directory for each language supported by that store. A different template must be stored within each of these language-dependent directories. No property files are required in this model, since each store and locale combination has its own JavaServer Page template.

For example, if you have two store locations, each displaying United States English and Canadian French, you might organize your JSP templates as follows:
/webapp/StoreA/web/template/en_US/abc.jsp
/webapp/StoreA/web/template/fr_CA/abc.jsp

/webapp/StoreB/web/template/en_US/abc.jsp
/webapp/StoreB/web/template/fr_CA/abc.jsp In this case, when registering the JSP template, the locale and file type will have to be included in the file registry. Each store and each locale must have a complete set of registered templates .

One template for each store directory structure

JSP templates for this model are shared within a store, but are exclusive to a single store. JSP-include files are required in this model to allow for template sharing between each language format.

For example, if you have two store locations, each displaying United States English and Canadian French, you might organize your JSP templates as follows: /webapp/StoreA/web/template/abc.jsp /webapp/StoreB/web/template/abc.jsp

The path for property files within this template management model would resemble this example: /webapp/StoreA/web/properties/en_US/abc.properties
/webapp/StoreA/web/properties/fr_CA/abc.properties
/webapp/StoreB/web/properties/en_US/abc.properties
/webapp/StoreB/web/properties/fr_CA/abc.properties

When registering the JSP templates for this model, only the file type needs to be included in the file registry. Each store will have to register its own complete list of the JSP files.

Adding a language to a store

To add support for a new language to an existing store, do the following:

1. Ensure that the language is available to your site. For a list of the ten national languages supported, refer to multilingual support. If the language is available go to step 3, if not, go to the next step.
2. Create a new display format for the language. Use the Store Profile notebook to add the language to the list of those supported by the store.
3. Copy the national languages file, for example, for FashionFlow it would be infashiontext_locale.properties to this location:
   AppServer/installedApps/*host*/WC_demo1.ear/Stores.war/WEB-INF/classes/storeDir
   There are many other XML files that need to be translated and populated using the Loader, such asw tax.xml, store.xml, fulfillment.xml, catalog.xml, businesspolicy.xml, contract.xml, accesscontrol.xml, shipping.xml.

Creating a globalized online catalog

To create a flexible online catalog that is suitable for a globalized site, include multiple details about each product, one for each language or culture you want to support. Consider that there are often differences between cultures that go beyond just language, such as the way certain types of data are represented. For example, in some cultures, a decimal number is represented by a comma, whereas in others it is represented using a period.

1. For each language that your store supports you must create a catalog. Select one of the following catalog creation methods:
   - Create a catalog using the Loader package or a catalog tool of your choice.
   - Create your catalog data in XML files and load it into the database using the Loader package, or publish it in a store archive format using the Administration Console. For more information, see Creating a catalog.

- Convert your existing catalog to an XML file format, suitable for use with the Loader, and then load the information into the database. For more information, see Loader package.
- Create your catalog using the sample store catalogs as a base, and then change the information using the Product Management tools (this works for small amounts of data only).

2. For each catalog that you create consider how to present the following types of information.

   Online catalog products
   There are multiple language descriptions for each catalog entry.

   Product descriptions
   Language and phrasing of the descriptions can be varied, highlighting different features to different groups of customers.

   Prices Prices can be varied to reflect tariffs and other shipping expenses, and can be expressed in different currencies.

   Cultural formats
   Dates, names, measurement units, and other data can be formatted to suit cultural expectations.

   Product images
   You may want to display different product images to different customers.

Manage globalization assets

To manage your Web assets, it is recommended that you employ a globalized programming model that uses one JSP template for all stores and languages, including the basic design of each page along with any culturally neutral information. The remaining culturally sensitive text is added to your pages at run time through the use of resource bundles or property files.

Determine which Web assets to translate. This list might include: banners, images, applets, text, messages, and other culturally sensitive content displayed in your pages. Create multiple versions of some of these components, one for each language or culture supported by your site. For an example how assets can be managed in a globalized store, refer to a sample store.

Translate property files

To translate property files, do the following:

1. Open the property file using any text editor.
2. Translate the text in the property file noting the following:
   - Do not translate the keyword. The keyword is the content to the left of the equal sign. Source: lastName.Label=Last Name Translation: lastName.Label=Nom de famille
   - For option attributes, only translate values to the right of the semi-colon (;). Source: title.Options=MR;Mr.|MRS;Mrs.|MS;Ms. Translation: title.Options=MR;M.|MRS;Mme.|MS;Mlle. Source: publishPhone.Options=Y;Yes|N;No Translation: publishPhone.Options=Y;Oui|N;Non
   - Optionally, translate comments, that is, any line that begins with the pound sign (#).
3. Save the property file as text. If you are using a programming model where:

a. Property files have the same name but are stored in locale specific directories
   - Save the file to the correct directory
b. Property files are stored in the same directory but the locale is appended to the name
   - Append the appropriate locale to the file name. The extension must be .properties

4. If the property file contains characters which are non-Latin 1 and non-Unicode, use the the native2ascii converter to convert the data from non-ascii format to Unicode ascii representations. This process will make the data contained within the property file platform independent. The native2ascii converter is in the following directory:

WC_installdir\jdk\bin

400 *WC_installdir*/Java400/jdk13/bin

For additional information about the native2ascii converter refer to the following site: www.java.sun.com

Part 9. Packaging your store

Chapter 35. Packaging a store

If you want to use your store as a sample to be delivered to others, to publish it using the publish utility in the Administration Console or to deploy it on another server or platform, you can package it in the store archive form.

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.
- Property resource bundles: Contains the text for your store pages. If your store supports more than one language, the store archive will contain multiple resource bundles, one per supported language, plus a default resource bundle (which does not include a locale). For example, `AddressText_en_US.properties` and `AddressText.properties`.
- 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 6, “Developing your store data,” on page 107.

  The store data assets in the sample store archives provided with WebSphere Commerce are part of a well-formed, XML file valid for the Loader package. The store data assets 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 portable. For more information, see Part 9, “Packaging your store,” on page 311.

  For more information on the Loader package, see Chapter 37, “Overview of loading store data,” on page 335.

  Note: Store data assets also include contract information, which provides the required information to create a contract. The contract information is not loaded through the Loader package; it provides input to a command that creates contracts.
- Payment assets: Configuration information for WebSphere Commerce Payments. The payment information is not loaded through the Loader package; it provides input to a command that configures WebSphere Commerce Payments.
- Descriptors: XML files that describe the store archive and information on how it should be published. These files include `store-refs.xml`, `ibm-wc-load.xml`, `unpack.xml`, and `ForeignKeys.dtd`.

The files in the sample store archives are grouped into the following structure:

- *Store directory*
  - JSPs files, HTML: grouped into subdirectories by functional areas. For example, ShoppingArea, AuctionArea, CustomerServiceArea. Each of these Areas is then grouped in sections. For example, the ShoppingArea is subdivided into the CatalogSection, the CheckoutSection, the DiscountSection and the ShopcartSection. Each of these sections may also be divided into sections if necessary.
  - Images: The images in the sample store archives are grouped by locale.

- SAR-INF
  - Contains information specific to publishing this store archive, including the following files:
    - `store-refs.xml`: defines the publish parameters to be used with this store archive.
    - properties files: the text used to describe the publish parameters for this store archive. The sample store archives also include locale-specific versions of this properties file.
    - `unpack.xml`: determines which assets in the store archive will be unpacked, how they will be unpacked and where they will be unpacked.

  Note: For more information on the `store-refs.xml` and `unpack.xml` files, see Chapter 36, "Publishing a complete store," on page 321.
- WEB-INF
  - Properties files: The store's properties files are located in the following directory structure in WEB-INF:
    - Classes
      - *Store directory*
  - Store data assets: The store's data assets, in the form of XML are located in the following directory structure in WEB-INF:
    - Stores

      Note: The composite store archive include an additional business model directory.
      - *Store directory*
        - data: If the store contains multiple language, the locale specific XML files are grouped by locale. The data directory also contains the following files needed for publishing the store archive:
          - `ForeignKeys.dtd`: stores the publishing parameter values.These parameters are entity name-value pairs which are referenced by the store data assets XML.
          - `ibm-wc-load.xml`: controls the loading of the data.
          - `store-data-assets.xml`: an XML file that includes all the store data to be loaded.

            Notes:
            1. For more information on the `ForeignKeys.dtd` and `ibm-wc-load.xml` files, see Chapter 36, "Publishing a complete store," on page 321.
            2. Payment configuration information, in the form of an XML file, is also included in the data directory.
  - Files for changing store flow: If the flow of the store can be changed using the store tools in the WebSphere Commerce Accelerator, the necessary files are located in the following directory structure:
    - xml
      - tools
        - stores
          - *Store directory*
            - devtools
              - flow

Creating a store archive

The packaging structure of the store archive is flexible. The instructions given here reflect the structure of the samples provided with WebSphere Commerce, however, you can alter this structure to meet your needs. Your store archive must include a SAR-INF directory that contains a `store-refs.xml` file and an`unpack.xml` file. The paths defined in thes two files must be consistent with the structure of your 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:

   - *WC_installdir*/samplestores/*businessmodel*

   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. Create the following subdirectories:
   - *Store directory* (name of your store)
   - SAR-INF
   - WEB-INF
4. In the *Store directory* directory do the following:
   - Create subdirectories for your JSP files by functional areas. See the sample store archives for an example. Copy your JSP files and any necessary HTML files to these subdirectories.
   - Create a subdirectory for your image files. If your store supports several languages, create subdirectories for language-specific information using locale names. For example, en_US. Copy your image files to these subdirectories.
5. In the SAR-INF directory do the following:
   a. Create a `store-refs.xml` file for your store archive. Using an existing `store-refs.xml` file from a sample store archive as an example, create a `store-refs.xml` file for your store. For more information on the XML specifications, see the descriptor, `store-refs.dtd` in the following directory:
      - *WC_installdir*/xml/sar

      Note: For more information on the `store-refs.xml` file, see Chapter 36, “Publishing a complete store,” on page 321.

   b. (Optional) If you want the store archive to have parameters that a user can select while publishing from the Administration Console, create a properties file that describes these parameters. Save this file in a subdirectory named properties.

   c. Using an existing `unpack.xml` file from a sample store archive as an example, create an `unpack.xml` file for your store. The `unpack.xml` file determines how the store archive will be unpacked. For more information on the XML specifications, see the descriptor, `unpack.dtd` in the following directory:
      - *WC_installdir*/xml/sar

   Note: For more information on the `unpack.xml` file, see Chapter 36, “Publishing a complete store,” on page 321.

6. In the WEB-INF directory, do the following:
   a. Create the following subdirectory structure for your store's properties files:
      - classes
        - *Store directory*
   b. Copy your properties files into the *Store directory*.
   c. Create the following subdirectory structure for your store's data assets:
      - stores
        - *Store directory*
          - data: If your store supports several languages, create subdirectories for language-specific information using locale names. For example, en_US.
   d. Copy the data assets into the data directory and corresponding subdirectories.
   e. Using an existing `ibm-wc-load.xml` file from a sample store archive as an example, create an `ibm-wc-load.xml` file for your store. The `ibm-wc-load.xml` file determines how the store data will be loaded. For more information on the XML specifications, see the descriptor, `ibm-wc-load` in the following directory:
      - *WC_installdir*/xml/sar

      Note: For more information on the `ibm-wc-load.xml` file, see Chapter 36, "Publishing a complete store," on page 321.
   f. (Optional) If you want the store archive to have parameters that a user can select while publishing from the Administration Console, you must create a `ForeignKeys.dtd` file that store the values for these parameters. Using an existing `ForeignKeys.dtd` file from a sample store archive as an example, create an `ForeignKeys.dtd` file for your store.

      Note: For more information on the `ForeignKeys.dtd` file, see Chapter 36, "Publishing a complete store," on page 321.
7. Create a ZIP file composed of the *Store directory*, the SAR—INF directory, and the WEB-INF directory. Name this ZIP file *storearchivename.sar*.
8. If you want to publish your store archive using the Administration Console, see "Making the store archive available to the Administration Console" on page 332.

Creating a sample store archive

After packaging your store as a store archive, you may choose to use it as a sample store in the Administration Console. 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:
   - *WC_installdir*/samplestores
2. (Optional) Create preview pages. In order for previews of your store pages to display in the Administration Console, you must create preview pages. Do the following:
   a. (Optional) In the publish utility in the Administration Console, select a store archive to publish, 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`
  - *WC_installdir*/wc.ear/SiteAdministration.war/ tools/devtools/preview/*locale*/ *businessmodel*/*storedir*/
- HTML
  - *WC_installdir*/wc.ear/SiteAdministration.war/ tools/devtools/preview/*locale*/ *businessmodel*/*storedir*
- locale independent images
  - *WC_installdir*/wc.ear/SiteAdministration.war/ tools/devtools/preview/images/ *businessmodel*/*storedir*
- locale dependent images
  - *WC_installdir*/wc.ear/SiteAdministration.war/ tools/devtools/preview/*locale*/ *businessmodel*/*storedir*/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. Add the store archive to the `sarregistry.xml` file so that it will display in the publish utility in the Administration Console. For instructions, see “Making the store archive available to the Administration Console” on page 332.

Part 10. 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 36, “Publishing a complete store,” on page 321 - 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 the Administration Console or the command line publish.
- Chapter 37, “Overview of loading store data,” on page 335 - This chapter discusses publishing the store data assets to the database using the Loader package.
- Chapter 38, “Loading WebSphere Commerce database asset groups,” on page 383 - This chapter discusses publishing groups of store data assets or all of the store data to the database using the Loader package.
- Chapter 39, “Publishing business accounts and contracts,” on page 395 - This chapter discusses publishing the account, contract and product set assets.
- Chapter 40, “Publishing storefront assets and store configuration files,” on page 399 - This chapter discusses publishing the store front assets and the store configuration files.

Chapter 36. Publishing a complete store

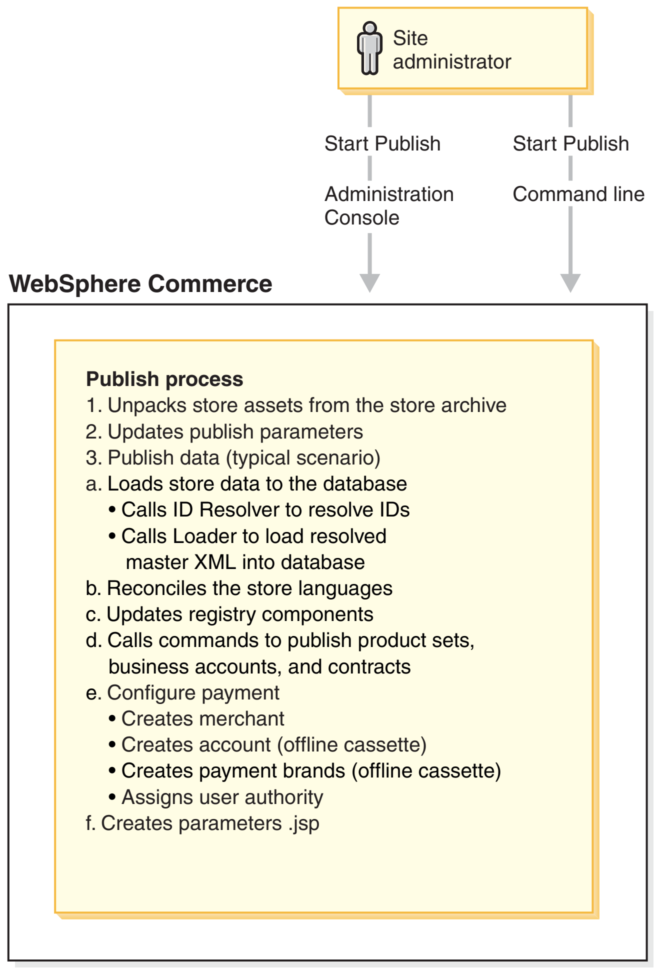
In order to create a functioning store, the store front Web assets must be published to the WebSphere Commerce Server, and the store data must be published to the WebSphere Commerce database. This chapter discusses publishing an entire store (store front and store data assets), if the store is in the form of a store archive, using either the publish utility in the Administration Console, 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 37, “Overview of loading store data,” on page 335, Chapter 38, “Loading WebSphere Commerce database asset groups,” on page 383, Chapter 39, “Publishing business accounts and contracts,” on page 395, and Chapter 40, “Publishing storefront assets and store configuration files,” on page 399.

Understanding publish in WebSphere Commerce

The publish option that is available from the Administration Console or from the command line allows you to publish a complete store (storefront 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 9, “Packaging your store,” on page 311.

The following diagram outlines the steps in the publishing process.


Site administrator
Start Publish
Administration Console
Start Publish
Command line
WebSphere Commerce
Publish process
1. Unpacks store assets from the store archive
2. Updates publish parameters
3. Publish data (typical scenario)
a. Loads store data to the database
• Calls ID Resolver to resolve IDs
• Calls Loader to load resolved master XML into database
b. Reconciles the store languages
c. Updates registry components
d. Calls commands to publish product sets, business accounts, and contracts
e. Configure payment
• Creates merchant
• Creates account (offline cassette)
• Creates payment brands (offline cassette)
• Assigns user authority
f. Creates parameters .jsp

Start publish

In order to publish a store, you must have Site Administrator authority. Site Administrators can initiate the publish process using either of the following methods:

- Administration Console
- Command line

Both methods of publishing require you to specify the store archive you want to publish.

Note: In order to publish a store archive using the Administration Console, it must be in the right location or registered. For more information, see "Making the store archive available to the Administration Console" on page 332.

You may then change values for selected parameters, if available, including the store identifier (the name that uniquely identifies the store), store directory (the unique location to which the JSP files and the images will be published) and organization (the organization to which you publish the store archive).

Publish parameters in the Administration Console

The publish parameters in the publish utility in the Administration Console are defined by the `store-refs.xml` file in each store archive.

Look at the following example of a `store-refs.xml` file from the `ConsumerDirectStore.sar` file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE store-refs SYSTEM "store-refs.dtd">
<store-refs
    target-dtd="WEB-INF/stores/FashionFlow/data/ForeignKeys.dtd"
    deploy-descriptor="WEB-INF/stores/FashionFlow/data/ibm-wc-load.xml"
 resource-bundle="/SAR-INF/properties/publishNLS">
ref id="storeDir" entity="STORE_DIR" >
        <input type="text"/>
</ref>
<ref id="storeIdent" entity="STORE_IDENTIFIER" >
        input type="text"/>
</ref>
<ref id = "parentOrg" entity="ORGANIZATION_DN">
 <input type="member" />
</ref>
</store-refs>
```

This file defines three publish parameters for the `ConsumerDirectStore.sar` file:

-
  ```
  <ref id="storeDir" entity="STORE_DIR" >
          <input type="text"/>
  ```
  This entity creates the publish parameter store directory.

-
  ```
  <ref id="storeIdent" entity="STORE_IDENTIFIER" >
         < input type="text"/>
  ```
  This entity creates the publish parameter store identifier.

-
  ```
  <ref id = "parentOrg" entity="ORGANIZATION_DN">
  <input type="member" />
  ```
  This entity creates the publish parameter organization.

where

- `ref id` is used as the key in the properties file specified by the store-refs resource-bundle attribute. It is used to obtain the translatable parameter name and the description that displays in the publish parameters page.
- `entity` is the name of the ENTITY in the target-dtd that is edited by this parameter.
- `input type` controls how the parameter is displayed on screen. If the input type is text, the parameter is displayed in an editable field. If the input type is member, all of the existing organizations display in a drop-down list. Read-only parameters cannot be edited.

The values that a user enters for these parameters are stored in the file identified in the `target-dtd` file. The `target dtd` is defined in the following code:

```
target-dtd="WEB-INF/stores/ConsumerDirect/data/ForeignKeys.dtd
```

This file is also part of the store archive and is unpacked with the store data assets. The entity value corresponding to each parameter is updated in the unpacked file. The DTD inside the store archive is not updated. The values for the parameters are stored in this file (in this case `ForeignKeys.dtd`) until publish is instantiated.

Finally, if a store may be published in several languages, as the sample stores are, the publish parameters and their accompanying descriptions are found in locale specific files. The field label and description for each publish parameter are located in the properties file defined in the resource-bundle attribute of the store-refs.xml. During publish, publish looks for the specific locale for the language used in the Administration Console. The stores-ref.xml file also defines these files:

- resource-bundle="/SAR-INF/properties/publishNLS"

Note: Publishing parameters are only available through the Administration Console. If you publish a store archive through the command line, you cannot specify parameter values. The default values contained in the store archive will be used.

The deploy descriptor specifies the location of the file(ibm-wc-load.xml) lthat controls the publish data portion of the publishing process. For example,deploy-descriptor="WEB-INF/stores/FashionFlow/data/ibm-wc-load.xml

After selecting your parameters, you click **Finish** to initiate the publish. After you have initiated the publish process using either the Administration Console 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.

For more detailed information on how to publish a store archive using either the Administration Console or the command line, see the WebSphere Commerce Production online help, topic, ″Publishing a store archive″.

Unpack the assets from the store archive

After you have clicked **Finish** in the publish wizard in the Administration Console, or run publish from the command line, WebSphere Commerce unpacks the assets from the store archive to the WebSphere Commerce Server. Unpacking the assets is controlled by the unpack.xml file, located in the SAR-INF directory in the store archive.

The following example of an unpack.xml file is from the ConsumerDirect.sar file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ibm-wc-unpack SYSTEM "unpack.dtd">
<ibm-wc-unpack>
 <unpack>
  <include file="*"/>
  <exclude file="SAR-INF/*"/>
  <exclude file="*.zip"/>
  <exclude file="*.war"/>
  <exclude file="FashionFlow/devtools/flow/ui/*.properties"/>
  <exclude file="xml/*"/>
  <rename-store-dir target-name="FashionFlow">
   <store-dir path="WEB-INF/stores/FashionFlow" />
   <store-dir path="FashionFlow" />
   <store-dir path="WEB-INF/classes/FashionFlow" />
   <store-dir path="WEB-INF/xml/tools/stores/FashionFlow" />
  </rename-store-dir>
 </unpack>
 <unpack dest="${wc:ToolsStoresPropertiesPath}">
  <include file="FashionFlow/devtools/flow/ui/*.properties"/>
  <rename-store-dir target-name="FashionFlow">
   <store-dir path="FashionFlow" />
  </rename-store-dir>
 </unpack>
 <unpack dest="${wc:instanceDir}">
  <include file="xml/member/MemberRegistrationAttributes.xml"/>
 </unpack>
</ibm-wc-unpack>
```

The `unpack.xml file` determines which files to unpack (using the include and exclude elements), to which directories the files will be unpacked (using the unpack dest entity), and also renames directories (using the rename-store-dir entity).

By default, unpack unpacks all of the files of the store archive that it is located in. However, unpack can also unpack just certain files within the store archive, if specified. Also by default, unpack unpacks the files to the path obtained by combining the StoreDocRoot and StoreWebPath paths from the DevTools element in the instance XML. This path points to the document root of the Stores Web module. However, if specified, as in the above example, `<unpack dest="${wc:ToolsStoresPropertiesPath}">`, unpack will unpack the files in another location. Note unpack accepts variables. In this case the `"${wc:ToolsStoresPropertiesPath}"` the ToolsStoresPropertiesPath variable is an attribute of the devtools element in the *instance*.xml.

Updates publish parameters

After the assets are unpacked from the store archive, WebSphere Commerce updates the DTD file (in the sample stores, `ForeignKeys.dtd`) with the publishing parameter values created or selected in the publish wizard. For example, if the original file contained `<!ENTITY STORE_IDENTIFIER "FashionFlow">`, the updated file contains `<!ENTITY STORE_IDENTIFIER "MyFashion">`.

Note: Publishing parameters are only available through the Administration Console. If you publish a store archive through the command line you can not select publishing parameters

Publish data

After the files are unpacked and the publish parameters are updated, a scheduled job is created for the publish process. The scheduled job number for publishing the store archive is displayed in the publishing utility in the Administration Console.

When the scheduler runs the publish job, WebSphere Commerce typically completes the following actions:

- Loads store data from the XML in the store archive to the database
- Reconciles store data
- Updates the registry components
- Calls the commands to publish business accounts and contracts
- Configures payment
- Creates parameters.jsp file

The publish job is controlled by the `ibm-wc-load.xml` file, contained in each store archive.This file is specified by the deploy-descriptor attribute in the `stores-refs.xml` file.

ibm-wc-load.xml

The `ibm-wc-load.xml` file determines what tasks will be completed in the publish job and the sequence of these tasks. The following is an example of an `ibm-wc-load.xml` file:

```
<data-deploy base-dir="." default-target="all">
    <asset id="master" location="store-data-assets.xml"/>
    <asset id="resolved.master" location="store-data-assets.resolved.xml"/>
    <asset id="foreignKeys" location="ForeignKeys.dtd" type="dtd"/>
    <asset id="pmconfigfile" location="paymentinfo.xml"/>
    <deploy-task-cmd name="configPM" class="com.ibm.commerce.tools.devtools.
publish.tasks.payment.ConfigurePaymentTaskCmd"/>
    <target id="all">
     <task name="idresolve">
              param name="infile" value="${asset:master}" />
              param name="outfile" value="${asset:resolved.master}" />
     </task>
     <task name="massload">
              param name="infile" value="${asset:resolved.master}" />
     param name="maxerror" value="1" />
              param name="noprimary" value="error" />
    </task>
    <task name="configPM">
              param name="paymentConfigFilename" value="${asset:pmconfigfile}" />
              param name="storeIdentifier" value="
${asset:foreignKeys#STORE_IDENTIFIER}" />
              param name="organizationDN" value="
${asset:foreignKeys#ORGANIZATION_DN}" />
        </task>
   </target>
</data-deploy>
```

where

- `base-dir` is the directory of the the information that is to be published. "." indicated that the information is located in the same directory as the `ibm-wc-load.xml` file.
- `default-target` is the ID of a target that is to be executed. Only one target is executed during publish.
- `asset id` is the ID assigned to the assets to be published. Assets are assigned IDs as more than one task may act upon them during the publish process.
- `location` is the location of the asset assigned the ID, relative to the base directory (base-dir).
- `deploy-task-cmd name` is the short name assigned to task commands used within the publish process.
- `deploy-task-cmd class` is the full name of task commands used within the publish process.
- `target id` is a name given to a sequence of tasks that are executed as a group. Multiple targets can be defined but only the one referred to by default target is executed during publish.
- `task name` is the name of the task to be completed. Note that in this sample the short name of the task is used. You may add new tasks to the `ibm-wc-load.xml` file, but any new task must extend the following command: `com.ibm.commerce.tools.devtools.publish.tasks.DeployTaskCmd.`
- `param name` is the name of the parameter for the task. Input parameters are passed to the task as name-value pair strings.
- `value` is the value for the parameter. Note that values can be variables. These variables will be resolved when the task is run.

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:

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 439.

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 37, "Overview of loading store data," on page 335.

When the 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. You can specify which method the publish will use in the WebSphere Commerce Configuration File, `*instance_name*.xml`. By default, publish uses the mixed method. For more information on the ID Resolver methods, see "ID Resolve command" on page 339.

Publish must also specify a customizer file to be used with the ID Resolver. The default customizer files are the following: DBConnectionCustomizer or OracleConnectionCustomizer.

Oracle

The OracleConnectionCustomizer customizer file is located in the following directory:

- *WAS_installdir*/installedApps/*cell_name*/WC_*instance_name*.ear/properties

DB2

The DBConnectionCustomizer file is located in the following ZIP file:

- *WAS_installdir*/installedApps/*cell_name*/WC_*instance_name*.ear/properties
- 400 WAS_*userdir*/WAS_*instance_name*/installedApps/*cell_name*/WC_*instance_name*.ear/lib/loader/IdResGen.zip

Note: If you want to specify your own customizer file, you must change the value of the following attribute in the DevTools section of the `*instance_name*.xml` file:

- `IDResolverCustomizerFile="*myIDResolverCustomizerFile*"`

store-data-asset.xml: Each sample store archive contains a `store-data-asset.xml` file. The `store-data-asset.xml` file includes placeholders for all of the data asset files in the store archive that will be included during publish.

The following is an portion of the `store-data-asset.xml` file for the `ConsumerDirect.sar`, illustrating the placeholders:

```
<?xml version="1.0"?>
<!DOCTYPE import SYSTEM "store-data-assets.dtd">

<import>
 &modelorg.xml;
```

```
&modelorgrole.xml;
&storeorg.xml;
&storeorgrole.xml;
&fulfillment.xml;
&store.xml;
&en_US_store.xml;
&en_US_fulfillment.xml;
&catalog.xml;
&en_US_catalog.xml;
&tax.xml;
```

During publish, all the data assets identified with placeholders in the `store-data-asset.xml` file are consolidated into the `store-data-asset.xml` file, creating one large data file.

The ID Resolver uses the `store-data-asset.xml` and corresponding DTD file, `store-data-asset.dtd` to resolve the IDs. After the IDs are resolved, ID Resolver creates the following file, `store-data-asset.resolved.xml` file, which contains the unique identifiers. If an error occurs during the ID resolving process, the Loader package adds an entry to the `messages.txt` file. For more information, see "Publish log files" on page 330.

Calls the Loader package to load the resolved master XML file into the database: The Loader package loads the resolved `store-data-asset.resolved.xml` into the database. If an error occurs during the loading process, the Loader package adds an entry to the `messages.txt` file.

For more information on the Loader package, see Chapter 37, "Overview of loading store data," on page 335.

When the Administration Console or command line publish calls the Loader package, it must specify which Loader method to use. The Administration Console can use any of the following Loader methods:
- SQL import
- Import
- Load

Note: By default the Administration Console uses the SQL import method.
You can specify which method the Administration Console 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 349.

The Administration Console 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 change the value of the following attribute in 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.

Reconciles the store languages

The sample store archives contain data for all languages supported by WebSphere Commerce. As a result, when the Loader package loads the store data, all language information is loaded for the store. However, a store can only support the languages that are supported by the instance the store resides in. The reconcile store language task ensures that only the languages supported in the instance are enabled in the store. If you want to add support to the instance for additional languages, do so by using the Configuration Manager. For more information, see the WebSphere Commerce Production online help, topic ″Configuration Manager.″

Updates registry components

The publish process also updates the registry components. Publish updates all of the registries in WebSphere Commerce by calling the `com.ibm.commerce.scheduler.commands.RefreshRegistryCmd`. For more information on the registries, see the WebSphere Commerce Production and Development online help.

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. The Administration Console 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 39, "Publishing business accounts and contracts," on page 395.

Configures payment

The publishing process also includes a step to configure payment. WebSphere Commerce supports WebSphere Commerce Payments. If you plan to use WebSphere Commerce Payments as your method of processing payment, you should create a payment XML file as described in Chapter 21, “Payments instruments,” on page 207. 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”).

Creates parameters.jsp file

The publish process creates the file `parameters.jsp`. This file includes the storeId parameter. . The index.jsp file in the sample stores uses this parameter to launch the store.

`parameters.jsp` is located in the following directory:

- *WAS_installdir*/installedApps/*cell_name*/WC_*instance_name*.ear/Stores.war/*storedir*/include
- ▶ 400 *WAS_userdir*/*WAS_instance_name*/installedApps/*cell_name*/WC_*instance_name*.ear/Stores.war/*storedir*/include

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 the Administration Console.

Publish log files

Any errors encountered during the publish assets phase of the publishing process are written to the following log and trace files:

- `activity.log`: WebSphere Application Server log. All error messages in WebSphere Commerce are written to activity.log. Check this log first when publish fails. `activity.log` is located in the following directory:
  - *WAS_installdir*/logs
  - ▶ 400 *WAS_userdir*/logs
  - ▶ Developer *WCDE_installdir*/*workspace_name*/.metadata/plugins/com.ibm.etools.server.core/tmp0/logs
- `SystemOut.log and SystemErr.log`: SystemOut.log is located in the following directory Contains any information written to standard output and standard error during store publish. `SystemOut.log and SystemErr.log` is located in the following directory:
  - *WAS_installdir*/logs/*instance_name*
  - ▶ 400 *WAS_userdir*/*WAS_instance_name*/logs/*WC_instance_name*

Note: `SystemOut.log` and `SystemErr.log` are not available in the development environment. Information written to standard output or standard error are displayed in the WebSphere Studio console and captured in the following location:

  – workspaceDir\.metadata\.plugins\com.ibm.etools.server.core\tmp0\logs\server1\trace.log

- `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 `store-data-asset.resolved.xml`file.`messages.txt` is located in the following directory:
  - WC_*installdir*/instances/*instance_name*/logs
  - 400 WC_*userdir*/instances/*instance_name*/logs
  - Developer WCDE_*installdir*/Commerce/logs

  Line or column numbers mentioned in these error messages may also refer to `store-data-asset.xml` file.
- `trace.txt`: Contains trace information for the Loader package part of the publishing process. This file also contains messages about the IDResolver part of the publishing process. By default, trace.txt is turned off. `trace.txt` is located in the following directory:
  - WC_*installdir*/instances/*instance_name*/logs
  - 400 WC_*userdir*/instances/*instance_name*/logs

    Note: By default, trace is turned on, and is generated as a circular log file.
  - Developer WCDE_*installdir*/Commerce/instances/*instance_name*/logs
- `trace.log`: Part of the WebSphere Application Server trace logs. Contains trace information for the publishing process, if the WC_DEVTOOLS trace component is enabled. For more information on enabling logging, see the WebSphere Commerce Administration Guide, Configuration chapter. `trace.log` is located in the following directory:
  - *WAS_installdir*/logs/*WC_instance_name*
  - 400 *WAS_userdir*logs/*WC_instance_name*
  - Developer *workspaceDir*\.metadata\.plugins\com.ibm.etools.server.core\tmp0\logs\server1
- 400 `RESWCSID.txt`: Contains messages from the IDResolver part of the publishing process. Line or column numbers mentioned in messages refer to the input files, for example the store-data-assets.xml file. `RESWCSID.txt` is located in the following directory:
  - WC_*userdir*/instances/*instance_name*/logs

To configure the trace.txt and messages.txt log files (that is, adjust the log level or other options), edit the following file:
- WC_*installdir*/instances/*instance_name*/xml/loader/WCALoggerConfig.xml
- 400 WC_*userdir*/instances/*instance_name*/xml/loader/WCALoggerConfig.xml
- Developer WCDE_*installdir*/Commerce/instances/*instance_name*/xml/loader/WCALoggerConfig.xml

Making the store archive available to the Administration Console

In order to publish a store archive from the Administration Console, the store archive must be available to the Administration Console through one of the following methods:

- Register the store archive in the SARRegistry.xml file
- Copy the store archive to the applicable store archive directory

Register the store archive in the SARRegistry.xml file

In order to publish the store archive and to preview the store from the Administration Console, the store archive must be registered in the `SARRegistry.xml` file. The `SARRegistry.xml` file is located in the following directory:

- *WC_installdir*/xml/tools/devtools
- 400 *WC_userdir*/xml/tools/devtools

To register the store archive, include the path of the store archive file, as well as the path to any preview files in the `SARRegistry.xml` file. The following example illustrates the entry that registers the consumer direct sample store archive in the `SARRegistry.xml` file.

```
<!-- Consumer Direct -->
 <SampleSAR fileName="ConsumerDirect.sar" relativePath="ConsumerDirect">
  <view name="ConsumerDirect" />
  <view name="default" />
  <html locale="de_DE" featureFile="" sampleSite="de_DE/B2C/FashionFlow/
index.html"/>
  <html locale="en_US" featureFile="" sampleSite="en_US/B2C/FashionFlow/
index.html"/>
  <html locale="es_ES" featureFile="" sampleSite="es_ES/B2C/FashionFlow/
index.html"/>
  <html locale="fr_FR" featureFile="" sampleSite="fr_FR/B2C/FashionFlow/
index.html"/>
  <html locale="it_IT" featureFile="" sampleSite="it_IT/B2C/FashionFlow/
index.html"/>
  <html locale="ja_JP" featureFile="" sampleSite="ja_JP/B2C/FashionFlow
index.html"/>
  <html locale="ko_KR" featureFile="" sampleSite="ko_KR/B2C/FashionFlow/
index.html"/>
  <html locale="pt_BR" featureFile="" sampleSite="pt_BR/B2C/FashionFlow/
index.html"/>
  <html locale="zh_CN" featureFile="" sampleSite="zh_CN/B2C/FashionFlow/
index.html"/>
  <html locale="zh_TW" featureFile="" sampleSite="zh_TW/B2C/FashionFlow/
index.html"/>
 </SampleSAR>
```

where

- `fileName` is the name of the store archive.
- `relativePath` is the directory path relative to the sampleSarPath attribute of the DevTools element in the WebSphere Commerce configuration file, *instance_name.xml*.
- `view name` is the name of the view that the store archive will display in from the Publish user interface. Note that a store archive may display in multiple views. For example, displays in both the Consumer Direct view and the default view.
- `html locale` is the locale for the HTML preview files.
- `featureFile`Feature file is deprecated an no longer used.

Copy the store archive to the applicable store archive directory

If you want to publish the store archive from the Administration Console, but do not plan to include preview pages, or specify the Publishing view in which the store displays in the Administration Console, you can simply copy the store archive to the following directory:

- *WC_installdir*/instances/*instance_name*/sar

The store archive will display in the Default view.

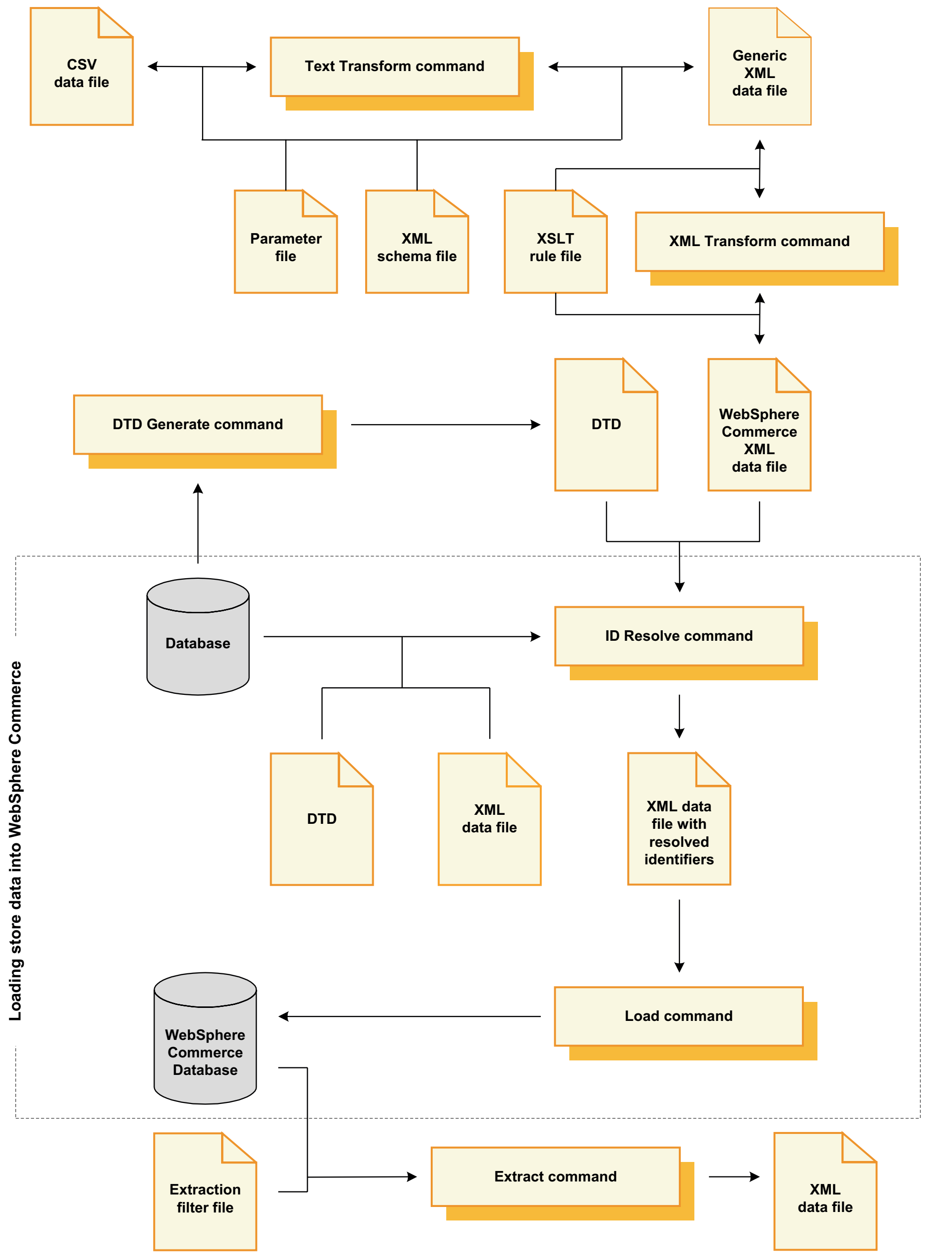
Chapter 37. Overview of loading store data

After creating your store data, you can choose to package it as a store archive and publish it using WebSphere Commerce Administration Console or you can load it directly into the WebSphere Commerce Server database using the WebSphere Commerce Loader package. Refer to Chapter 38, “Loading WebSphere Commerce database asset groups,” on page 383 and “Loading database asset groups” on page 390 for information on the loading process for WebSphere Commerce database asset groups.

The Loader package provides six command-line utilities and two 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 5, “Store data overview,” on page 95.

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 339, "Using the Loader package commands and scripts" on page 370, and "Examples of resolving identifiers" on page 371.
- **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 349, "Using the Loader package commands and scripts" on page 370, and "Example of loading data" on page 378.

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 359 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 362 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 365 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 366 for more information.

These commands are not the primary focus of this chapter. For detailed information on these commands, refer to the WebSphere Commerce Production and Development online help.

The WebSphere Commerce Loader package 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.

These tools are not the primary focus of this chapter. For detailed information on these tools, refer to the most recent version of the WebSphere Commerce Production and Development online help.

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.

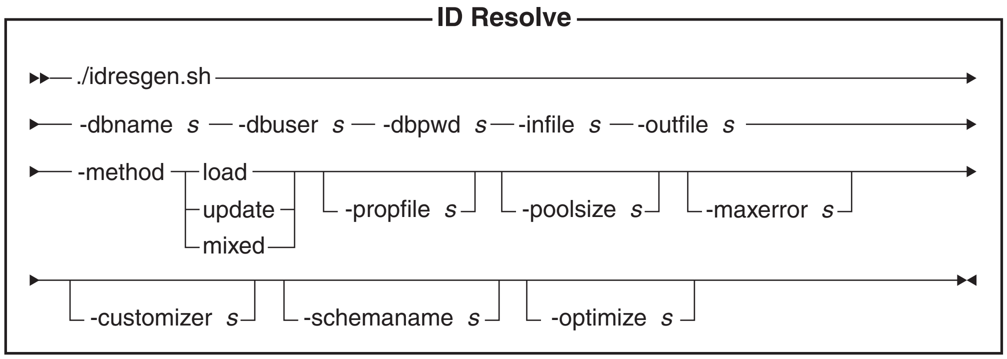
The WebSphere Commerce database schema defines primary and foreign keys within its tables that are used to represent various relationships between the tables. For this reason, WebSphere Commerce XML elements must contain corresponding attributes with unique identifiers. Within the WebSphere Commerce 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 439.

Notes:

1. The above diagram is intended primarily as a reference for the command parameters.
2. File names specified as parameters for this command can be preceded by relative or absolute paths.



```
ID Resolve

./idresgen.sh
-dbname s -dbuser s -dbpwd s -infile s -outfile s
-method load|update|mixed [-propfile s] [-poolsize s] [-maxerror s]
[-customizer s] [-schemaname s] [-optimize s]
```

Note: File names specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

-dbname

AIX Linux Solaris Name of the target database.

400 This is the database name as displayed in the relational database directory (WRKRDBDIRE).

-dbuser

AIX Linux Solaris Name of the user connecting to the database.

400 This is usually the same as the instance user name.

-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. The command can treat the input file as though the records do not exist in the database (load) or as if there are already identifiers for the input objects (update). Use the mixed method when some records do not exist in database and some do. 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:

AIX Linux Solaris

```
-propfile WC_installdir/my_directory/file_name.properties
-propfile WC_installdir/my_directory/file_name
```

400

```
-propfile WC_userdir/my_directory/file_name.properties
-propfile WC_userdir/my_directory/file_name
```

If this file exists in the current directory, the same file can be specified as shown in the following example:

```
-propfile file_name.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 file_name
```

where, *my_directory* is a user defined directory and *file_name* is the name of the property file that you are going to use

For more information on creating and specifying a new properties file for use with the ID Resolver, refer to the WebSphere Commerce Production and Development online help.

-poolsize
Number of identifiers to be reserved. This parameter is optional. The default number is 50.

-maxerror
Number of errors after which the ID Resolver will terminate. This parameter is optional. The default value is 1.

-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:

AIX Linux Solaris DB2ConnectionCustomizer.properties

400 ISeries_RESWCSID_Customizer.properties

If you have configured your instance to use the toolbox driver, then use the Toolbox_RESWCSID_Customizer customizer file provided for the toolbox driver. You must also specify the hostname for the -dbname parameter. The following is an example of invoking the idresgen.sh script:

```
./idresgen.sh -dbname MY.HOSTNAME.CA -dbuser instance -dbpwd mypass
-infile /path/infile.xml -outfile /path/outfile.xml -method sqlimport
-customizer Toolbox_RESWCSID_Customizer
```

The customizer property file can be specified as shown in either of the following examples:

AIX Linux Solaris

```
-customizer WC_installdir/my_directory/file_name.properties
```

```
-customizer WC_installdir/my_directory/file_name
```

400

```
-customizer WC_userdir/my_directory/file_name.properties
-customizer WC_userdir/my_directory/file_name
```

where, *my_directory* is a user defined directory and *file_name* is the name of the property file that you want to use.

If this file exists in the current directory, the same file can be specified as shown in the following example:

```
-customizer file_name.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 file_name
```

For more information on creating and specifying a new customizer property file, refer to the WebSphere Commerce Production and Development online help.

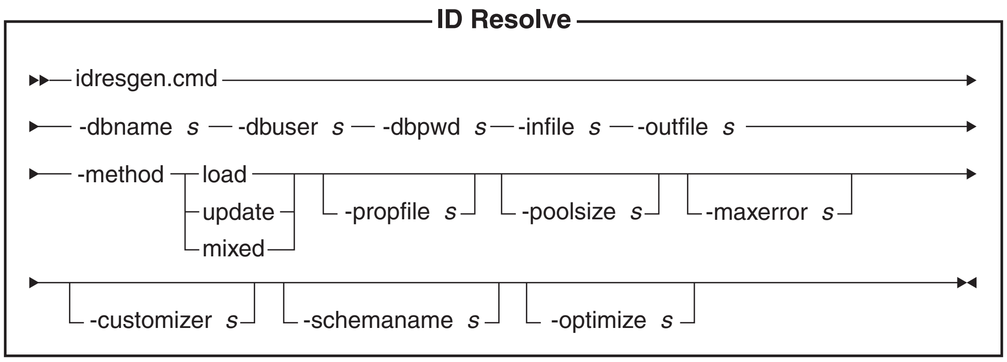
-schemaname
Name of the target database schema. This parameter is optional.

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 KEYS table in the database. 400 If neither a command-line nor a property-file specification for the -schemaname parameter exists, the command defaults to the value of the -dbuser parameter.

-optimize
-optimize no

The IdResolver will skip duplicate record checking before writing resolved records to the output file. This option allows the user to switch off the optimization feature in IdResolver.

Windows



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. The command can treat the input file as though the records do not exist in the database (load) or as if there are already identifiers for the input objects (update). Use the mixed method when some records do not exist in database and some do. 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 WC_installdir\my_directory\ file_name.properties
-propfile WC_installdir\my_directory\file_name
```

If this file exists in the current directory, the same file can be specified as shown in the following example:

```
-propfile file_name.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 file_name
```

where, *my_directory* is a user defined directory and *file_name* is the name of the property file that you want to use.

For more information on creating and specifying a new properties file for use with the ID Resolver, refer to the WebSphere Commerce Production and Development online help.

-poolsize
Number of identifiers to be reserved. This parameter is optional. The default number is 50.

-maxerror

Number of errors after which the ID Resolver will terminate. This parameter is optional. The default value is 1.

-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 WC_installdir\my_directory\file_name.properties
-customizer WC_installdir\my_directory\file_name
```

If this file exists in the current directory, the same file can be specified as shown in the following example:

```
-customizer file_name.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 file_name
```

where, *my_directory* is a user defined directory and *file_name* is the name of the property file that you are going to useFor more information on creating and specifying a new customizer property file, refer to the WebSphere Commerce Production and Development online help.

-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 KEYS table in the database.

-optimize

-optimize no

The IdResolver will skip duplicate record checking before writing resolved records to the output file. This option allows the user to switch off the optimization feature in IdResolver.

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 the Administration Console, 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. When publishing from the WebSphere Commerce Administration Console 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 439.

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:

- AIX Linux 400 Solaris

  ```
  ./idresgen.sh -dbname db -dbuser user -dbpwd pwd -infile input.xml
  -outfile output.xml -method load -customizer customizer -schemaname wcsadmin
  ```

- Windows

  ```
  idresgen -dbname db -dbuser user -dbpwd pwd -infile input.xml
  -outfile output.xml -method load -customizer customizer -schemaname wcsadmin
  ```

Note: Refer to "Using the Loader package commands and scripts" on page 370 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 346. 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:

- AIX Linux 400 Solaris

  ```
  ./idresgen.sh -dbname db -dbuser user -dbpwd pwd -infile input.xml
  -outfile output.xml -method update -customizer customizer -schemaname wcsadmin
  ```

- Windows

  ```
  idresgen -dbname db -dbuser user -dbpwd pwd -infile input.xml
  -outfile output.xml -method update -customizer customizer -schemaname wcsadmin
  ```

Note: Refer to "Using the Loader package commands and scripts" on page 370 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:

- AIX Linux 400 Solaris

  ```
  ./idresgen.sh -dbname db -dbuser user -dbpwd pwd -infile input.xml
  -outfile output.xml -method mixed -customizer customizer -schemaname wcsadmin
  ```

- Windows

  ```
  idresgen -dbname db -dbuser user -dbpwd pwd -infile input.xml
  -outfile output.xml -method mixed -customizer customizer -schemaname wcsadmin
  ```

Notes:

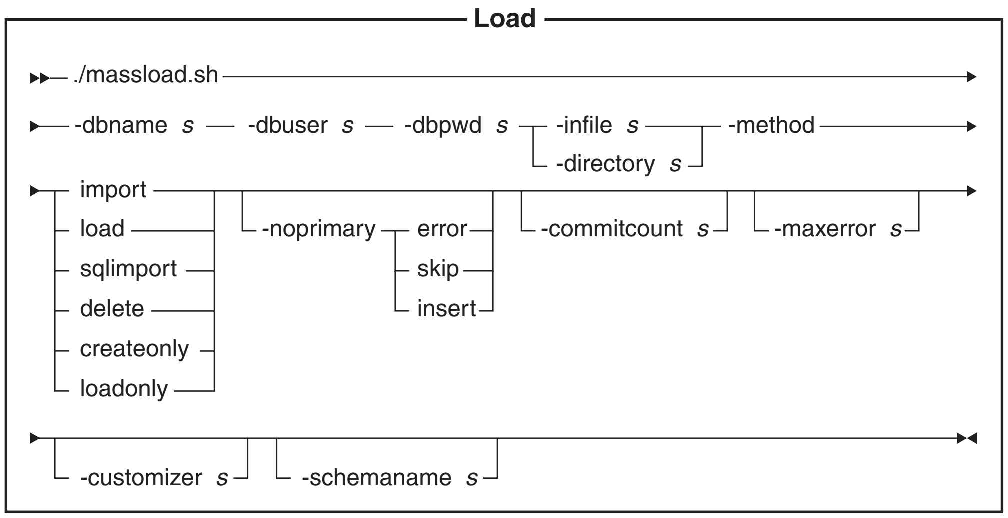
1. Refer to "Using the Loader package commands and scripts" on page 370 for the location of the appropriate ID Resolve command or script.
2. Mixed method is the recommended method for the Administration Console.

For detailed information on setting up and customizing the files used to run this command, refer to the WebSphere Commerce Production and Development online help.

Load command

This command loads an XML input file into a target database.

AIX Linux 400 Solaris



Notes:

1. The above diagram is intended primarily as a reference for the command parameters.
2. File names specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

-dbname

AIX Linux Solaris Name of the target database

400 This is the database name as displayed in the relational database directory (WRKRDBDIRE).

-dbuser

AIX Linux Solaris Name of the user connecting to the database.

400 This is usually the same as the instance user name.

-dbpwd
Password for the user connecting to the database

-infile Name of the input XML file

-directory
You use the -infile parameter if you are using any option other than loadonly for the -method parameter as described below. If you use the loadonly method, you must replace the -infile parameter with the -directory parameter or an error will result. For the value of the -directory parameter when you use the loadonly method, specify the fully qualified path of the MassLoadOutputFiles directory that was created using the createonly method as described below.

-method
Mode of operation for the Loader to use when modifying the database using input data

load 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.

400 The load method does not support bit data or DBCLOB fields.

import
The import 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 default is import.

400 The import method can only be used on local databases.

sqlimport
The SQL import (sqlimport) method can be used with both local and remote databases.

delete

The delete method deletes data from the database

createonly

To improve performance during instance creation, use the createonly method. Use the createonly method to create mass-load data (MLD) files without loading the data into the database. The files that are created when you use this method (.mld and .cmd files) are placed in a directory named MassLoadOutputFiles. AIX Linux Solaris This directory is created as a subdirectory under the directory from which you run the Load command, which is your working directory. As a result, your working directory must be writable. 400 This directory is created as a subdirectory of the temp directory, located in the root directory of the instance. The default location of the directory will be *WC_userdir*/instances/*instance_name*/temp/MassLoadOutputFiles.

Here is an example of running the Load command using the createonly method:

```
./massload.sh -dbname mall -dbuser db2admin -dbpwd db2admin
-infile WC_installdir/data/example.xml
-method createonly
```

400

```
./massload.sh -dbname mall -dbuser db2admin -dbpwd db2admin
-infile WC_userdir/data/example.xml
-method createonly
```

You can later use your native database load utility to load the MLD files that you created into a WebSphere Commerce database by running the Load command using the loadonly method described below.

Note: The program obtains information about the native database load utility that your database product uses from the customizer property file.

loadonly

Use the loadonly method to load MLD files that were created using the createonly method described previously. When you use the loadonly method, you must also use the -directory parameter or an error will result.

Note: The -directory parameter replaces the -infile parameter that you would specify if you were using any method other than loadonly.

For the value of the -directory parameter, you must specify the fully qualified path of the MassLoadOutputFiles directory that was created using the createonly method.

Here is an example of running the Load command using the loadonly method (and the required -directory parameter) :

```
./massload.sh -dbname mall -dbuser db2admin -dbpwd db2admin
-method loadonly -directory WC_installdir/bin/MassLoadOutputFiles
 -schemaname wcsadmin
```

Always specify the name of the target database schema using the -schemaname parameter when you run the Load command using this method. Otherwise, the program uses the name of the database schema obtained when the MassLoadOutputFiles directory and its files were originally created.When you use the loadonly method, errors and other messages are saved in files that have a .log extension. These log files are written to the MassLoadOutputFiles directory specified for the -directory parameter.

Use the loadonly method **only** for instance creation. If you use it at any other time, the result may not be desirable.

-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. The default value is 1.

-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. The default file is:

AIX Linux Solaris MassLoadCustomizer.properties

400 `ISeries_LODWCSDTA_Customizer.properties`

If you have configured your instance to use the toolbox driver, then use the `Toolbox_LODWCSDTA_Customizer` customizer file provided for the toolbox driver. You must also specify the hostname for the -dbname parameter. The following is an example of invoking the `massload.sh` script:

```
./massload.sh -dbname MY.HOSTNAME.CA -dbuser instance -dbpwd mypass
-method sqlimport -customizer Toolbox_LODWCSDTA_Customizer
-infile /path/file.xml
```

The customizer property file can be specified as shown in the following example:

AIX Linux Solaris

`-customizer` *`WC_installdir/my_directory/file_name`*`.properties`

`-customizer` *`WC_installdir/my_directory/file_name`*

400

`-customizer` *`WC_userdir/my_directory/file_name`*`.properties`

`-customizer` *`WC_userdir/my_directory/file_name`*

If this file exists in the current directory, the same file can be specified as shown in the following example:

`-customizer` *`file_name`*`.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` *`file_name`*

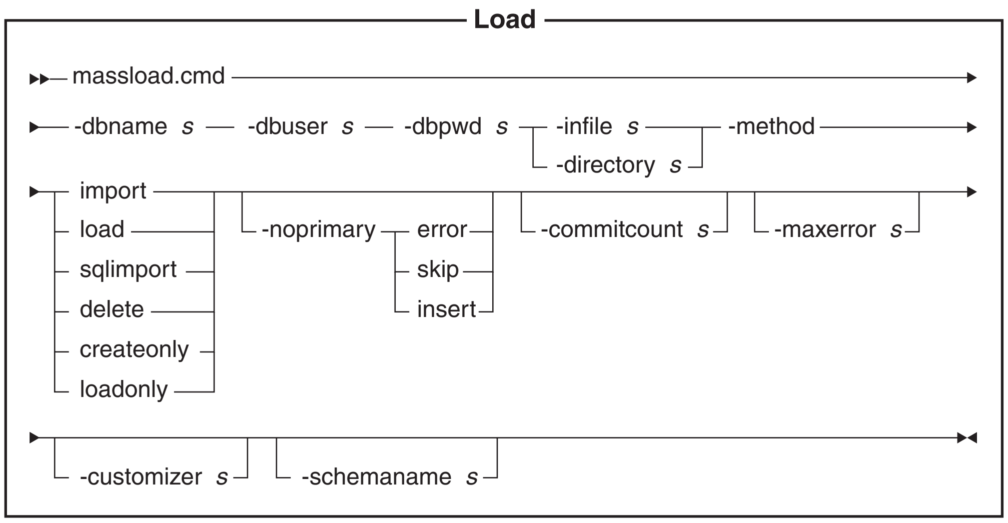
where, *my_directory* is a user defined directory and *file_name* is the name of the property file that you are going to useFor more information on creating and specifying a new customizer property file, refer to the WebSphere Commerce Production and Development online help.

-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 KEYS table in the database. 400 If neither a command-line nor a property-file specification for the -schemaname parameter exists, the command defaults to the value of the -dbuser parameter.

Windows



Note: File names 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

-directory
You use the -infile parameter if you are using any option other than loadonly for the -method parameter as described below. If you use the loadonly method, you must replace the -infile parameter with the -directory parameter or an error will result. For the value of the -directory parameter when you use the loadonly method, specify the fully qualified path of the MassLoadOutputFiles directory that was created using the createonly method as described below.

-method
Mode of operation for the Loader to use when inserting data into the database

load 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.

import
The import 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 default is import.

sqlimport
The SQL import (sqlimport) method can be used with both local and remote databases.

delete

The delete method deletes data from the database.

createonly

To improve performance during instance creation, use the createonly method. Use the createonly method to create mass-load data (MLD) files without loading the data into the database. The files that are created when you use this method (.mld and .cmd files) are placed in a directory named MassLoadOutputFiles. This directory is created as a subdirectory under the directory from which you run the Load command, which is your working directory. As a result, your working directory must be writable.

Here is an example of running the Load command using the createonly method:

```
massload -dbname mall -dbuser db2admin -dbpwd db2admin -infile
WC_installdir\data\example.xml
-method createonly
```

You can later use your native database load utility to load the MLD files that you created into a WebSphere Commerce database by running the Load command using the loadonly method described below.

Note: The program obtains information about the native database load utility that your database product uses from the customizer property file.

loadonly
Use the loadonly method to load MLD files that were created using the createonly method described previously. When you use the loadonly method, you must also use the -directory parameter or an error will result.

Note: The -directory parameter replaces the -infile parameter that you would specify if you were using any method other than loadonly.

For the value of the -directory parameter, you must specify the fully qualified path of the MassLoadOutputFiles directory that was created using the createonly method.

Here is an example of running the Load command using the loadonly method (and the required -directory parameter) :

```
massload -dbname mall -dbuser db2admin -dbpwd db2admin -method
loadonly -directory WC_installdir\bin\MassLoadOutputFiles
-schemaname wcsadmin
```

Always specify the name of the target database schema using the -schemaname parameter when you run the Load command using this method. Otherwise, the program uses the name of the database schema obtained when the MassLoadOutputFiles directory and its files were originally created.When you use the loadonly method, errors and other messages are saved in files that have a .log extension. These log files are written to the MassLoadOutputFiles directory specified for the -directory parameter.

Use the loadonly method **only** for instance creation. If you use it at any other time, the result may not be desirable.

-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 process (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. The default file is MassLoadCustomizer.properties. The customizer property file can be specified as shown in the following example:

```
-customizer WC_installdir\my_directory\file_name.properties
```

If this file exists in the current directory, the same file can be specified as shown in the following example:

```
-customizer file_name.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 file_name
```

where, *my_directory* is a user defined directory and *file_name* is the name of the property file that you are going to use. For more information on creating and specifying a new customizer property file, refer to the most recent version of the WebSphere Commerce Production and Development online help

-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 KEYS table in the database.

Methods for the Load command:
Before loading data, you should determine which of the available 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:

- 400 The load method cannot insert or update data in bit data fields or DBCLOB fields. Only new records are inserted into the database; existing records will cause an error.
- DB2 With the load method, only new records are inserted into the database; existing records are not updated. The load method can only be used for local, not remote, DB2 databases.

Import method:
AIX Linux Solaris Windows 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.
- 400 The import method cannot insert or update data in bit data fields or DBCLOB fields and can be used only on local databases
- 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.

Createonly method for AIX, Linux, Solaris, and Windows systems:
To improve performance during instance creation, use the createonly method. Use the createonly method to create mass-load data (MLD) files without loading the data into the database. You can later use your native database load utility to load the MLD files that you created into a WebSphere Commerce database by running the Load command using the loadonly method.

Loadonly method for AIX, Linux, Solaris, and Windows systems:
Use the loadonly method to load MLD files that were created using the createonly method. Use the loadonly method **only** for instance creation. If you use it at any other time, the result may not be desirable.

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 WebSphere Commerce Production and Development online help.
- **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. After taking into consideration the size of the `input.xml` file, you may use a commit count larger than the number of records in your file. This enables rollback of the entire `input.xml` file if an error occurs.

  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 WebSphere Commerce Production and Development online help.

For detailed information on setting up and customizing the files used to run this command, refer to the WebSphere Commerce Production and Development online help.

Loader package commands for transforming and extracting data

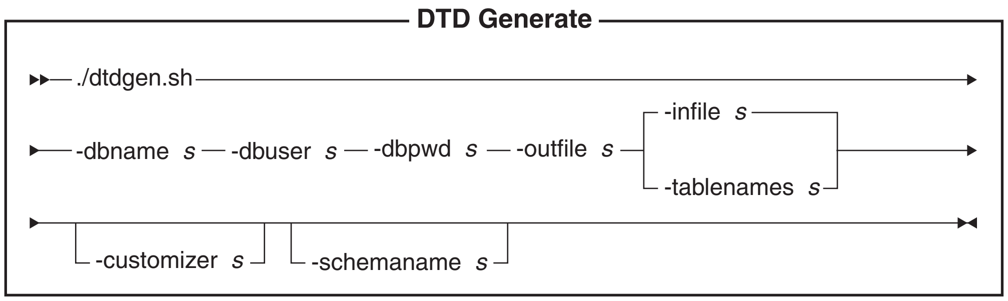
DTD Generate command

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 *WC_installdir*/xml/sar directory.

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.

AIX Linux 400 Solaris



Notes:

1. The above diagram is intended primarily as a reference for the command parameters.
2. File names specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

-dbname

AIX Linux Solaris Name of the target database

400 This is the database name as displayed in the relational database directory (WRKRDBDIRE)

-dbuser

AIX Linux Solaris Name of the user connecting to the database

400 This is usually the same as the instance user name

-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 ("")

-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:

AIX Linux Solaris `DB2ConnectionCustomizer.properties`

400 `ISeries_GENWCSDTD_Customizer.properties`

If you have configured your instance to use the toolbox driver, then use the `Toolbox_GENWCSDTD_Customizer` customizer file provided for the toolbox driver. You must also specify the hostname for the -dbname parameter. The following is an example of invoking the `dtdgen.sh` script:

```
 ./dtdgen.sh -dbname MY.HOSTNAME.CA -dbuser instance -dbpwd mypass
-outfile /path/out.dtd
  method sqlimport -customizer Toolbox_GENWCSDTD_Customizer
 -infile /path/file.xml
```

The customizer property file can be specified as shown in the following example:

AIX Linux Solaris

-customizer *WC_installdir/my_directory/file_name*.properties

-customizer *WC_installdir/my_directory/file_name*

400

-customizer *WC_userdir/my_directory/file_name*.properties

-customizer *WC_userdir/my_directory/file_name*

If this file exists in the current directory, the same file can be specified as shown in the following example:

-customizer *file_name*.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:

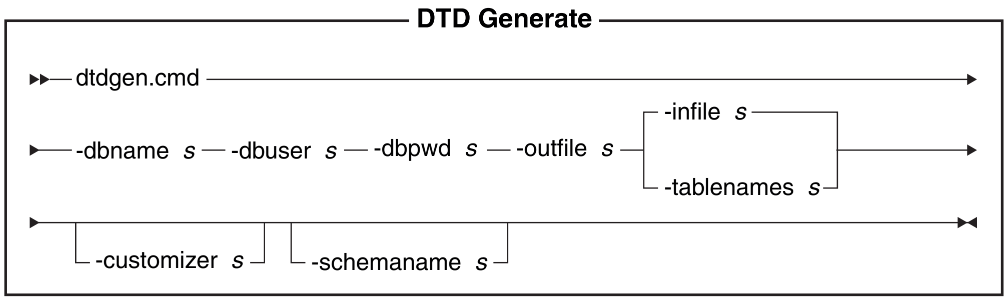
```
-customizer file_name
```

where, *my_directory* is a user defined directory and *file_name* is the name of the property file that you are going to use. For more information on creating and specifying a new customizer property file, refer to the WebSphere Commerce Production and Development online help.

-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 owner of the table in the database. 400 If neither a command-line nor a property-file specification for this parameter exists, the command defaults to the name of the database user.

Windows



Note: File names 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

-infile Name of an input file containing a database-table name on each line

-tablenames
Names of tables separated by commas

-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 WC_installdir\my_directory\file_name.properties
-customizer WC_installdir\my_directory\file_name
```

If this file exists in the current directory, the same file can be specified as shown in the following example:

```
-customizer file_name.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 file_name
```

where, *my_directory* is a user defined directory and *file_name* is the name of the property file that you are going to use. For more information on creating and specifying a new customizer property file, refer to the WebSphere Commerce Production and Development online help.

-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 owner of the table in the database.

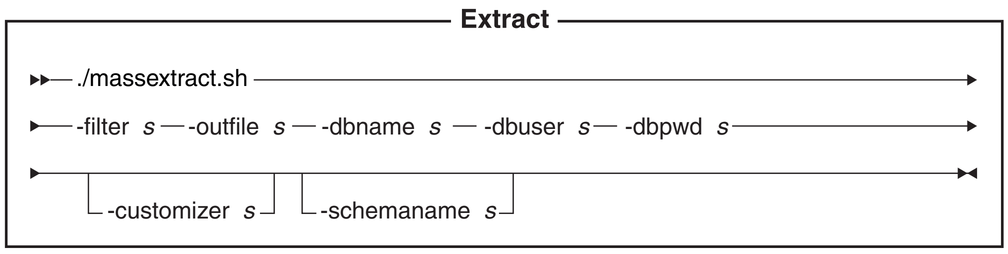
For detailed information on setting up and customizing the files used to run this command, refer to the WebSphere Commerce Production and Development online help.

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.

AIX Linux 400 Solaris



Notes:

1. The above diagram is intended primarily as a reference for the command parameters.

2. File names 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

AIX Linux Solaris Name of the database from which data is being extracted

400 This is the database name as displayed in the relational database directory (WRKRDBDIRE)

-dbuser

AIX Linux Solaris Database user name for the database from which data is being extracted

400 This is usually the same as the instance user name

-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. The default file is:

AIX Linux Solaris DB2ConnectionCustomizer.properties

400 ISeries_EXTWCSDTA_Customizer.properties If you have configured your instance to use the toolbox driver, then use the Toolbox_EXTWCSDTA_Customizer customizer file provided for the toolbox driver. You must also specify the hostname for the -dbname parameter. The following is an example of invoking the massextract.sh script:

```
./massextract.sh -dbname MY.HOSTNAME.CA -dbuser instance -dbpwd mypass
 -filter/path/filter.xml
 -outfile /path/file.xml -customizer Toolbox_EXTWCSDTA_Customizer
```

The customizer property file can be specified as shown in the following example:

AIX Linux Solaris DB2ConnectionCustomizer.properties

-customizer *WC_installdir/my_directory/file_name*.properties

-customizer *WC_installdir/my_directory/file_name*

400

-customizer *WC_userdir/my_directory/file_name*.properties

-customizer *WC_userdir/my_directory/file_name*

If this file exists in the current directory, the same file can be specified as shown in the following example:

-customizer *file_name*.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:

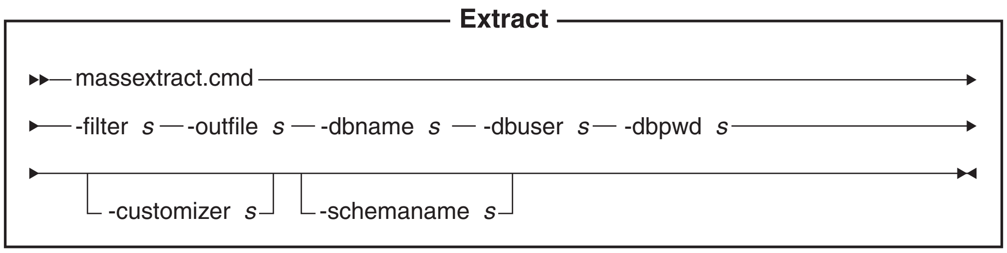
```
-customizer file_name
```

where, *my_directory* is a user defined directory and *file_name* is the name of the property file that you are going to use. For more information on creating and specifying a new customizer property file, refer to the WebSphere Commerce Production and Development online help.

-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 If neither a command-line nor a property-file specification for the -schemaname parameter exists, the command defaults to the value of the -dbuser parameter.

Windows



Note: File names 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 WC_installdir\my_directory\file_name.properties
-customizer WC_installdir\my_directory\file_name
```

If this file exists in the current directory, the same file can be specified as shown in the following example:

```
-customizer file_name.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 file_name
```

where, *my_directory* is a user defined directory and *file_name* is the name of the property file that you are going to use. For more information on creating and specifying a new customizer property file, refer to the WebSphere Commerce Production and Development online help.

-schemaname
Name of the database schema from which data is being extracted. This parameter is optional.

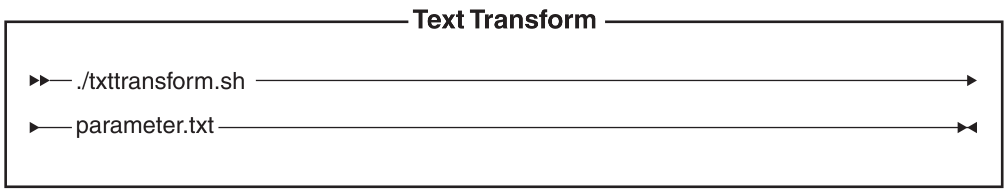
If this parameter is not specified when running the command, the command looks for a name=value pair in the customizer property file that specifies the value of SchemaName. If this pair is present in the property file, the command uses the value specified. If neither a command-line nor a property-file specification for this parameter exists, the command defaults to the schema name of the table in the database.

For more information on this command, refer to the WebSphere Commerce Production and Development online help.

Text Transform command

This command transforms data between a character-delimited variable format and an XML format.

AIX Linux 400 Solaris



Note: The above diagram is intended primarily as a reference for the command parameters.

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

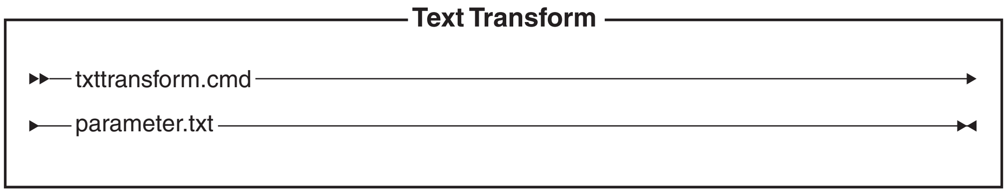
schema file
Name of the XML schema file to be used in the transformation

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.

Windows



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.

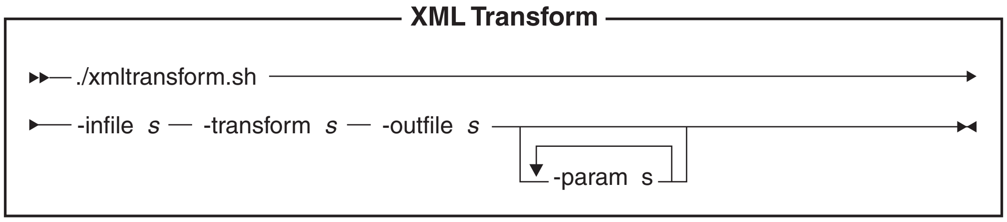
Note: This file is also referred to as a ″manifest″ or ″command″ file. It can contain multiple lines of four parameters each.

For more information on this command, refer to the WebSphere Commerce Production and Development online help.

XML Transform command

This command converts an XML file into an alternate XML format.

AIX Linux 400 Solaris



Notes:

1. The above diagram is intended primarily as a reference for the command parameters.
2. File names specified as parameters for this command can be preceded by relative or absolute paths.

Parameter values:

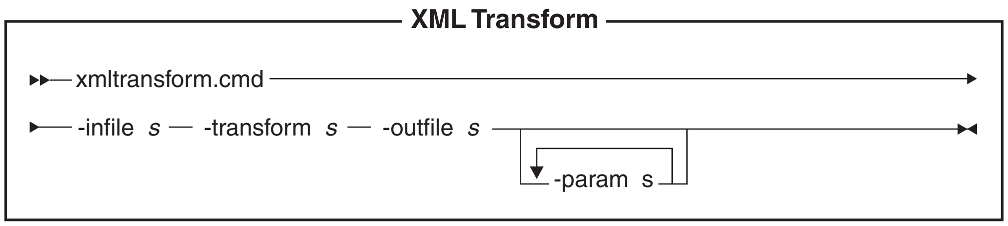
-infile Name of the file to be transformed

-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.

Windows



Note: File names 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.

For more information on this command, refer to the WebSphere Commerce Production and Development online help.

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 WebSphere Commerce Production and Development online help.

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 WebSphere Commerce Production and Development online help.

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 environment. For information on running these commands in other environments, refer to the WebSphere Commerce Production and Development online help.
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 WebSphere Commerce Production and Development online help.

Refer to Chapter 38, “Loading WebSphere Commerce database asset groups,” on page 383 and “Loading database asset groups” on page 390 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 *WC_installdir*/bin directory within WebSphere Commerce.

The scripts and commands are as follows:

- AIX Linux 400 Solaris

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

- 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

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 346 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 347.

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 installed on a Windows machine. If you have WebSphere Commerce installed on a different operating system, please substitute the appropriate values for your operating system.

1. Create a working directory.

   For this example, create the *WC_installdir*/test directory.

   Note: If you do not use *WC_installdir*/test as your working directory, substitute the name and path of the working directory that you do use for *WC_installdir*/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 WC_installdir\samplestores\
   B2BDirect\B2BDirect.sar WC_installdir\test
   ```

   This copies the `B2BDirect.sar` file into `WC_installdir\test`.

b. From a Windows command prompt, enter the following command:

```
cd to WC_installdir\test
```

c. Do one of the following:

- If you have Java installed, enter the following command from the Windows command prompt:

  ```
  jar -xvf B2BDirect.sar
  ```

  This extracts the business direct ToolTech sample store XML files into drive:\WebSphere\CommerceServer55\test.
- Use any up-to-date unzipper product (such as WinZip or PKZIP) to extract the entire contents of *WC_installdir*\samplestores\B2BDirect\B2BDirect.sar into *WC_installdir*\test.

This extracts the Business ToolTech sample store XML files into *WC_installdir*\test\WEB-INF\stores\BusinessDirect\data\ToolTech\data.

d. From the Windows command prompt, enter the following command:

```
copy WC_installdir\xml\sar\store.dtd
WC_installdir\test\WEB-INF\stores\BusinessDirect\data\ToolTech\data
```

This copies the store.dtd file into *WC_installdir*\test\WEB-INF\stores\BusinessDirect\data\ToolTech\data.

e. From the Windows command prompt, enter the following command:

```
copy WC_installdir\xml\sar\DBLoadMacros.dtd
WC_installdir\test\WEB-INF\stores\BusinessDirect\data\ToolTech\data
```

This copies the DBLoadMacros.dtd file into *WC_installdir*\test\WEB-INF\stores\BusinessDirect\data\ToolTech\data.

f. From the Windows command prompt, enter the following command:

```
copy WC_installdir\xml\sar\fulfillment.dtd
WC_installdir\test\WEB-INF\stores\BusinessDirect\data\ToolTech\data
```

This copies the fulfillment.dtd file into *WC_installdir*\test\WEB-INF\stores\BusinessDirect\data\ToolTech\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. To resolve identifiers for the fulfillment.xml file, do the following:

   a. Edit the fulfillment.xmlfile to include the following:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE  fulfillment-asset SYSTEM "fulfillment.dtd">
   <fulfillment-asset>
   </fulfillment-asset>
   ```

   The fulfillment.xml should look like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE  fulfillment-asset SYSTEM "fulfillment.dtd">
<fulfillment-asset>
<!--defaultshipoffset can be overridden in the STORITMFFC table.-->
<!--Now in ToolTech STORITMFFC.shippingoffset is set to 86400
  seconds which is one day--<
<ffmcenter
   ffmcenter_id="@ffmcenter_id_1"
   member_id="&MEMBER_ID;"
    name="ToolTech Home"
    defaultshipoffset="0"
    markfordelete="0"
 />
</fulfillment-asset>
```

b. Edit the `DBLoadMacros.dtd` file to include the following, if it is missing:

```
<!ENTITY MEMBER_ID "-2001">
```

c. 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 WC_installdir\test\WEB-INF\stores\BusinessDirect\data\ToolTech\data\
fulfillment.xml
-outfile WC_installdir\test\WEB-INF\stores\BusinessDirect\data\ToolTech\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.

5. To resolve identifiers for the `store.xml` file, do the following:

   a. Edit the `store.xml`file to include the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE store-asset SYSTEM "store.dtd">
<store-asset>
</store-asset>
```

   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 `*WC_installdir*\test\WEB-INF\stores\BusinessDirect\data\ToolTech\data`

   c. Edit the `DBLoadMacros.dtd` file to include the following, if it is missing:

```
<!ENTITY MEMBER_ID "-2001">
<!ENTITY STORE_IDENTIFIER "ToolTech">
<!ENTITY STORE_DIR "ToolTech">
```

d. Enter the following command:

```
idresgen -dbname mall -dbuser db2admin -dbpwd db2admin
-infile WC_installdir\test\WEB-INF\stores\
BusinessDirect\data\ToolTech\data\store.xml-outfile
WC_installdir\test\WEB-INF\stores\
BusinessDirect\data\ToolTech\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.

6. In the store.xml file, you will find the following element:

   ```
   <storeent
      STOREENT_ID="@storeent_id_1"
      MEMBER_ID="@seller_b2b_mbr_id"
      TYPE="S"
      IDENTIFIER="&STORE_IDENTIFIER"
      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.
7. 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 370.

   For this example, enter the following command from a Windows command prompt:

   ```
   cd WC_installdir\bin
   ```

   where *WC_installdir*\bin should be changed to the name of the directory containing the ID Resolve command idresgen.cmd if it is not located in *WC_installdir*\bin on your system.
8. From the Windows command prompt, enter the following command:

   ```
   idresgen -dbname mall -dbuser wcs -dbpwd wcs1 –infile
   WC_installdir\test\WEB-INF\stores\BusinessDirect\data\ToolTech\
   data\store.xml -outfile
   WC_installdir\test\WEB-INF\stores\BusinessDirect\data\ToolTech\
   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.

You can resolve identifiers using one of the following options:

- Option 1:
  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 2: Load the store-assets data group using the process described in "Loading database asset groups" on page 390.

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`. To modify and use IdResolveKeys.properties, copy this file into a user defined directory, make the required changes and then specify this new file when invoking the ID Resolve command. The default IdResolveKeys.properties file is located in the *WC_installdir*/properties directory:

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 `*WC_installdir*\test\WEB-INF\stores\ BusinessDirect\data\ToolTech\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 339.

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 371.

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 `*WC_installdir*\test\WEB-INF\stores\BusinessDirect\data\ToolTech\data\` that you created in "Examples of resolving identifiers" on page 371.
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 371 is in `*WC_installdir*\test\WEB-INF\stores\BusinessDirect\data\ToolTech\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 370.

   For this example, enter the following command from a Windows command prompt:

   ```
   cd WC_installdir\bin
   ```

   where *WC_installdir*`\bin` should be changed to the name of the directory containing the Load command `massload.cmd` if it is not located in *WC_installdir*`\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
WC_installdir\test\WEB-INF\stores\BusinessDirect\data\ToolTech
\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 the Administration Console and described in "Database asset loading sequence" on page 383, the `store1.xml` file created in "Examples of resolving identifiers" on page 371 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 371 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 (*`WC_installdir`*`\bin` in the above example) by default. For ▸ 400 the trace.txt is created in `QIBM/Userdata/CommerceServer55/instances/`*`instance_name`*`/logs.` 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 WebSphere Commerce Production and Development online help.

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 349.

Chapter 38. 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 37, "Overview of loading store data," on page 335, 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 C, "Database asset groups," on page 441. 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 441.

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 396 for more information. You should load the contract assets after the other database asset groups.

Refer to Appendix C, "Database asset groups," on page 441 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 stores are available with WebSphere Commerce 5.5. 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 439.
   b. Appendix C, “Database asset groups,” on page 441, as you need to know which WebSphere Commerce database asset files and database tables are affected.
   c. Chapter 37, “Overview of loading store data,” on page 335, 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 390 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 contained within the WebSphere Commerce sample store archives. The sample store archives are organized by business model in the *WC_installdir*/samplestores directory. 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 *WC_installdir*/schema/xml directory. 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. 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 "store-data-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.
- `store-data-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 fragment file. The path and location of the database asset are defined in the database asset group DTD file. The name of the *&database asset*.xml; file will change to match the database asset 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 % ForeignKeys.dtd SYSTEM "ForeignKeys.dtd">
%ForeignKeys.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.
- `ForeignKeys.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. Examples of such files are contained within the WebSphere Commerce sample store archives. The sample store archives are organized by business model in the *WC_installdir*/samplestores directory.
- The *`locale_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 ForeignKeys.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 339.
8. Run the Load command on the resolved data file as described in "Load command" on page 349. 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:
   - AIX Linux Solaris Windows *WC_installdir*/logs
   - 400 *WC_userdir*/instances/*instance_name*/logs
9. Business Run the AccountImport command as described in "Publishing business account assets" on page 396.
10. If applicable, publish contracts as described in "Publishing contract assets" on page 396.
11. Complete the tasks in "Publishing storefront assets and store configuration files by copying to the WebSphere Commerce Server" on page 399

Loading database asset groups

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 439
   b. Appendix C, “Database asset groups,” on page 441, as you need to know which WebSphere Commerce asset files and database tables are affected.
   c. Chapter 37, “Overview of loading store data,” on page 335, 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 385 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 contained within the WebSphere Commerce sample store archives. The sample store archives are organized by business model in the *WC_installdir*/samplestores directory. 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 *WC_installdir*/schema/xml directory.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"?>. You must ensure that the encoding you specify matches the actual encoding of the file.

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 % ForeignKeys.dtd SYSTEM "ForeignKeys.dtd">
   %ForeignKeys.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.
- `ForeignKeys.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 *WC_installdir*/samplestores directory.
- The *path*`_store.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 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 `ForeignKeys.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.

Notes:

a. 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;`.

b. If you are referencing a member ID that already exists in the database, then you can replace the internal alias used in the sample store data with `&MEMBER_ID;`. If not, you can include the XML needed to resolve the member ID using `@member_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 339.
8. Run the Load command on the resolved data file as described in "Load command" on page 349. 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:
   - AIX Linux Solaris Windows *WC_installdir*/logs
   - 400 *WC_userdir*/instances/*instance_name*/logs
9. Business Run the AccountImport command as described in "Publishing business account assets" on page 396.
10. If applicable, publish contracts as described in "Publishing contract assets" on page 396.
11. Complete the tasks in "Publishing storefront assets and store configuration files by copying to the WebSphere Commerce Server" on page 399

Chapter 39. 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 the Administration Console or from the command line, as part of the publishing a complete store option, described in Chapter 36, "Publishing a complete store," on page 321, 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.

Note: For more information on these commands, see the WebSphere Commerce Production and Development 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 Production 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 Administration Console 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 Administration Console 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 Administration Console or the command line

You can publish business accounts and contracts using the Administration Console, or using the publish utility from the command line. In order to publish the business accounts and contracts using either the Administration Console 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 9, "Packaging your store," on page 311.

For detailed step-by-step instructions on publishing assets using the Administration Console or the command line, see the WebSphere Commerce Production 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.

Publishing business account assets

To publish the business account assets obtained from the sample stores, do the following:

1. Copy `ForeignKeys.dtd` to the following location:
   - *WC_installdir*/xml/trading/dtd
   - ▶ 400 *WC_userdir*/instances/*instance_name*/xml/trading/xml

   ForeignKeys.dtd contains the entity values referenced by *businessaccount.xml*.
2. Copy `businessaccount.xml` to the following location:
   - *WC_installdir*/xml/trading/xml
   - ▶ 400 *WC_userdir*/instances/*instance_name*/xml/trading/xml
3. Open the Administration Console. Login as an administrator.
4. In a browser, type the following URL:
   - https://*hostname*:8002/webapp/wcs/admin/servlet/AccountImport?fileName=businessaccount.xml&URL=
     The URL to redirect to upon successful completion

Note: For more information on the command syntax and parameters, see the WebSphere Commerce Production and Development online help.

Publishing contract assets

To publish the contract assets obtained from the sample stores, do the following::

1. Copy `ForeignKeys.dtd` to the following location:
   - *WC_installdir*/xml/trading/dtd
   - ▶ 400 *WC_userdir*/instances/*instance_name*/xml/trading/xml

   ForeignKeys.dtd contains the entity values referenced by *businessaccount.xml*.
2. Copy `contract.xml` to the following location:
   - *WC_installdir*/xml/trading/xml
   - ▶ 400 *WC_userdir*/instances/*instance_name*/xml/trading/xml
3. Open the Administration Console. Login as an administrator.
4. In a browser, type the following:
   - https://*hostname*:8002/webapp/wcs/admin/servlet/ContractImportApprovedVersion?fileName=contract.xml
     &xsd=false&URL=ContractDisplay

5. If your store contains multiple `contract.xml` files (for example, locale specific contract files), repeat steps 1 through 4 for each `contract.xml` file.

Chapter 40. Publishing storefront assets and store configuration files

Publishing the storefront 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 storefront assets using the Administration Console or from the command line, as part of the publishing a complete store option, described in Chapter 36, "Publishing a complete store," on page 321, or you can publish the storefront 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 and you plan to change the store flow, you will also need to publish the store configuration files that are part of that store archive. The XML configuration files for the store are located in the following directory in the store archive:

WEB-INF/xml/tools/stores/StoreDirectory/devtools/flow

The properties file for the Change Flow tooling are located in the following directory in the store archive:

StoreDirectory/devtools/flow/ui

Publishing storefront assets and store configuration files using the Administration Console or the command line

You can publish the storefront assets and store configuration files using the Administration Console, or using the publish utility from the command line.

In order to publish the store front assets and store configuration files using either the Administration Console or the command line, the storefront assets and store configuration files must be packaged in the store archive format. For more information on packaging the storefront assets as a store archive, see Part 9, "Packaging your store," on page 311.

Publishing storefront 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 storefront 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. The store configuration files must be copied to the appropriate locations under the stores or tools web modules.

Note: When you unzip the sample store archives into the Stores web module, (retaining the path structure) the file assets will be placed in the correct location. However, the properties files used by the Change Flow tool (StoreDirectory/devtools/flow/ui/*.properties) must be copied to the ToolsStoresPropertiesPath defined in the WebSphere Commerce configuration file, `instance_name.xml`.

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:
   - *WAS_installdir*/installedApps/*cell_name*/*WC_instance_name*.ear/Stores.war/*storedir*
   - 400 *WAS_userdir*/*WAS_instance_name*/installedApps/*cell_name*/WC_*instance_name*.ear/Stores.war/*storedir*

   where *storedir* is the value of the DIRECTORY column from the STORE database table.
2. Copy the resource bundles and properties files to the application properties path:
   - *WAS_installdir*/installedApps/*cell_name*/WC_*instance_name*.ear/Stores.war/WEB-INF/classes/*storedir*
   - 400 *WAS_userdir*/*WAS_instance_name*/InstalledApps/*cell_name*/*WC_instance_name*.ear/Stores.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:
   - *WC_installdir*/instances/*instance_name*/xml
   - 400 *WC_userdir*/instances/*instance_name*/xml

   The store configuration files are copied to the following locations:
   - The store configuration XML (WEB-INF\xml\tools\stores\storedirectory\devtools\flow) is copied to the ToolsStoresXMLPath. This path is defined in the WebSphere Commerce Configuration File, `instance_name.xml`.
   - The store configuration properties files are copied to the ToolsStoresPropertiesPath. 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 `index.jsp` file under:
     - *WAS_installdir*/installedApps/*instance_name*/WC_*instance_name*.ear/Stores.war/*storedir*
     - 400 *WAS_userdir*/*WAS_instance_name*/installedApps/*cell_name*/WC_*instance_name*.ear/Stores.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 11. 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 41, “Adding customer care to your store,” on page 405
- Chapter 42, “Adding e-Marketing Spots to your store,” on page 429

Chapter 41. Adding customer care to your store

Professional Business 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*. If the Sametime server does not use the same LDAP server as WebSphere Commerce, 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 the WebSphere Commerce Accelerator, if you create your store based on one of the following sample stores:

- Business B2B direct (ToolTech)
- Consumer direct (FashionFlow)

After publishing the store using the publish utility in the Administration Console, open the WebSphere Commerce Accelerator, select the **Stores** menu, then select **Change Flow** and enable the customer care features. For more detailed instructions, see the WebSphere Commerce Production online help, help topic "Changing store flows using WebSphere Commerce Accelerator"

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.

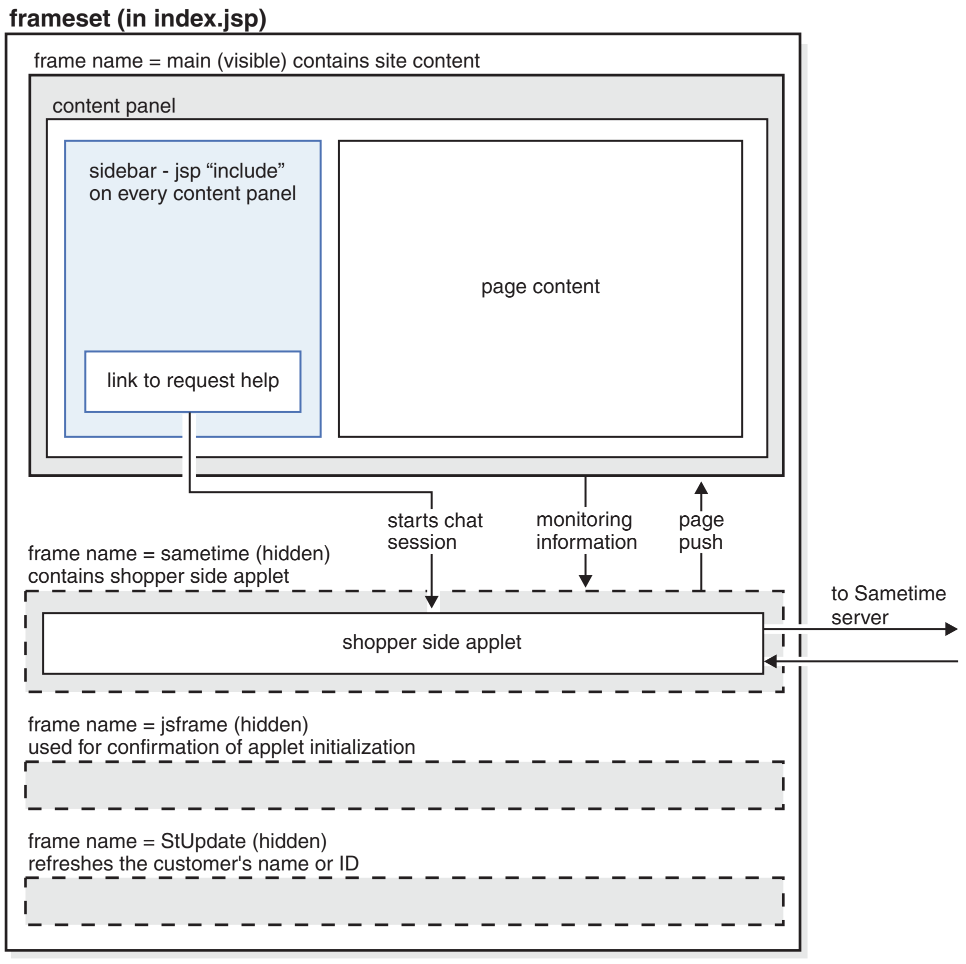
Note: The sample stores Business ToolTech and FashionFlow demonstrate how customer care should be implemented, and provide the code that you can use in your store to enable customer care. This chapter will refer to examples from these two stores to illustrate how to enable customer care in your store. Ensure that you have the latest version of the sample stores when reading this chapter.

Note: To support backward compatibility and the built-in JVM that comes with Internet Explorer and Netscape browsers version 4.x, the applet code is developed with JDK1.1 and AWT components. Therefore some of the features available with JDK 1.2 or higher (including language support, BI-direction support, and accessibility) will not be available or fully supported. Sun Java plug-ins and Netscape browser 6 and 7 are not supported.

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.

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
page content
link to request help
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 417.
- 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 information, including 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.

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. 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 URL of the frameset, not individual pages.
- Refreshing: When a customer is in the frameset and clicks refresh, they will be taken back to the original page, as coded in the frameset, which may not be the same page that is currently displaying in the browser.
- 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: In order to work correctly, pages within the customer care frameset must be able to communicate with each other. This communication is enabled by JavaScript function calls. 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 JavaScript function that calls 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). To solve this problem, you should always use the StoreFramesetView command to enter the store, as it will enforce the HTTPS connection for the frameset URL.
- In order to communicate between frames, the Java applet and JavaScript functions also must communicate. Since the applet code base is on the Sametime

server, not the WebSphere Commerce Server, some browsers (for example Netscape versions 6 and 7) that use the Sun Java plug-in prevent the JavaScript from communicating with Java applets that are loaded from a different host.

Using customer care without a frameset

Customer care will work without a frameset, however none of the monitoring customer activities will be available to the CSR. That is, information gathered from a customer is only accurate to the moment when the customer submits the help request. If after that point a customer changes pages, or adds more items to the shopping cart, the new information will not be updated for the CSR in the monitoring list.

In order to use customer care without a frameset, do the following:

1. Add the following code to the JSP file that includes the link to launch the chat page:

```
<script>
function LaunchChat()
{
<%
String pname = request.getRequestURI();
int indpn = pname.lastIndexOf('/');
indpn = pname.lastIndexOf('/', indpn-1);
if(indpn != -1) { pname = pname.substring(indpn+1);}
String headerType = (String) request.getAttribute("liveHelpPageType");
if (headerType==null) { headerType="";}
if (headerType.equals("personal"))
{
%>
 currentPageURL='PERSONAL_URL';
<% } else { %>
 currentPageURL=escape(location.href);
<% } %>
 currentpageDesc="<%=pname%>";
 chatURL="<%=com.ibm.commerce.tools.util.UIUtil.getWebappPath(request)%>
CCChatPageView?"
  + "pageURL=" + currentPageURL
  + "&pageDesc=" + currentpageDesc;
 WindowName="";
 chatAttr="toolbars=no,location=no,directories=no,status=yes,
menubar=no,scrollbars=no,resizable=no,width=360,height=400"
 window.open(chatURL,WindowName, chatAttr);
 return true;
}
</script>
```

   Note: Ensure that the width and height are the same as defined in the APPLET tag of CustomerCareChatSetup.jsp.

2. If the page you add the above code to is is a personal page, ensure you define the personal page before this code block with following statement: `request.setAttribute("liveHelpPageType", "personal");`
3. You may also want to add additional parameter value pairs in the chatURL string for customized attributes or other useful information, which can be picked up in `CustomerCareChatSetup.jsp` through the request object.
4. Add the following code to launch the chat page from a link or an image:

```
<A HREF="javascript:void(LaunchChat());">
<FONT COLOR="#ffffff" STYLE="font-size : 8pt">
<%= tooltechtext.getString("LiveHelp")%></FONT></A>
```

5. (Optional) Customize the `CustomerCareChatSetup.jsp` file to pass in additional customer information. The following applet parameters allow you to customize

the `CustomerCareChatSetup.jsp`.

Table 16.

Customizable Parameter Name	**Description**	**Note**
CHAT_FONT_SIZE	Font size to be used in the chat area.	Default value is 12.
CHAT_FONT_COLOR	Font color to be used for incoming messages in the chat area.	Default is blue (#0000FF).
CHAT_NAME_LENGTH	Length of characters that are reserved for displaying user names in the chat area.	Default is 15.
WAIT_RANGE_1	Integer value, if the number of waiting customer in the store is less than this value, the waiting message 1 displays. Otherwise the waiting message according to the WAIT_RANGE_2 setting displays.	Use -1 when only message 1 will be displayed.
WAIT_RANGE_2	Integer value, if the number of customers waiting in the store is less than this value, but greater than WAIT_RANGE_1, waiting message 2 displays. Otherwise the waiting message according to the WAIT_RANGE_3 setting displays.	Ignored if WAIT_RANGE_1 is -1. Use -1 to disable this range.
WAIT_RANGE_3	Integer value, if the number of customers waiting in the store is less than this value, but greater than WAIT_RANGE_2, waiting message 3 displays. Otherwise waiting message 4 displays.	Ignored if WAIT_RANGE_2 is -1. Use -1 to disable this range.
contentFrame	Name of the frame that is used for regular WebSphere Commercestore pages.	Default is ″_blank″. When the CSR sends back a URL, it will always launch a new browser window.
COUNTER_UNIT_WAIT	Integer value, indicates how frequently the wait counter should increased by 1.	Default is 30 seconds. Ensure it is the same as defined in the store’s CustomerCareMonitorList.jsp file.
WIDTH	Preferred width of the chat frame in pixel.	The default width is 360 pixels. The length of the invitation message will affect the finial width.
HEIGHT	Height of the chat frame in pixels.	The default is 400 pixels.

Table 16. (continued)

QUEUE_ID	Customer Care queue ID	Providing a valid queue ID will put the help request directly into the queue.
ML_ATTRIBUTES	List of customized monitoring attribute IDs, separated by commas.	Example: <PARAM name=″ML_ATTRIBUTES″ value=″8001,9002″>
ML_ATTRIBUTE_xxxx	Provide the value of customized monitoring attribute xxxx (xxxxx is the attribute ID)	Example <PARAM name=″ML_ATTRIBUTE_8001″ value=″A value string ″> Ensure the attribute ID has already been defined in ML_ATTRIBUTES parameter, or this parameter will be ignored.

Defining Customer Care

Before using customer care in your store, the following information must be defined:

- Defining the store's monitoring list
- Defining the store's topic list
- Defining the store's URLs

The following sections explain these concepts, using examples from the sample stores to illustrate how information is defined.

Defining the store's monitoring list

Each store contains a file that defines the items to be monitored by the CSR. The file, `CustomerCareMonitorList.jsp`, is located in the following directory:

- *WAS_installdir*/installedApps/*instance_name*/WC_*instance_name*.ear/Stores.war /*storedir*/CustomerServiceArea/CollaborationSection
- 400 *WAS_userdir*/WAS_*instance_name*/installedApps/*cell_name*/ WC_*instance_name*.ear/Stores.war/*storedir*/CustomerServiceArea/ CollaborationSection

`CustomerCareMonitorList.jsp` is loaded into the CSR applet when the WebSphere Commerce Accelerator calls the CCMonitorListView command. This command returns a JSP file that returns the following XML string, which defines the monitoring information for the store:

```
<?xml version="1.0" encoding="UTF-8"?>
<WCS_CUSTOMERCARE>
      <MONITORING_LIST>
         <ATTR ID="4" LABEL="MonitoringVisitorsTableCurrentPage" ></ATTR>
         <ATTR ID="3010" LABEL="MonitoringVisitorsTableInSite" UNIT="30" ></ATTR>
         <ATTR ID="3011" LABEL="MonitoringVisitorsTableInPage" UNIT="30" ></ATTR>
         <ATTR ID="6" LABEL="MonitoringVisitorsTableCart" ></ATTR>
      </MONITORING_LIST>
</WCS_CUSTOMERCARE>
```

The attribute element (ATTR), as illustrated above, defines a monitoring attribute for the customer.

Table 17.

Attribute Name	Description	Note

Table 17. (continued)

ID	The Sametime attribute ID that is used to track this item.	required
Label	The label key for the description of this item in the properties file. The corresponding description will display in the CSR applet.	required
Unit	The integer value that indicates after how many seconds that the counter will increase by 1.	optional. Used by predefined counter attributes only.

When the CSR launches the CSR applet, the applet loads the XML string, creating a monitoring list. The monitoring list displays the string value associated with the corresponding Sametime attribute. If necessary, JavaScript functions retrieve the attribute values from appropriate store pages.

Note: If the XML string contains an error, or the `CustomerCareMonitorList.jsp` file cannot be located, the CSR applet uses the default monitoring list, which only includes the customer's name.

Example CustomerCareMonitorList.jsp: The following code example is from the B2B direct sample store, ToolTech:

```
<%
// Create XML string
String strCfg=ECLivehelpConstants.EC_CC_XML_HEADER
      + LiveHelpConfiguration.getOpenTagString(ECLivehelpConstants.EC_CC_XML_ROOT)
     + LiveHelpConfiguration.getOpenTagString(ECLivehelpConstants.
EC_CC_XML_MONITORING_LIST);
%>

<%  // modify this block to customize monitoring list
      strCfg=strCfg + LiveHelpConfiguration.getMonitorAttributeElementString
(ECLivehelpConstants.EC_CC_ST_ATTR_PAGE_URL, "MonitoringVisitorsTableCurrentPage")
 + LiveHelpConfiguration.getCloseTagString(ECLivehelpConstants.
EC_CC_XML_MONITORING_ATTR);
      strCfg=strCfg + LiveHelpConfiguration.getMonitorCounterAttributeElementString
(ECLivehelpConstants.EC_CC_ST_ATTR_SITE_COUNTER,
"MonitoringVisitorsTableInSite","30")
 + LiveHelpConfiguration.getCloseTagString(ECLivehelpConstants.
EC_CC_XML_MONITORING_ATTR);
     strCfg=strCfg + LiveHelpConfiguration.getMonitorCounterAttribute
ElementString
(ECLivehelpConstants.EC_CC_ST_ATTR_PAGE_COUNTER,
 "MonitoringVisitorsTableInPage", "30")
 + LiveHelpConfiguration.getCloseTagString(ECLivehelpConstants.
EC_CC_XML_MONITORING_ATTR);
     strCfg=strCfg  + LiveHelpConfiguration.getMonitorAttributeElementString
(ECLivehelpConstants.EC_CC_ST_ATTR_CART_ITEMS,
"MonitoringVisitorsTableCart")
 + LiveHelpConfiguration.getCloseTagString
(ECLivehelpConstants.EC_CC_XML_MONITORING_ATTR);
%>

<%
strCfg=strCfg
 + LiveHelpConfiguration.getCloseTagString
```

```
(ECLivehelpConstants.EC_CC_XML_MONITORING_LIST)
 + LiveHelpConfiguration.getCloseTagString
(ECLivehelpConstants.EC_CC_XML_ROOT);
%>
```

The sample uses the utility methods of the LiveHelpConfiguration class to ensure the correctness of the XML. In order to avoid parsing problems, the sample also uses encoded attribute values. The following table provides more detail on the methods.

Table 18.

Method	**Description**	**Notes**™
static String getOpenTagString(String tagName)	returns an open tag string	LiveHelpConfiguration. getOpenTagString(″HELLO″) returns string <HELLO>
static String getCloseTagString(String tagName)	returns a closed tag string	LiveHelpConfiguration. getCloseTagString (″HELLO″) returns string </HELLO>
static String getMonitorAttribute ElementString(String sAttributeId, String sLabelKey)	returns an ATTR element string for the monitoring list	LiveHelpConfiguration. getMonitorAttribute ElementString(″1234″, ″myLabel″) returns string <ATTR ID=″1234″ LABEL=″myLabel″ >
static String getMonitorCounter AttributeElementString (String sAttributeId, String sLabelKey, String sUnit)	returns a Counter ATTR element string for the monitoring list	LiveHelpConfiguration .getMonitorCounter AttributeElementString (″1234″, ″myLabel″,″60″) returns string <ATTR ID=″1234″ LABEL=″myLabel″ UNIT=″60″ >

The following table lists the predefined Sametime attribute IDs that can be monitored.

Table 19.

Attribute ID	**Description**	**Label Key String**	**Notes**
ECLivehelpConstants. EC_CC_ST_ATTR_ PAGE_URL	The web page that shopper is browsing	″MonitoringVisitors TableCurrentPage″	Requires input from store pages
ECLivehelpConstants. EC_CC_ST_ATTR_ CART_ITEMS	The number of items that have been added to the shopping cart	″MonitoringVisitors TableCart″	Requires input from store pages
ECLivehelpConstants. EC_CC_ST_A TTR_SITE_COUNTER	How long the customer has been in the store site	″Monitoring VisitorsTable InSite″	Counter attribute, uses UNIT to set the updating frequency
ECLivehelpConstants .EC_CC_ST_ATTR _PAGE_COUNTER	How long the customer has been in this page	″Monitoring VisitorsTable InPage″	Counter attribute, uses UNIT to set the updating frequency, requires input from store pages to reset the counter

Table 19. (continued)

ECLivehelpConstants. EC_CC_ST_ATTR _WAIT_COUNTER	How long the customer has been waiting for the CSR	″Monitoring Visitors TableInWait″	Counter attribute, uses UNIT to set the updating frequency. Reset by a customer submitting a help request or when a customer is requeued by a CSR during chatting. It is also used to determine which customer will be served first by the CSR when the CSR uses the ″Serve Next″ button
ECLivehelpConstants. EC_CC_ST _ATTR_ REQ_QUEUE	Name of the queue that the customer's request is in	″Monitoring VisitorsTable Queue	Initial queue name, may be provided when a customer submits a help request in store page
ECLivehelpConstants. EC_CC_ ST_ATTR _REQ_CSR_NAME	Name of the CSR that is chatting with the customer	″Monitoring Visitors TableCSR″	Will be updated when a CSR accepts a request from a queue

Note: Customized monitoring items can be defined using the attribute ID greater than 8000. For example,

```
<ATTR ID="8004" LABEL="MonitoringVisitorsTableItem8004"></ATTR>
```

Defining the store's topic list

Each store contains a file that defines the topics that can be used by the CSR during the chat session with a customer. The file, `CustomerCareStoreQuestionList.jsp`, is located in the following directory:

- *WAS_installdir*/installedApps/*instance_name*/ WC_*instance_name*.ear/Stores.war/*storedir*/CustomerServiceArea// CollaborationSection
- 400 *WAS_userdir*/WAS_*instance_name*/ installedApps/*cell_name*/WC_*instance_name*.ear/ Stores.war/*storedir*/CustomerServiceArea/CollaborationSection

This JSP file returns the following XML string, which defines the chat topics for the store:

```
<?xml version="1.0" encoding="UTF-8"?>
<WCS_CUSTOMERCARE>
     <QUESTION_LIST>
          <GROUP NAME="TopicGroupName" >
         <      <QUESTION TITLE="TopicName" TEXT="Topic+Text" ></QUESTION>
         /GROUP>
    </QUESTION_LIST>
</WCS_CUSTOMERCARE>
```

The group element (GROUP), as illustrated above, defines a topic group which can have multiple subtopics. It has the following attributes:

Table 20.

Attribute Name	Description	Note
Name	If a group name is duplicate, the latest group definition is used	required

The question element (QUESTION) defines a single topic. It has the following attributes:

Table 21.

Attribute Name	Description	Note
Title	If a title is duplicate, the last title definition is used.	required
Text	The content of the topic.	required

Example CustomerCareStoreQuestionList.jsp: The following code is from the B2B direct sample store's, ToolTech, `CustomerCareStoreQuestionList.jsp`:

```
<% // Create XML string
String strCfg=ECLivehelpConstants.EC_CC_XML_HEADER
    + LiveHelpConfiguration.getOpenTagString(ECLivehelpConstants.EC_CC_XML_ROOT)
    + LiveHelpConfiguration.getOpenTagString(ECLivehelpConstants.
EC_CC_XML_QUESTION_LIST);
%>
<% //unmark this block to add Topic group/topics
 // start of Topic group block, repeat for more topic groups
 strCfg=strCfg + LiveHelpConfiguration.getTopicGroupElementString
("TopicGroupName");
  // start of Topic block, repeat for all topics in the same
group
  strCfg=strCfg + LiveHelpConfiguration.getTopicElementString("TopicName","
Topic Text")
           + LiveHelpConfiguration.getCloseTagString(ECLivehelpConstants.
EC_CC_XML_QUESTION_QUESTION);
  // end of Topic block
 strCfg=strCfg + LiveHelpConfiguration.getCloseTagString
(ECLivehelpConstants.EC_CC_XML_QUESTION_GROUP);
 // end of Topic group block
%>
<%
strCfg=strCfg + LiveHelpConfiguration.getCloseTagString
(ECLivehelpConstants.EC_CC_XML_QUESTION_LIST)
         + LiveHelpConfiguration.getCloseTagString
(ECLivehelpConstants.EC_CC_XML_ROOT);
%>
```

The sample uses the utility methods of the LiveHelpConfiguration class to ensure the correctness of the XML. The following table provides more detail on the methods.

Notes:

1. In order to avoid parsing problems, the sample also uses encoded attribute values. Unicode string is also used to avoid character corruption.
2. To support multiple languages, you may use a properties file per language, and retrieve the translated name or topic according to the language ID selected in CSR's session.

Table 22.

Method	Description	Notes
static String getOpenTagString (String tagName)	returns an open tag string	LiveHelpConfiguration. getOpenTagString (″HELLO″) returns string <HELLO>
static String getCloseTagString(String tagName)	returns a closed tag string	LiveHelpConfiguration. getCloseTagString (″HELLO″) returns string </HELLO>
static String getTopicGroupElement String (String sGroupName)	returns a GROUP element string for the topic list	LiveHelpConfiguration. getTopicGroupElementString (″myGroup″) returns string <GROUP NAME = ′myGroup′>
static String getTopic ElementString (String sTitle, String sText)	returns a QUESTION element string for the topic list	LiveHelpConfiguration. getTopicElementString (″myTitle″, ″myText″) returns <QUESTION TITLE =″myTitle″ TEXT =″myText″>

Defining the store’s URL list

Each store contains a file that defines the URLs that the CSR can send to a customer’s browser during the chat session. The file, `CustomerCareStoreURLList.jsp` , is located in the following directory:

- *WAS_installdir*/installedApps/*instance_name* /WC_*instance_name*.ear/Stores.war/*storedir*/CustomerServiceArea/ CollaborationSection
- 400 *WAS_userdir*/WAS_*instance_name*/installedApps/*cell_name* /WC_*instance_name*.ear/Stores.war/*storedir*/CustomerServiceArea/ CollaborationSection

This JSP file returns the following XML string, which contains the URL information for the store:

```
<?xml version="1.0" encoding="UTF-8"?>
<WCS_CUSTOMERCARE>
     <URL_LIST>
          <GROUP NAME="URLGroupName" >
               <PAGE NAME="IBM" URL="http%3A%2F%2Fwww.ibm.com" ></PAGE>
          </GROUP>
     </URL_LIST>
</WCS_CUSTOMERCARE>
```

The group element (GROUP), as illustrated above, defines a URL group which can have multiple URLs. It has the following attributes:

- Name: The name of the URL group. Name is a required attribute. If a group name is duplicate, the latest group definition is used.

The page element (PAGE) defines a single URL address. It has the following attributes:

- Name: The name of the URL page. Name is a required attribute. If a page name is duplicate, the latest page definition is used.
- URL: The URL address of the page. URL is a required attribute.

Example CustomerCareStoreURLList.jsp: The following code is from the B2B direct sample store's, ToolTech, `CustomerCareStoreURLList.jsp`:

```
<% // Create XML string
String strCfg=ECLivehelpConstants.EC_CC_XML_HEADER
 + LiveHelpConfiguration.getOpenTagString(ECLivehelpConstants.
EC_CC_XML_ROOT)
 + LiveHelpConfiguration.getOpenTagString(ECLivehelpConstants.
EC_CC_XML_URL_LIST);
%>

<% //unmark following block to add URL group/pages
// start of URL group block, repeat for more URL groups
strCfg=strCfg + LiveHelpConfiguration.getURLGroupElementString("URLGroupName");
 // start of URL pages block, repeat for all pages in the same group
 strCfg=strCfg + LiveHelpConfiguration.getURLPageElementString
("IBM","http://www.ibm.com")
         + LiveHelpConfiguration.getCloseTagString
(ECLivehelpConstants.EC_CC_XML_URL_PAGE);
 // end of URL pages block
strCfg=strCfg  + LiveHelpConfiguration.getCloseTagString
(ECLivehelpConstants.EC_CC_XML_URL_GROUP);
// end of URL group block
%>
<%
strCfg=strCfg + LiveHelpConfiguration.getCloseTagString
(ECLivehelpConstants.EC_CC_XML_URL_LIST)
 + LiveHelpConfiguration.getCloseTagString(ECLivehelpConstants.EC_CC_XML_ROOT);
%>
```

The sample uses the utility methods of the LiveHelpConfiguration class to ensure the correctness of the XML. The following table provides more detail on the methods.

Note: In order to avoid parsing problems, the sample also uses encoded attribute values. Unicode string is also used to avoid character corruption.

Table 23.

Method	**Description**	**Notes**
static String getOpenTagString(String tagName)	returns an open tag string	LiveHelpConfiguration .getOpenTagString (″HELLO″) returns string <HELLO>
static String getCloseTagString(String tagName)	returns a closed tag string	LiveHelpConfiguration. getCloseTagString (″HELLO″) returns string </HELLO>
static String getTopicGroup ElementString (String sGroupName)	returns a GROUP element string for the URL list	LiveHelpConfiguration. getTopicGroupElementString (″myGroup″) returns string <GROUP NAME = ′myGroup′>
static String getURLPage ElementString (String sName, String sURL)	returns a PAGE element string for URL list	LiveHelpConfiguration. getURLPageElement String (″myName″, ″myURL″) returns <QUESTION TITLE =″myName″ TEXT =″myURL″>

Monitoring customers using customer care

When a customer selects a link in the main frame, a new page is returned, setting the following chain of events in motion.

1. This new page includes the `CustomerCareHeaderSetup.jsp` file in its header.
2. `CustomerCareHeaderSetup.jsp` calls the following JavaScript function in the frameset page: parent.setPageParams().
3. parent.setPageParams() updates the variables to save the current store page information, and then calls UpdateStInfo().
4. UpdateStInfo() reloads the StUpdate frame, which then calls the CCShopperInfoUpdatePageView view command.
5. When the CCShopperInfoUpdatePageView returns, it loads the `CustomerCareInformationSetup.jsp` file, and gathers customer information including the customer name, ID, and items in the customer's shopping cart.
6. CCShopperInfoUpdatePageView then calls setCustomerName(), setShoppingCartItems() to update this customer information, and then calls changeSTAttributes() to update all the customer attributes in the applet.
7. changeSTAttributes() calls several applet methods to update the attributes values that the CSR is monitoring.
8. changeSTAttributes() then calls the changeWCSAttrs() method of the applet to send all of the updated attribute values to the Sametime system, which then notifies the CSR applet that the attribute values have been changed..

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
- Tracking customized monitoring items

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 including the `CustomerCareInformationSetup.jsp` in the store pages header file. The following code in the `CustomerCareInformationSetup.jsp` file obtains the customer's name and ID:

```
<jsp:useBean id="userRDB" class="com.ibm.commerce.user.beans.
UserRegistrationDataBean" scope="page">
<% DataBeanManager.activate(userRDB, request); %>
</jsp:useBean>
```

```
<%
String customer_name="";
customer_name=userRDB.getUserId();
if (userRDB.findUser()){
   if (userRDB.getLastName() !=null && userRDB.getLastName().
length() > 0){
         if (locale.toString().equals("ja_JP")||locale.toString().
equals("ko_KR")||locale.toString().equals("zh_CN")||locale.toString().
equals("zh_TW"))
            {customer_name = userRDB.getLastName() + " " +
userRDB.getFirstName();}
         else
            {customer_name = userRDB.getFirstName() + " " +
 userRDB.getLastName();}
      }
}
if (customer_name.equals("-1002")) {
 customer_name="";
 }
else {
 // need to check order items
 ....
}
customer_name=customer_name.trim();
%>
```

Note: Each time a customer browses a new page in the store, the customer's name or ID is refreshed.

The following code in the `CustomerCareInformationSetup.jsp` file updates the customer's name and ID:

```
<script language="javascript">
 ....
function changeSTAttributes()
{
 if (typeof top.setCustomerName == 'function') {
    top.setCustomerName(<%=userRDB.getUserId()%>, '<%=customer_name%>');
    top.setShoppingCartItems(<%=shoppingCartItems%>);
    top.changeSTAttributes();
    }
}
/script>
```

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 is `UserLogoffRouter.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 including `CustomerCareHeaderSetup.jsp` file in the header file

(HeaderDisplay.jsp). The following code in the CustomerCareHeaderSetup.jsp file obtains the page URL information and updates the shopper applet:

```
<script language="javascript">
var PageName="";
var PersonalPage=false;
<%
 String pname = request.getRequestURI();
 int indpn = pname.lastIndexOf('/');
 indpn = pname.lastIndexOf('/', indpn-1);

 if(indpn != -1)
         pname = pname.substring(indpn+1);

 String headerType = (String) request.getAttribute("liveHelpPageType");
 if (headerType==null) headerType="";

 // Determine if this is a personal page or not
 if (headerType.equals("personal"))
 {
  %>
    if (typeof parent.setPageParams == 'function') {
     PersonalPage=true;
       parent.setPageParams('PERSONAL_URL', '<%=pname%>');
       }
  <%
 }
 else
 {
  %>
    if (typeof parent.setPageParams == 'function')
      parent.setPageParams(location.href, '%=pname%>');
  <%
 }
 %>
 Pagename="<%=pname%>";
</script>
```

You should not allow CSR to see customer pages that contain personalized information, since the content viewed by CSR may not be the same as viewed by a customer. 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 should be marked as personal to avoid misleading the CSR during a chat session.

In order to mark pages as personal, that is, not available to the CSR, the sample stores include the following code in the header page, just before the CustomerCareHeaderSetup.jsp file is included.

```
<flow:ifEnabled feature="customerCare">
<%
      request.setAttribute("liveHelpPageType", "personal");
%>
</flow:ifEnabled>
```

Note: Although a CSR cannot see the content of a page marked personal by using the View Customer Page button, 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 following code in the CustomerCareInformationSetup.jsp file obtains the number of items in the shopping cart:

```
<%
JSPHelper jhelper = new JSPHelper(request);
String storeId = jhelper.getParameter("storeId");
int shoppingCartItems = 0;
%>
<
jsp:useBean id="userRDB" class="com.ibm.commerce.user.beans.
UserRegistrationDataBean" scope="page">
<% DataBeanManager.activate(userRDB, request); %></jsp:useBean>

<%
....
 // need to check order items
 OrderListDataBean orderListBean = new OrderListDataBean();
 orderListBean.setStoreId(new Integer(storeId));
 orderListBean.setOrderStatus("P");
 orderListBean.setUserId(cmdcontext.getUserId());
 DataBeanManager.activate(orderListBean, request);
 Vector pendingOrders = orderListBean.getOrders();
 for (int k=0; k< pendingOrders.size(); k++) {
  OrderAccessBean next_order = (OrderAccessBean) pendingOrders.
elementAt(k);
  OrderDataBean orderBean = new OrderDataBean();
  orderBean.setOrderId(next_order.getOrderId());
  DataBeanManager.activate(orderBean, request);
  //Get items in the order
  OrderItemDataBean [] orderItems = orderBean.getOrderItemDataBeans();
  for (int i = 0; ((orderItems != null) &&
(i < orderItems.length)); i++)
  {
   OrderItemDataBean orderItem = orderItems[i];
   shoppingCartItems += orderItem.getQuantityInEJBType().intValue();
   }
  }
....
%>
```

The following JavaScript function will update the cart information to Sametime applet:

```
<script language="javascript">
 ...
function changeSTAttributes()
{
 if (typeof top.setCustomerName == 'function') {
     top.setCustomerName(<%=userRDB.getUserId()%>,
 '<%=customer_name%>');
top.setShoppingCartItems(<%=shoppingCartItems%>);
     top.changeSTAttributes();
     }
}
</script>
```

Note: 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
```

Tracking customized monitoring items

In order to track customized monitoring items, do the following:

1. Define a monitoring attribute ID, (attribute IDs less than 8000 are reserved). For more information, see "Defining the store's monitoring list" on page 410.
2. Define a JavaScript variable in the `Sametime.js` file to save the attribute ID and value, and allow access from all the frames. For example: `var myTrackAttributeId=8001; var myTrackAttributeValue="anything";`
3. Add JavaScript code to assign or update the value for this variable in store pages. For example: `top.myTrackAttributeValue="new Value";`
4. . Call the setWcsAttribute() method of the applet to update the attribute in changeSTAttributes() function in the `Sametime.js` file:

   ```
   function changeSTAttributes()
   {

        if(CustomerAppletIsUp) {
          sametime.document.applets["InteractivePresenceApplet"].
   setWcsAttribute(myTrackAttributeId,myTrackAttributeValue);
      sametime.document.applets["InteractivePresenceApplet"].
   changeWCSAttrs();
    }
        else
            setTimeout("changeSTAttributes()",3000);
   // wait for 3 sec and try again
   }
   ```

   Note: Insert your code before the changeWCSAttrs() method.

Sending requests directly to a customer care queue

By default, when a customer requests a live chat with CSR, the customer's request will be placed into a default queue (queueId=0). It is then the CSR's responsibility to route the request to a more specific queue. However, you can also customize your store page to send requests directly to a defined customer care queues. To customize your store pages to send requests to a defined queue, do the following:

1. Create customer care queues for your store using the WebSphere Commerce Accelerator. For more information, see the WebSphere Commerce Production online help.
2. Assign these queues to CSRs. For more information, see the WebSphere Commerce Production online help.
3. Keep a record of the queue IDs of the queues you have created. For example the queue ID is 10020.

4. Use JavaScript to update the reqQueue variable in store JSPs. For example, `top.reqQueue="10020";`.

Queue information will not be automatically updated after a CSR launches the CSR applet. If a new queue is added into the system, or an existing queue is changed, the changes will not take effect until the CSR launches the CSR applet again.

If the queue ID is not recognized by a CSR applet, the queue ID will be put back into the default queue for the CSR. This behavior does not affect the queue attribute value associated with the customer.

Customizing customer care

You have the following options to customize customer care:

- Customizing applet parameters
- Customizing store messages

Customizing applet parameters

The following table lists the customizable applet parameters, which allow you to customize customer care by modifying the `CustomerCareFrameSetup.jsp` file.

Table 24.

Customizable Parameter Name	Description	Note
CHAT_FONT_SIZE	Font size to be used in CharArea	Default value is 12
CHAT_FONT_COLOR	Font color to be used for incoming message in the chat area.	Default is blue (#0000FF)
CHAT_NAME_LENGTH	Length of characters reserved for displaying user names in the chat area.	Default is 15
WAIT_RANGE_1	Integer value, if the number of waiting customer in the store is less than this value, the waiting message 1, displays. Otherwise the waiting message according to the WAIT_RANGE_2 setting displays.	Use -1 when only message 1 will be displayed.
WAIT_RANGE_2	Integer value, if the number of waiting customer in the store is less than this value, but greater than WAIT_RANGE_1, the waiting message 2 displays. Otherwise the waiting message according to the WAIT_RANGE_3 setting displays.	Ignored if WAIT_RANGE_1 is -1. Use -1 to disable this range.

Table 24. (continued)

WAIT_RANGE_3	Integer value, if the number of waiting customer in the store is less than this value, but greater than WAIT_RANGE_2 the waiting message 3, displays, otherwise the waiting message 4 displays.	Ignored if WAIT_RANGE_2 is -1. Use -1 to disable this range.
contentFrame	Name of the frame that is used for regular WebSphere Commerce store pages	Default is ″main″
COUNTER_UNIT_PAGE	Integer value that indicates how frequently the page counter will increase by 1	Default is 30 seconds. Ensure it is the same value as defined in the store's CustomerCare MonitorList.jsp file.
COUNTER_UNIT_SITE	Integer value that indicates how frequently the site counter will increase by 1	Default is 30 seconds. Ensure it is the same value as defines in the store's CustomerCareMonitorList.jsp file.
COUNTER_UNIT_WAIT	Integer value that indicates how frequently the wait counter will increase by 1	Default is 30 seconds. Ensure it is the same value as defined in the store's CustomerCareMonitorList.jsp file.
WIDTH	Preferred width of the chat frame in pixels	Default is 360 pixel. The length of the invitation message will affect the final width.
HEIGHT	Height of the chat frame in pixels	Default is 400® pixel.

Customizing store messages

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:
   - *Customer_Care_installdir*\properties
2. Make the necessary changes. The following table lists the message keys for each message.

Table 25.

Message Key	**Description**	**Notes**

Table 25. (continued)

WelcomeMessage	The first message to be displayed in the customer's applet after the CSR accepts a chat request.	
GoodbyeMessage	Message displayed in the customer applet right after the CSR or customer ends the chat session.	
PushPageMessage	Message to be displayed in the customer applet after the CSR sends a URL to the customer's browser.	
CallCSRMessage	Message to be displayed when a customer submits a help request and the applet is connecting to the Sametime server.	
NoConnectionMessage	Message to be displayed when the customer applet is unable to connect to the Sametime server or has lost connection with the Sametime server.	
StoreCloseMessage	Message to be displayed when CSRs are not available to serving the customer.	Always displayed with the StoreWorkingHour message.
StoreWorkingHour	Message to be displayed when CSRs are not available to serve the customer. Describes the store's hours of operation.	Always displayed with the StoreCloseMessage message.
WaitingMessage	Message to be displayed when a customer submits a request and the total number of waiting customers is less than WAIT_RANGE_1.	
WaitingMessage1	Message to be displayed when a customer submits a request and the total number of waiting customers is less than WAIT_RANGE_2, but greater than WAIT_RANGE_1.	
WaitingMessage2	Message to be displayed when a customer submits a request and the total number of waiting customers is less than WAIT_RANGE_3, but greater than WAIT_RANGE_2.	

Table 25. (continued)

WaitingMessage3	Message to be displayed when a customer submits a request and the total number of waiting customers is greater than WAIT_RANGE_3.	
CIWarningLabelLineOne	The first line of the invitation message that displays when a CSR initiates a chat request to a customer.	The display length of the message will affect the width of the chat frame. The line separator character should not be used.
CIWarningLabelLineTwo	The second line of the invitation message that displays when a CSR initiates a chat request to a customer.	The display length of the message will affect the width of the chat frame. The line separator should not be used.

Note: The line separator character ″\n″ should be inserted in the message to start a new line. If not, the message may exceeds the boundary of the display area.

Note: You may need to modify properties for different locales to ensure the correct translation of the message.

3. Close and save the file.

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*.
- If the Sametime server does not use the same LDAP server as WebSphere Commerce, create a CSR and register the CSR for customer care using the Administration Console. For more information, see the WebSphere Commerce Production online help.

Part 2: Copying the customer care integration files from the sample store

The sample stores FashionFlow 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.

- `CustomerCareBlankSetup.jsp:`: An empty JSP file. It is called by the CCShopperBlankPageView command, as a placeholder for the frames.
- `CustomerCareFrameSetup.jsp` : Contains JavaScript functions and embeds the customer care applet for the store front. It is called by the CCShopperFramePageView command, and loads the customer applet in the frameset..
- `CustomerCareAppletReadySetup.jsp :` : Indicates that the applet is loaded properly. It is called by the CCShopperReadyPageView command, and indicates that the customer applet is ready for Javascript functions.
- `CustomerCareHeaderSetup.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. It must be included in every page to update the customer's page URL.
- `CustomerCareInformationSetup.jsp`: Updates the customer's name information and ID. It is called by the CCShopperInfoUpdatePageView command.
- `CustomerCareMonitorList.jsp`: The monitoring list configuration file. it is called by the CSR to load monitoring list.
- `CustomerCareStoreQuestionList.jsp`: The topic list configuration file. It is called by the CSR to load the store topic list
- `CustomerCareStoreURLList.jsp`: The store level URL list configuration file. It is called by CSR to load the store URL list
- `CustomerCareChatSetup.jsp`: The setup file for using customer care without a frameset. It is called by the CCChatPageView command and launches the customer applet in the non-frameset configuration.
- `EnvironmentSetup.jsp`: Used by all customer care JSP files for store level configuration (for example, resource bundles).
- `index.jsp`: The entry page that redirects the customer's browser to the frameset page or regular catalog pages according to the customer care store flow settings.
- `StoreFramesetPage.jsp`: It is called by the StoreFramesetView command to construct a frameset for customer care.

To copy the Sametime integration files from the sample store to your store, do the following:

1. Locate the store archive file for the consumer direct (FashionFlow store) or the B2B direct (ToolTech) store. The store archive files are located in the following directory:
   - *WC_installdir*/samplestores
2. Open either the ConsumerDirect or B2BDirect folder, then select a consumerdirect.sar or B2Bdirect.sar.
3. Open the store archive file using WinZip or a similar tool.
4. Select the files listed above:
5. Extract the files to the directory that contains the web assets for your store. To maintain the same directory structure as the samples stores, you may want to create subdirectories for the following files:
   - .../CustomerServiceArea/CollaborationSection
     - CustomerCareAppletReadySetup.jsp
     - CustomerCareBlankSetup.jsp
     - CustomerCareBlankSetup.jsp
     - CustomerCareFrameSetup.jsp
     - CustomerCareInformationSetup.jsp
     - CustomerCareMonitorList.jsp

  - CustomerCareStoreQuestionList.jsp
  - CustomerCareStoreURLList.jsp
  - CustomerCareChatSetup.jsp
- /include
  - CustomerCareHeaderSetup.jsp
  - EnvironmentSetup.jsp
- ..\
  - index.jsp
  - Sametime.js
  - StoreFramesetPage.jsp

Part 3: Adding code to determine which page the customer is browsing

To determine which page the customer is browsing, do the following:

1. Include the `CustomerCareHeaderSetup.jsp` file to the store's header file, for example:
```
<%@ include file="include\CustomerCareHeaderSetup.jsp" %>
```
2. Add the following code to any pages that should be marked personal, and thus not available for access by the CSR. Ensure the following code is added before you include the `CustomerCareHeaderSetup.jsp` file.
```
<flow:ifEnabled feature="customerCare">
<%
     request.setAttribute("liveHelpPageType", "personal");
%>
</flow:ifEnabled>=incfile%>"flush="true"/>
```

Part 4: 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:

   Note: You may need to replace infashiontext with the object name used for that store resource).
```
<a href="javascript:if((parent.sametime != null))
top.interact();"><%=infashiontext.getString("LiveHelp")%> </a>
```

Part 5: Create an entry page that will redirect to the customer care frameset page

Since the frameset is required for most customer care features to function properly, the customer must call the StoreFramesetView command to activate the frameset. For an example, see index.jsp of the consumer direct or B2B direct sample store.

Chapter 42. 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 20, “Campaign assets,” on page 203 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(com.ibm.commerce.server.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() + "/";
 }

   /**
    * END - the following code should exist only once in a page, it initialize the
    *       command context and store data bean.
    */
%>

<!-- ===========================================================================================
// The following HTML form submits the request on the e-Marketing Spot to the ClickInfo
// command which captures the campaign statistics, and redirects 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>

<%
   /**
    * 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, but 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
   String emsName = request.getParameter("emsName");
   if (emsName == null) {
      emsName = "defaultEMSName";
   }
   eMarketingSpot.setName(emsName);

   // Set the catalog entry ID that is currently displaying on the page.  This is required if the
   // up-sell/cross-sell initiative is based on the content of the current page.
   String sourceCatentryId = request.getParameter("sourceCatentryId");
   if (sourceCatentryId != null) {
      eMarketingSpot.setSourceCatalogEntryId(sourceCatentryId); // use this method to set single ID
      //eMarketingSpot.setMultipleSourceCatalogEntryId(sourceCatentryId2);
         // use this method to set multiple IDs
   }

   // 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);
   eMarketingSpot.setMaximumNumberOfAssociateCatalogEntries(20);

   // instantiate the bean
   com.ibm.commerce.beans.DataBeanManager.activate(eMarketingSpot, request);
%>

<%
   // The following block is used to display the up-sell/cross-sell 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.getAssociateCatalogEntries() != null
      && eMarketingSpot.getAssociateCatalogEntries().length < 0) {
%>
<TABLE>
   <%
      for (int i=0; i<eMarketingSpot.getAssociateCatalogEntries().length; i++) {
         String associateCatalogEntryThumbNail = null;
         String associateCatalogEntryShortDescription = null;
         try {
            associateCatalogEntryThumbNail =
               eMarketingSpot.getAssociateCatalogEntries()[i].getDescription(emsCommandContext.
               getLanguageId()).getThumbNail();
            associateCatalogEntryShortDescription =
               eMarketingSpot.getAssociateCatalogEntries()[i].getDescription(emsCommandContext.
getLanguageId()).getShortDescription();
         }
         catch (Exception e) {
            // no description defined for the current language
         }
   %>
   <TR>
      <TD>
         <A HREF="/webapp/wcs/stores/servlet/ClickInfo?evtype=CpgnClick&mpe_id=<%=
            eMarketingSpot.getId() %>&intv_id=<%=
            eMarketingSpot.getAssociateCatalogEntries()[i].getInitiativeId() %>
            &URL=/webapp/wcs/stores/servlet/ProductDisplay&<%=
            com.ibm.commerce.server.ECConstants.EC_STORE_ID %>=<%=
            emsCommandContext.getStoreId().toString() %>&<%=
            com.ibm.commerce.server.ECConstants.EC_PRODUCT_ID %>=<%=
            eMarketingSpot.getAssociateCatalogEntries()[i].getCatalogEntryID() %>&<%=
            com.ibm.commerce.server.ECConstants.EC_LANGUAGE_ID %>=<%=
            emsCommandContext.getLanguageId().toString() %>">
         <IMG SRC="<%= collateralPath + associateCatalogEntryThumbNail %>"
            ALT="<%= associateCatalogEntryShortDescription %>" BORDER=0 WIDTH=60>
         </A>
      </TD>
      <TD><%= associateCatalogEntryShortDescription %></TD>
   </TR>
<%  } %>
</TABLE>
<% } %>

<%
 // 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++) {
   String catalogGroupName = null;
   String catalogGroupLongDescription = null;
         try {
            catalogGroupName = eMarketingSpot.getCategories()[i].getDescription
               (emsCommandContext.getLanguageId()).getName();
            catalogGroupLongDescription = eMarketingSpot.getCategories()[i].getDescription
               (emsCommandContext.getLanguageId()).getLongDescription();
         }
         catch (Exception e) {
            // no description defined for the current language
         }
%>
   <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&<%=
            com.ibm.commerce.server.ECConstants.EC_STORE_ID %>=<%=
            emsCommandContext.getStoreId().toString() %>&<%=
            com.ibm.commerce.server.ECConstants.EC_CATEGORY_ID %>=<%=
            eMarketingSpot.getCategories()[i].getCategoryId() %>&<%=
            com.ibm.commerce.server.ECConstants.EC_CATALOG_ID %>=<%=
            eMarketingSpot.getCategories()[i].getCatalogId() %>&<%=
            com.ibm.commerce.server.ECConstants.EC_LANGUAGE_ID %>=<%=
            emsCommandContext.getLanguageId().toString() %>">
   <%= catalogGroupName %>
   </A>
  </TD>
  <TD><%= catalogGroupLongDescription %></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++) {
    String catalogEntryThumbNail = null;
      String catalogEntryShortDescription = null;
      try {
         catalogEntryThumbNail =
            eMarketingSpot.getCatalogEntries()[i].getDescription(emsCommandContext.
            getLanguageId()).getThumbNail();
         catalogEntryShortDescription =
            eMarketingSpot.getCatalogEntries()[i].getDescription(emsCommandContext.
            getLanguageId()).getShortDescription();
      }
      catch (Exception e) {
      // no description defined for the current language
      }
%>
   <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&<%=
            com.ibm.commerce.server.ECConstants.EC_STORE_ID %>=<%=
            emsCommandContext.getStoreId().toString() %>&<%=
            com.ibm.commerce.server.ECConstants.EC_PRODUCT_ID %>=<%=
            eMarketingSpot.getCatalogEntries()[i].getCatalogEntryID()
            %>&<%= com.ibm.commerce.server.ECConstants.EC_LANGUAGE_ID %>=<%=
            emsCommandContext.getLanguageId().toString() %>">
         <IMG SRC="<%= collateralPath + catalogEntryThumbNail %>"
            ALT="<%= catalogEntryShortDescription %>" BORDER=0 WIDTH=60>
   </A>
  </TD>
  <TD><%= catalogEntryShortDescription %></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, but do not use the same spot name.
  */
%>
```

The preceding e-Marketing Spot supports four types of campaign initiatives:

- Product recommendation
- Category recommendation
- Awareness advertisement
- Merchandising association

Note: For more detailed information on each of these initiatives, see Chapter 20, “Campaign assets,” on page 203.

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 429 to a new JPS file inside of the store Web application.
4. Customize the sample e-Marketing Spot to fit the layout of your JSP file(s).
5. 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.
6. Add the e-Marketing Spot to the JSP file by dynamically including the spot using the `<jsp:include>` tag.
7. If you require more than one e-Marketing Spot per JSP file, repeat steps 2 through 6.
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. The URL parameter, CatalogDisplay should start with ″&″ instead of ″?″ because the code isn’t referencing the command directly.

c. By using the Java dynamic include tag to add the e-Marketing Spot to the store JSP, the new Dynacache feature can be enabled, so that the content of the JSP, excluding the e-Marketing Spot, will be cached. The e-Marketing Spot will be refreshed on every visit, thus displaying the proper dynamic content. Here is an example of how the dynamic include can be used to add the e-Marketing Spot to a JSP file, called `ESpot.jsp`:

```
<jsp:include page="ESpot.jsp" flush="true">
<jsp:param name="emsName" value="StoreHomePage" />
</jsp:include>
```

d. If a merchandising association initiative is being scheduled to the e-Marketing Spot, and the source of the association is based on the content of the page, it can be set using the `setSourceCatalogEntryId(String source)` and the `setMultipleCatalogEntryId(String source)` methods provided in `com.ibm.commerce.marketing.beans.EMarketingSpot`. For example, in the Product Display page, if the product displayed in the page is to be used as the source of the association, the following method will be invoked:

```
eSpot.setSourceCatalogEntryId(productId)
```

where *eSpot* is an instance of the `com.ibm.commerce.marketing.beans.EMarketingSpot` class, and *productId* is the identifier of the source product.

Part 12. 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`.

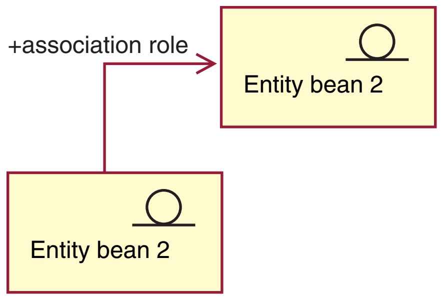
UML diagrams consist of the following items:

- 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 end of the line can be ″associated″ of the class on the other end of the line..
- 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:

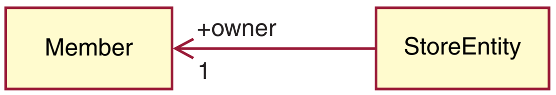
Cardinality number	Relationship type
1	one and only one
0..1	zero or one
0..n	zero or more
1..n	one or more

  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.

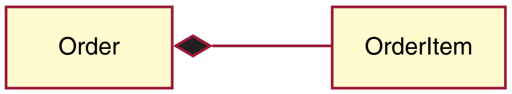
The following diagrams illustrate the above concepts:



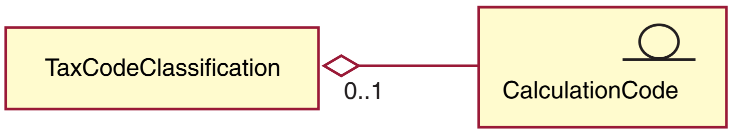
This diagram shows two entity beans with the decoration stereotype symbol indicating an Enterprise Java Bean. There is a unidirectional association from the first bean to the second entity bean. The plus sign is followed by text that describes what role Entity bean 2 plays the association.



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:

- Review the information in Chapter 37, "Overview of loading store data," on page 335.
- 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. For more information on the order of loading assets, see Chapter 38, "Loading WebSphere Commerce database asset groups," on page 383.

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 `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 the Administration Console, 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=FashionFlow"
    description="FashionFlow Catalog"
    tpclevel="0"/>
    ```
  - Post ID Resolver

    ```
    <catalog
    catalog_id="10001"
    member_id="-2000"
    identifer=FashionFlow"
    description="FashionFlow Catalog"
    tpclevel="0"/>
    ```

where 10001 is the unique ID assigned by the ID Resolver and -2000 is the member ID selected by the user in the Administration Console. 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.

Appendix C. 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 390, 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	**`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 396for 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

Notices

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 USA