

Course Guide

IBM WebSphere Commerce V8 Programming Essentials

Course code 6G80 ERC 2.0



March 2017 edition

Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

IBM Director of Licensing

IBM Corporation

North Castle Drive, MD-NC119

Armonk, NY 10504-1785

United States of America

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

© Copyright International Business Machines Corporation 2016.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Unit 1.....	4
Unit 2.....	77
Unit 3.....	141
Unit 4.....	187
Unit 5.....	314
Unit 6.....	383
Unit 7.....	425
Unit 8.....	479
Unit 9.....	538

Unit 1

WebSphere Commerce V8 Architecture

© Copyright IBM Corporation 2016

This section describes an overview of the WebSphere Commerce V8 Architecture.

Unit objectives

This unit is designed to enable you to:

- Start a development project that implements WebSphere Commerce V8.
- Describe new features and functions in WebSphere Commerce V8.
- Describe the runtime architecture of WebSphere Commerce.
- Understand the Application Layers
- Understand WebSphere Commerce functional overview
- Understand WebSphere Commerce Design Patterns
- Describe the development model for WebSphere Commerce.
- Summarize the Business Logic and Persistence Layers.
- Troubleshoot and debug WebSphere Commerce applications.

© Copyright IBM Corporation 2016

About WebSphere Commerce

- WebSphere Commerce is a single, unified platform that offers the ability to do business directly with consumers (B2C), directly with businesses (B2B), and indirectly through channel partners (indirect business models).
- Highly customizable, scalable, and high availability solution that is built to leverage open standards.
- Started in 1996 as Net.Commerce (V1.0, V2.0, V3.1, and V3.2) and later renamed to WebSphere Commerce Suite, releasing versions V4.1 and V5.1
- Next major releases were in 2006 Version 6.0, in 2009 Version 7.0 and in 2015 Version 8.0
- Business Models :
 - Extended sites
 - B2C (only in extended sites, standalone not supported)
 - B2B (only in extended sites, standalone not supported)
- WCS 8 Product portfolio :
 - WebSphere Commerce Professional
 - WebSphere Commerce Enterprise
- Fix packs & mod packs

© Copyright IBM Corporation 2016

B2C - A store that supports commerce transactions involving products, services, or information directly between businesses and consumers.

B2B - A store that supports commerce transactions involving products, services, between two businesses or parties.

Extended Site - A site that allow your selling organization to provide unique storefronts for different enterprise customers or showcase a number of branded storefronts

WebSphere Commerce Professional - Provides a comprehensive business-to-consumer (B2C) cross-selling solution that midsized companies can use to deliver a personalized and consistent shopping experience

WebSphere Commerce Enterprise - Is an omni-channel, e-commerce platform that enables business-to-consumer (B2C) and business-to-business (B2B) sales to customers across all channels, which include web, mobile, social, store, and call center.

Key benefits of WebSphere Commerce

- Supports all Ecommerce business models
 - Business-to-consumer (B2C)
 - Business-to-business (B2B)
 - Supply and demand chains
- Provides powerful business user tools for
 - Merchandising and search term associations
 - Catalog Management
 - Cross-channel precision marketing campaigns and promotions
 - Site and store administration
- Drives improved customer loyalty and increased shopping cart sizes (customer centricity)
- Delivers a seamless, branded shopping experience across all channels (cross channel focus)
- Uses the power of the underlying IBM platform for optimal performance, scalability, reliability, and high availability

© Copyright IBM Corporation 2016

WebSphere Commerce offers many benefits to build strong and effective web-enabled stores. Some of these benefits include:

- Support for all company selling business models, including business-to- consumer, business-to-business, supply and demand chains, as well as other models, such as extended sites
- Powerful business user tools for Merchandising and search term associations, Catalog Management, cross-channel Precision Marketing campaigns, promotions, and site and store administration
- Customer centricity - Drives improved customer loyalty and increased shopping cart sizes by delivering rich, personalized, and contextually relevant content at each stage of the shopping experience
- Cross channel focus - Delivers a seamless, branded shopping experience across all channels (web, mobile, kiosk)
- Leverages the power of the underlying IBM platform (IBM WebSphere Application Server and Rational Application Developer) for optimal performance, scalability, reliability, and high availability

WebSphere Commerce features: Cross-channel

- Seamless customer experience across multiple channels:

- Web (WebSphere Commerce)
- Mobile (WebSphere Commerce Mobile Store)
- In Store POS and Kiosks (Retail Store Solutions)
- Call Center (Sterling Order Management)
- Social Media/Social Commerce



© Copyright IBM Corporation 2016

A channel describes one method in which a consumer may navigate or purchase from your site. For instance, a typical e-commerce channel is the website, where consumers browse through the products of a store and make purchases. Consumers may, however, make purchases through a contact center, in-store devices, such as kiosks, or mobile devices, which are now popular.

Today, consumers continuously traverse across channels to research, review, compare, and purchase products, and expect the brand experience to be consistent and seamless regardless of channel, touchpoint, or order of interaction. Inconsistent cross-channel information can be destructive at the point of purchase. WebSphere Commerce provides a set of ready-to-use Web shopping flows across a wide variety of channels that display views of local in-store inventory, lets a customer place orders online that can be picked up in-store, and update a customer with status updates throughout the completion of the order lifecycle.

With the increased use of mobile devices and technology, WebSphere Commerce Version 8.0 enables smart phones as new customer touch points. For example, browsing the online store, conducting side-by-side product comparisons, receiving marketing messages (Short message service – SMS text messages), promotion codes, digital coupons, store location information, inventory availability, and completing transactions. Mobile devices also support a buy-online, pickup-in-store model. The mobile store model provides ready-to-use support for smartphones allowing companies to quickly adopt mobile devices as a new channel for doing business.

You can also interact with a human by calling the Retailer after hours call-center to schedule a service or report a problem. It is powered by Sterling Commerce Order Management call center application. On social networking sites such as Facebook or Twitter you can post positive reviews of discuss your shopping experience. Coremetrics Social Analytics can analyze data from such social networking sites to determine if positive or negative feedback is being discussed about a campaign.

Getting started with WebSphere Commerce V8

After completing this topic, you should be able to describe the following areas of WebSphere Commerce V8:

- New features and functions in WebSphere Commerce V8
- Programming specifications of WebSphere Commerce V8
- Establishing a team and developing a process for implementing a solution by using WebSphere Commerce V8

© Copyright IBM Corporation 2016

What is new in WebSphere Commerce V8

- Supports IBM Commerce Insights
- Cloud-based offering
- New DHTML-based Management Center with modernized user interface
- Customer service features added to Aurora starter store
- Better customer service and support experience
- Migration assistance utilities
- Developer infrastructure improvements

© Copyright IBM Corporation 2016

Commerce Insights : Commerce Insights is a tool for merchandisers to get a better view of product and category performance Graphical focus, see data overlaid on product images.

- 1) Browse store preview with performance data overlaid
- 2) Click-to-edit to launch CMC with SSO
- 3) Optional link to Watson Analytics

IBM Commerce Insights for Watson Analytics gives the merchandiser access to analyze and discover trends in the sales data through Watson Analytics.

Cloud-based offering : Cloud based offering requires WebSphere Commerce Version 8.
Both on premise and on-cloud WebSphere Commerce V8 supported.

IBM Management Center enhancements: The IBM Management Center for WebSphere Commerce, otherwise known as Management Center, is the next generation business user tool for managing business tasks for online businesses. Using IBM Management Center, business users can perform tasks to create, update, and maintain merchandising and marketing assets in stores that use the consumer direct, B2B direct, demand chain, supply chain, and extended sites business models.

Management Center is now based on Dynamic HTML (DHTML) and the open source Spring Framework. There is no longer a dependency on Adobe Flash technology. Developers who customize the Management Center can now view their changes without first compiling their code. Management Center user interface now has a new look

Each business function can be managed by using tools within IBM Management Center. New features and functionality are added to each of the business user tools in IBM Management Center to provide usable, intuitive, and efficient ways to complete your business tasks.

Here are a few examples:

- Marketing tool: The Marketing tool in WebSphere Commerce Version 8 includes enhancements to features such as the Activity Builder, custom activity templates, dialog activities, and e-Marketing Spots.
- Promotions tool: The Promotions tool in WebSphere Commerce Version 8 includes new promotion types and enhancements to help you manage your promotions.
- Catalogs tool: The Catalogs tool in WebSphere Commerce Version 8 includes enhancements to the creation and management of sales catalogs and supports additional search criteria in both the simple and advanced search.

Customer Service in Aurora: WebSphere Commerce Version 8 includes the initial use cases for Customer Service Representative (CSR) support in the Aurora starter store, including adding shoppers, buying on behalf of shoppers, managing orders (cancelling, reordering), managing shopper account information, etc.

Although included in the standard starter store pages in the V8 Aurora, the Customer Service functionality (JSPs and APIs) will require a separate license to use.

Migration utilities: Migrations from any level of WebSphere commerce Version 7 to Version 8 is supported. If you are on an older version, you should migrate through Version 7. For example, V6.0 → V7.0 FEP8 → V8.0.

Discontinued the instance migration tool, Instead, create an empty Version 8 instance, restore Version 7 database, deploy custom code, and run database migration tool.

Developer infrastructure improvements: The WebSphere Commerce development environment contains several enhancements from V7 to V8. Development environment consists of:

- Rational Application Developer or Rational Software Architect Version 9.5 or higher
- A 64-bit WebSphere Application Server. You can install
 - either one of the following WebSphere Application Server Test Environment V8.5.5.7 or
 - higher WebSphere Application Server Full
 - Version V8.5.5.7 or higher

If you are installing the WebSphere Application Server full version (WebSphere Application Server Version 8.5.5), then you need to update to WebSphere Application Server Fix Pack 7.

WebSphere Commerce Development database is used to store development artifacts. Database can be IBM DB2, Apache Derby which is installed by default or Oracle.

WebSphere Commerce V8 requires 64-bit machine which is supported on all Windows operating systems. It requires minimum disk space of 50 GB and memory of 4 GB

Refer to the Knowledge Center for the current suggested versions for all required software to get more information.

Deprecated and Discontinued features (Summary)

- Deprecated
 - BOD based search runtime programming model
 - WebSphere Commerce search profiles
 - 3-DES encryption
 - Massload
 - Support for the Configuration noun
 - PriceTCMasterCatalogWithOptionalAdjustment
- Discontinued
 - WebSphere Commerce – Express Edition
 - WebSphere Commerce Patterns
 - Madisons, Elite, Brazil, MayUJoy(China) stores
 - Coshopping
 - sMash-based Social Commerce
 - BazaarVoicelegacy integration (“PRR”)
 - KickAppsintegration
 - iOS hybrid and Android native sample application

© Copyright IBM Corporation 2016

This is a summary page of all the deprecated and discontinued features. More details at the end of the deck.

Deprecated :

•WebSphere Commerce search runtime programming model

- The BOD-based WebSphere Commerce search runtime programming model is deprecated in WebSphere Commerce Version 8. This includes search configurations on the WC EAR, and catalog storefront services.
- Use REST based search runtime programming model.

• WebSphere Commerce search profiles

- Extending search profiles is deprecated in WebSphere Commerce Version 8, due to the complexity of search profiles inheritance logic.
- Include all your required configurations and properties when creating new custom search profiles. Or, redefine default search profiles by overriding any properties that you want to change.

•3-DES encryption

- We recommend using the stronger AES encryption
- See also the security presentation from the V7 FEP7 release enablement (link below)

•**Massload** - Has been deprecated since V7 FEP6, continues to be supported, but we recommend you switch to using DataLoad

•**Support for the Configuration noun** - Use the OnlineStore noun instead

•**PriceTCMasterCatalogWithOptionalAdjustment** - Use PriceTCMasterCatalogWithFiltering instead

Discontinued :

- WebSphere Commerce – Express Edition
- WebSphere Commerce Patterns
- The Madisons, Elite, Brazil or MayUJoy(China) stores
 - You cannot publish these SARs on V8, but you can migrate an already published store from V7 to V8
- Coshopping
 - Use 3rd-party providers, e.g. *Shop with Friends* by OpenClove
- sMash-based Social Commerce
 - Use relevant IBM Commerce Partner Solution
- BazaarVoice legacy integration (“PRR”)
 - Use new BazaarVoice Conversations integration through Composer widgets
- KickApps integration
 - Implement REST-based integration, see the Pinterest Buy It button support for an example
- iOS hybrid and Android native sample application
 - Use the RWD features of Aurora

What is new in Version 8 Mod Pack 1

- Marketing Cloud Integration - Silverpop Transaction
- Customer Service Representation (CSR) comments on orders
- Enhancements to enable new features in IBM Commerce Insights
 - Changing the display sequence using drag and drop
- Management Center Migration Utility
- Staging Environment Migration
- WebSphere Commerce on IBM i

© Copyright IBM Corporation 2016

Marketing Cloud Integration : Integration of Silverpop Transaction with WebSphere Commerce is possible so that transaction emails can be sent by Silverpop Transact. Silverpop Transact provides the following benefits:

Send custom-branded transactional messages in rich HTML.

Cross-sell and up-sell by dynamically recommending products or services in transactional messages.

Review reports that use advanced tracking to gain insight into the deliverability and performance of transactional emails.

Integrate with Point of Sale and receipt systems to monitor offline and online behaviors.

Trigger billing notifications and password resets for real-time interactions.

Save copies of transactional messages for future use. Access Engage Transact by using XML APIs or SMTP relays.

CSR order comments : A customer service representative (CSR) can add or view customer comments for specific orders on either the Aurora B2C or the Aurora B2B storefront.

Changing the display sequence by dragging products :

In Commerce Insights Store view, merchandisers and administrators can drag products to position them within a category. No form editor or special tool is required to set a top product in a category or move a slow-selling item to a lower position.

Management Center Migration Utility:

A command-line based tool that helps migrate OpenLaszlo-based customizations from WC 7 FEP 2 and onwards to V8 based Management Center code. Custom OpenLaszlo source code and definitions files -> Version 8 CMC definitions

Struts configurations are ported to Spring configurations Images, resource bundles, configuration files are ported

migrateLOBTools.bat version7LOBToolsdirectory featurePackNumber [Edition]

Eg. migrateLOBTools c:\IBM\WCDE_ENT70\workspace\LOBTools 8 pro

Staging environment migration:

V7 staging environments can be migrated to V8 with minimal downtime (Includes live production)

A documented sequence of migrations steps that can be configured for staging or authoring instances

Refer to the knowledge Center topic: Migrating runtime environments (with staging servers)

http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.migrate.doc/tasks/tmg_server_stage.htm?lang=en

WebSphere Commerce on IBM i:

Install WebSphere Commerce on systems that are running IBM i Version 7.1 or later.

WebSphere Commerce for IBM i is packaged with Mod Pack 1

Programming specifications

WebSphere Commerce Version 8.0 uses a number of programming specifications:

- Dojo
 - Stores: 1.8.7
 - LOBTools (Management Center): 1.10.4
- EJB 2.1
- EMF 2.2
- Java Servlet 2.5
- ICU4J 55
- JavaMail 1.5
- JAX-RPC 1.1
- JEE 5
- JDK/JRE Server and Client Tools 1.6
- JMS 1.1
- JSP support (from WebSphere Application Server) 2.1
- JTOpen 8.5
- SDO 2.4 in compatibility mode
- Apache Solr 4.10.3
- Spring 4.1.7
- Struts 1.2.9
- Swagger 1.2
- Wink 1.4
- XML
- XSD 1.1
- XSLT

© Copyright IBM Corporation 2016

WebSphere Commerce is a customizable, scalable, enterprise-class Java EE application that is built upon a number of open standards and APIs.

Development process: The development cycle

Stage 1: Model

Gather requirements, design, simulate, and optimize business models
Team: Architects, Business Analysts

Stage 2: Assemble

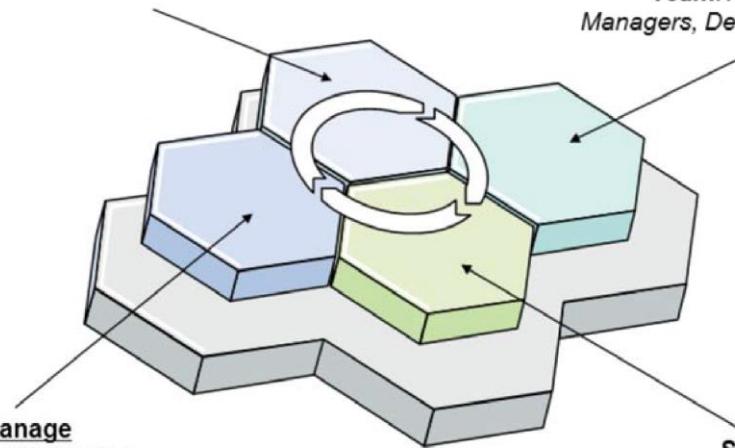
Discover, assemble, test
Team: Architects, Project Managers, Development teams

Stage 4: Manage

Business monitoring for coordinated interaction, analytics, and optimization
Team: Architects, Administrators

Stage 3: Deploy

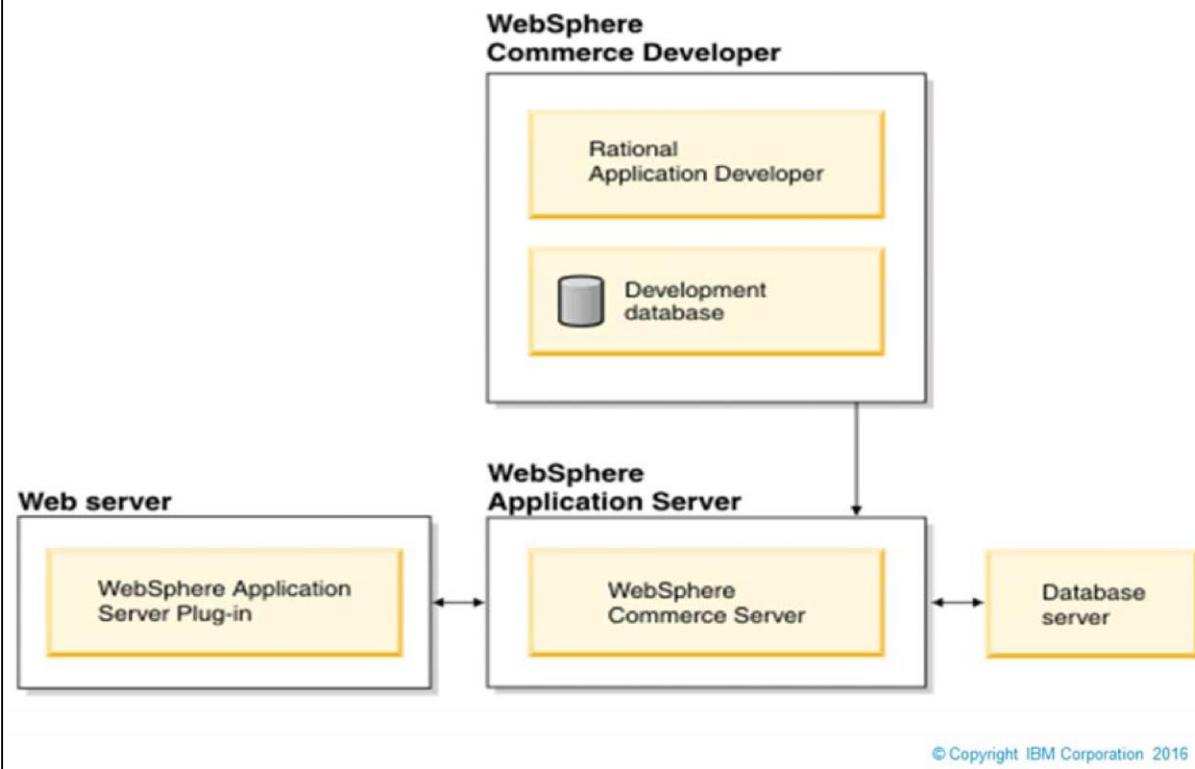
Integrated deployment of processes
Team: Project Managers, Administrators



© Copyright IBM Corporation 2016

Development follows a cyclic approach. It begins with a planning stage, which leads to assembling the solution, deploying it, and then managing it for future modeling and planning.

WebSphere Commerce common architecture



This diagram shows a simplified view of the common architecture:

The Web server is the first point of contact for incoming HTTP requests for your e-commerce application. In order to interface efficiently with the WebSphere Application Server, it uses the WebSphere Application Server plug-in to manage the connections between the two components.

The WebSphere Commerce Server runs within the WebSphere Application Server, allowing it to take advantage of many of the features of the application server. The database server holds most of your application's data, including product and customer data. In general, extensions to your application are made by modifying or extending the code for the WebSphere Commerce Server. In addition, you may have a need to store data that falls outside of the realm of the WebSphere Commerce database schema within your database.

Developers use Rational Application Developer to perform the following tasks:

- Create and customize storefront assets such as JSP and HTML pages
- Create and modify business logic in Java
- Create and modify access beans and EJB entity beans
- Test code and storefront assets
- Create and modify Web services

The WebSphere Commerce development environment uses a development database. Developers can use their preferred database tools (including Rational Application Developer) to make database

modifications. WebSphere Commerce supports a one to one mapping between the WebSphere Commerce instance and the WebSphere Commerce database. Running multiple WebSphere Commerce instances against the same database is not supported.

WebSphere Commerce runtime framework enhancements

- The runtime framework enable support for multiple sales channels
- A sales channel is a method that customer can use to purchase merchandise
 - For example,
 - in-store, from an online store, or
 - from a call center.
 - Requests can enter the WebSphere Commerce Server from different types of clients such as a rich client, kiosk or as a browser request.
- Struts
 - Uses the struts open source implementation.
 - Struts enforce
 - Best practices and design patterns,
 - Boasts a large developer community, and
 - Supported by IBM development tools such as Rational Application Developer, and by third-party tools.
- Web services
 - A series of open protocols and standards that are used in communication between two systems

© Copyright IBM Corporation 2016

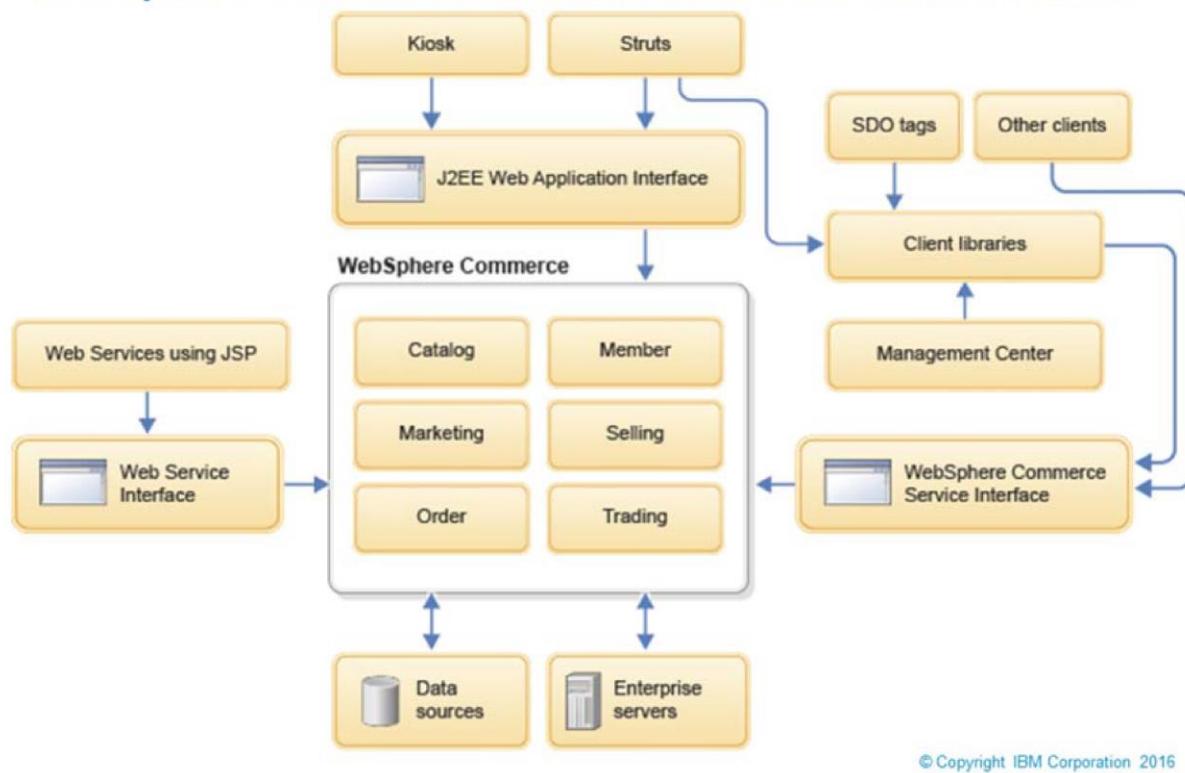
The runtime framework enhancements better enable support for multiple sales channels by further decoupling the presentation tier from the business logic tier. A sales channel is a method that a customer can use to purchase merchandise for example, in-store, from an online store, or from a call center. Requests can enter the WebSphere Commerce Server from different types of clients such as a rich client, kiosk or as a browser request.

WebSphere Commerce runtime framework enhancements

- Web services
 - Perform functions that range from simple requests to complicated business processes.
 - WebSphere Commerce web services framework has the following capabilities:
 - Uses the WebSphere web service runtime infrastructure and Rational tools
 - Promotes the use of industry-standard service definitions
 - Allows top-down and bottom-up creation of web services
 - Uses the JSP page composition service to generate the web service response, allowing dynamic caching to be optimized
 - Uses the existing command pattern to represent the business logic
- Business context service
 - Tracks user session information from different channels by using different types of business contexts
 - Provides a pluggable interface where custom context information can be defined
 - Extend WebSphere Commerce out session attributes with custom information

© Copyright IBM Corporation 2016

WebSphere Commerce runtime framework enhancements



© Copyright IBM Corporation 2016

Reference to WebSphere Commerce runtime framework:

http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/csdwebsphere_commerce_runtime_framework.htm?lang=en

WebSphere Commerce application layers (1 of 4)

- Business models
 - Represents a sample business situation in which the WebSphere Commerce product may be used
 - The business models provided
 - B2B
 - B2C
 - Within each business model, starter stores are provided
- Business processes
 - Processes available are divided by business model
 - Administrative processes
 - Used to administer a site, a store, or an organization
 - Starter stores
 - Contains sample processes, use them as a guideline, or a starting point, for site development.
 - Solution
 - Describes the high-level view for administrative processes and starter store processes
 - Combines processes to explain the relationship between the various process groups



© Copyright IBM Corporation 2016

Business models

A business model represents a sample business situation in which the WebSphere Commerce product may be used. A business model describes a scenario in which various parties use WebSphere Commerce to achieve their needs. Some business models that are provided with WebSphere Commerce include:

- B2B

- B2C

Within each business model, WebSphere Commerce provides starter stores, which may be used as a starting point to develop online sites.

Business processes

Represents the processes available in WebSphere Commerce divided by business model. The business processes are divided into three areas:

Administrative processes: Processes that are used to administer a site, a store, or an organization. These processes are generally used as-is, that is, a change to, or an addition of, an additional administrative process, usually means customizing WebSphere Commerce.

Starter stores: Starter stores contain sample processes that the customers of the stores follow. Many different kinds of stores, satisfying a wide range of business needs, can be created with WebSphere Commerce. Use the processes that are described in starter stores as a guideline, or a starting point, for site development. Changing or adding a process to the starter store processes requires changes to the site design. Typically, this type of change does not require customizing the underlying infrastructure.

Solution: A solution describes the high-level view of how all the administrative processes and starter store processes fit in the overall business model framework. A solution combines processes into a coherent picture, which explains the relationship between the various process groups.

WebSphere Commerce application layers (2 of 4)

- Presentation layer
 - Responsible for displaying results
 - Multiple presentations layers supported
 - You can use an appropriate presentation layer based on your business requirements.
 - Web : Display is rendered using JSP files
 - MVC design pattern is used to implement view layer
- Service layer
 - Implemented using OAGIS messages (Open Applications Group Integration Specification)
 - Channel-independent mechanism to access business logic
 - Segregates the implementation of business logic such as order and catalog
 - Supports two transport mechanisms
 - local Java binding
 - Web services

© Copyright IBM Corporation 2016

Presentation layer

The presentation layer is responsible for displaying results. Multiple presentation layers can be supported. By default, the Web presentation layer is demonstrated in the starter stores. For the Web presentation layer the display is rendered using JSP files. WebSphere Commerce uses Java Server Pages (JSP) to implement the view layer of the Model-View-Controller (MVC) design pattern. The view layer is in charge of retrieving data from the database through the use of REST services and formatting it to meet the display requirements.

Service layer

OAGIS stands for Open Applications Group Integration Specification.

WebSphere Commerce service interfaces are defined using the OAGIS message structure, from The Open Applications Group. The OAGIS standard provides a consistent message structure and model for messaging using XML. The OAGIS standards describe Business Object Documents (BODs) as nouns and the services that interact with those nouns are called verbs. For more details on **WebSphere Commerce use of Open Applications Group (OAGIS) messaging, see**

https://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.webservices.doc/concepts/cwvoagis.htm

The service layer, which is implemented by using OAGI messages, is a channel-independent mechanism that can access WebSphere Commerce business logic. The service layer segregates the implementation of business logic such as order and catalog. This segregation permits the underlying implementation to change without requiring that the caller change. All clients, including web clients and back-end services, go through the service layer to run business logic. The service layer supports two transport mechanisms: local Java binding and web services.

WebSphere Commerce application layers (3 of 4)

- Business logic
 - Business rules are implemented independent of the presentation layer
 - Is implemented using the command pattern
 - Two types of commands are implemented
 - NVP Controller commands
 - Deal with Name Value Pairs
 - Accessible by the presentation layer
 - Used as a coordinator of tasks
 - Internally use task commands to break down business logic into discrete independent tasks
 - BOD commands
 - Process OAGIS messages
 - Deal with service data (instead of NVPs)

© Copyright IBM Corporation 2016

Business logic

The business logic layer is the business components that provide OAGIS services to return data or start business processes. The presentation layer uses these OAGIS services to display data, or to invoke a business process. The business logic provides data required by the presentation layer. The business logic layer exists because more than just fetching and updating data is required by an application; there is also additional business logic independent of the presentation layer.

•The business logic layer is where business rules are implemented independent of the presentation layer. Business logic is implemented by using the command pattern. Two types of commands that are implemented include:

- Controller commands - accessible by the presentation layer and used as a coordinator of tasks.
- Task commands - not accessible by the presentation layer but called from the controller commands. This command type is used to implement business rules.

Commands can obtain information using the command context. Examples of information available include the user's ID, the user object, the language identifier, and the store identifier.

WebSphere Commerce application layers (4 of 4)

- Persistence layer
 - Records the data and operations of the WebSphere Commerce system
 - Represents entities within the commerce domain
 - Encapsulate the data-centric logic required to extract or interpret information contained within the database
 - EJBs
 - Data Service Layer (DSL)
- Database schema
 - Includes over 600 tables, is designed specifically for e-commerce applications and their data requirements.
 - Supports persistence requirements for the subsystems like:
 - Order
 - Catalog
 - Member
 - Marketing
 - Trading
 - Supports Derby (only in development environment), DB2 and Oracle databases.

© Copyright IBM Corporation 2016

Persistence layer

The persistence layer records the data and operations of the WebSphere Commerce system. The persistence layer represents entities within the Commerce domain and encapsulates the data-centric logic that is required to extract or interpret information that is contained within the database. These entities comply with the Enterprise JavaBeans specification. These entity beans act as an interface between the business components and the database. In addition, the entity beans are easier to comprehend than complex relationships between columns in database tables.

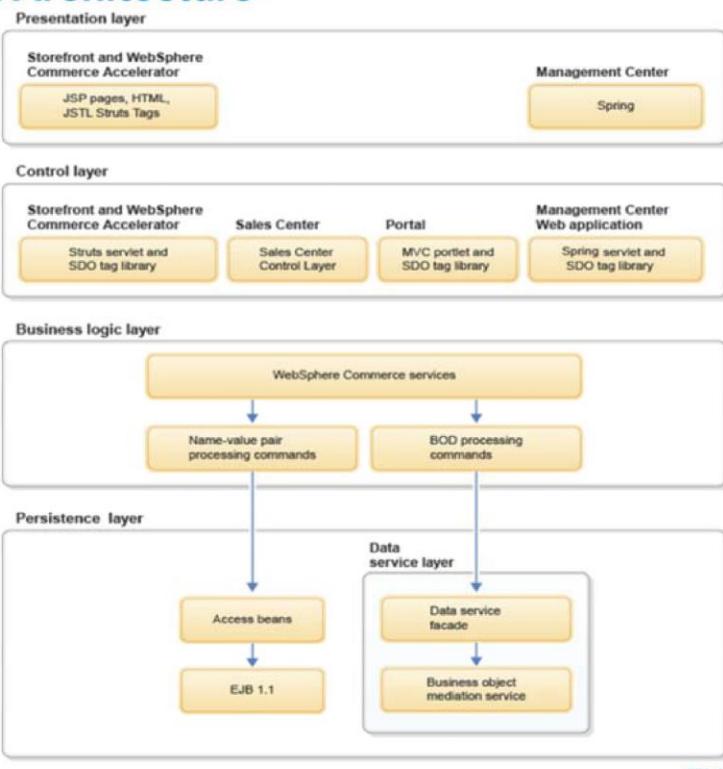
The data service layer (DSL) provides an abstraction layer for data access that is independent of the physical schema.

The interaction between the business objects and persistence layer is isolated in an object called the Business Object Mediator. Business object document (BOD) commands interact with the Business Object Mediator to handle the interaction with the logical objects and how they are persisted.

Database schema

WebSphere Commerce database schema, which includes over 600 tables, is designed specifically for e-commerce applications and their data requirements. The database schema supports persistence requirements for the WebSphere Commerce subsystems (Order, Catalog, Member, Marketing, Trading). WebSphere Commerce supports both DB2 and Oracle relational databases, also supports Derby database in development environment.

Functional Architecture



Copyright IBM Corporation 2016

WebSphere Commerce is multichannel-enabled, meaning that WebSphere Commerce can support transactions across various sales channels. The framework enhancements in this release support multiple presentation layers, responsible for displaying results, which decouple control logic from business logic.

WebSphere Commerce supports two channels: the Web channel and the sales channel. For the Web channel the presentation is rendered using JSP pages and the Web controller layer uses Struts. For the sales channel, the display uses the Eclipse rich client technology.

The functional architecture is split into four layers:

Presentation layer

Supports all channels/touchpoints using a variety of technologies.

Controller layer

It controls the transaction scope and manages the session related information for the request. Depending on channel, uses Struts, SDO tag library, MVC portlet or REST JAX-RS

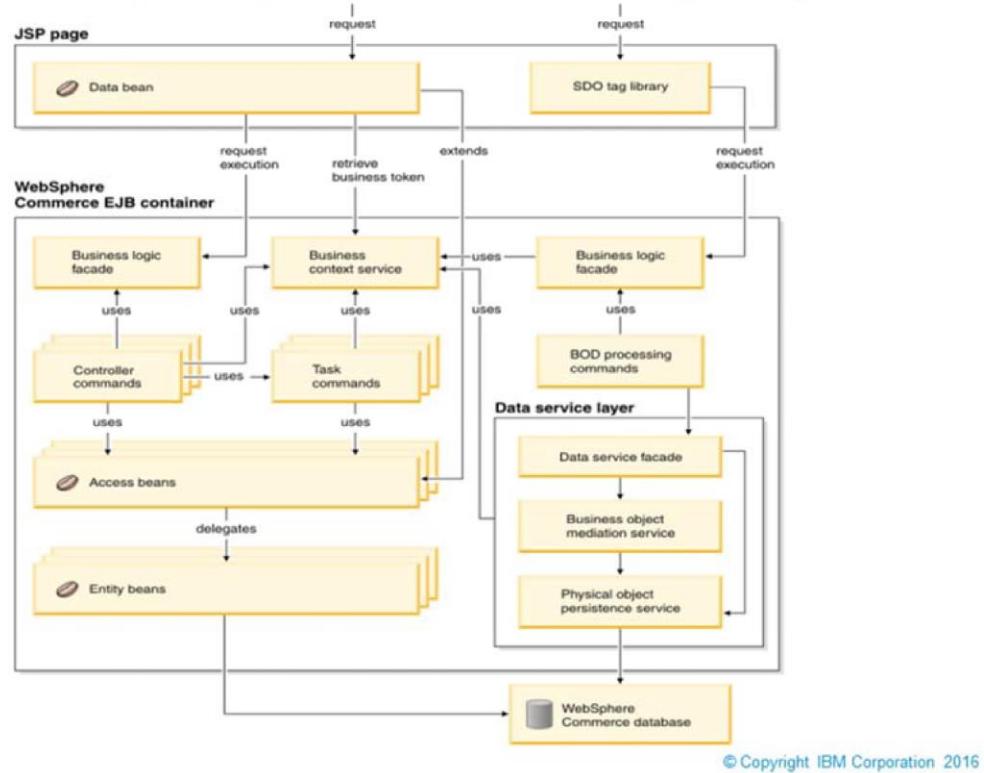
Business Logic layer:

Implements the WebSphere Commerce services architecture using a combination of NVP controller commands and OAGIS BOD commands

Persistence layer:

Handles all persistence using either EJBs or Data Services Layer (DSL). The data service layer (DSL) provides an abstraction layer for data access that is independent of the physical schema.

WebSphere Commerce functional overview (1 of 4)



© Copyright IBM Corporation 2016

Based on a loose coupling of the presentation and business logic layers WebSphere Commerce functional architecture is shown as above.

Refer more on the knowledge center:

http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/csdfunover.htm?lang=en

Although there have been enhancements in the WebSphere Commerce runtime, task commands, controller commands, access beans, and entity beans should continue to function as in previous releases.

Controller layer: The conductor of operations for a request. It controls the transaction scope and manages the session related information for the request. The controller first dispatches to a command and then calls the appropriate view processing logic to render the response.

Presentation layer: The presentation layer displays the result of command execution. The presentation layer can use JSP pages, or other rendering technologies.

Business Context Service (BCS) A service that manages contextual information used by business components. The contexts include such information as globalization and entitlement.

Business logic facade: This generic interface is implemented as a stateless session bean which the controller calls to invoke controller commands. Controller commands A controller command business process logic such as OrderProcess. It invokes task commands to accomplish different unit of work in the business process. By default, access control is enabled for controller commands.

Task commands: A task command is an autonomous task that accomplishes a specific unit of application logic such as check inventory. A task command usually works with other task commands to complete processing of a controller command. By default, access control is not enabled for task

commands.

Access beans: Access beans are simple persistent objects with setters and getters. The access bean behaves like a Java bean and hides all the enterprise bean specific programming interfaces, like JNDI, home and remote interfaces from the clients. Rational Application Developer provides tooling support to generate access beans from the schema.

Entity beans: Entity beans are used in the persistence layer within WebSphere Commerce. The architecture is implemented according to the EJB component architecture. The EJB architecture defines two types of enterprise beans: entity beans and session beans.

Data service layer: The purpose of the data service layer is to provide a consistent interface called the data service façade for accessing data, independent of the object-relational mapping framework. Java objects are implemented as physical service data objects (SDOs). The data service layer performs bidirectional transformations between physical SDOs and logical SDOs. It allows you to perform create, retrieve, update, and delete operations on the logical SDOs.

WebSphere Commerce functional overview (2 of 4)

- Controller layer
 - The conductor of operations for a request.
 - It controls the transaction scope and manages the session related information for the request.
 - The controller first dispatches to a command and then calls the appropriate view processing logic to render the response
- Presentation layer
 - Displays the result of command execution.
 - Layer can use JSP pages, or other rendering technologies.
- Business Context Service (BCS)
 - service that manages contextual information used by business components
 - The contexts include such information as globalization and entitlement
- Business logic façade
 - Generic interface is implemented as a stateless session bean which the controller calls to invoke controller commands

© Copyright IBM Corporation 2016

WebSphere Commerce functional overview (3 of 4)

- Controller commands
 - Process logic such as OrderProcess
 - Invokes task commands to accomplish different unit of work in the business process.
 - By default, access control is enabled for controller commands.
- Task commands
 - An autonomous task that accomplishes a specific unit of application logic such as check inventory
 - Usually works with other task commands to complete processing of a controller command
 - By default, access control is not enabled for task commands
- Access beans
 - Simple persistent objects with setters and getters
 - The access bean behaves like a Java bean and hides all the enterprise bean-specific programming interfaces, like JNDI, home and remote interfaces from the clients
 - Rational Application Developer provides tools support to generate access beans from the schema

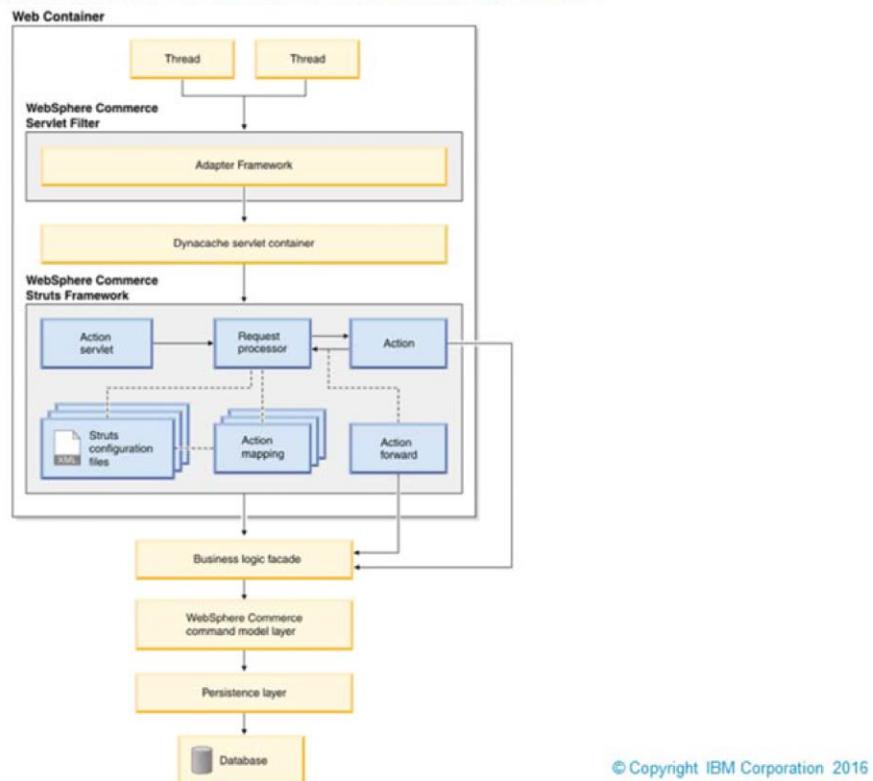
© Copyright IBM Corporation 2016

WebSphere Commerce functional overview (4 of 4)

- Entity beans
 - Used in the persistence layer within WebSphere Commerce.
 - Architecture is implemented according to the EJB component architecture.
 - The EJB architecture defines two types of enterprise beans:
 - Entity beans
 - Session beans

© Copyright IBM Corporation 2016

WebSphere Commerce framework interaction



Above diagram provides a summary of the interaction flow between components when forming a response to a request.

- 1) The request is directed to the Presentation layer in its own thread.
- 2) The thread handling the request is dispatched to the WebSphere Commerce servlet filter. The filter passes the request to the adapter framework.
- 3) The adapter manager determines which adapter is capable of handling the request, then returns that adapter to associate with the request.

For example, if the request originated from an Internet browser, the adapter manager associates the request with the HTTP browser adapter. The adapter is passed back to the DynaCache servlet container. The filter regains control and passes the request to the servlet engine for processing. At this point, either of the following actions can occur:

- a. The request can be cached and the cached response can be returned.
 - b. If the request is not cached, the Struts action invokes the business logic facade by specifying the interface name of the business logic to invoke and the associated parameters. The business logic facade queries the command registry to determine the appropriate implementation for the store that is associated with the request.
- 4) The business logic facade invokes the appropriate controller command. 5) The controller command begins execution:
 - a. The controller command can access the database using an access bean and its corresponding entity bean.
 - b. The controller command can invoke one or more task commands. Then, task commands can access the database, using access beans and their corresponding entity beans.
 - c. A combination of a and b.
 - 6) The business logic facade returns a set of properties to the Struts action. One of the elements that

is part of the properties is the key to the global forward that represents the response.

7) The action looks up the global forward in the Struts configuration files. It resolves to the right one based on the store configuration.

The action forward implementation that is selected is the appropriate one for the device of the request.

8) The Struts request processor executes the action forward which will execute the appropriate JSP page. Within the JSP page, a data bean is required to retrieve dynamic information from the database. The data bean manager is used to activate the data bean.

9) The access bean from which the data bean is extended accesses the database using its corresponding entity bean. Based on globalization information in the request, the data bean formats the data.

Design patterns

- After completing this topic, you should be able to know about following design patterns.
- Any solution extending from WebSphere Commerce should adhere to these high-level design patterns.
 - Model-View-Controller design pattern
 - Command design pattern
 - Display design pattern

© Copyright IBM Corporation 2016

Model-View-Controller design pattern (1 of 2)

- Consist of a data model, presentation information, and control information.
- Each is separated as different object
 - Model
 - Contains only the pure application data
 - View
 - Presents the model's data to the user
 - Knows how to access the model's data
 - Controller
 - Exists between the view and the model
 - Listens to events triggered by the view and executes the appropriate reaction to these events.

© Copyright IBM Corporation 2016

The model-view-controller (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

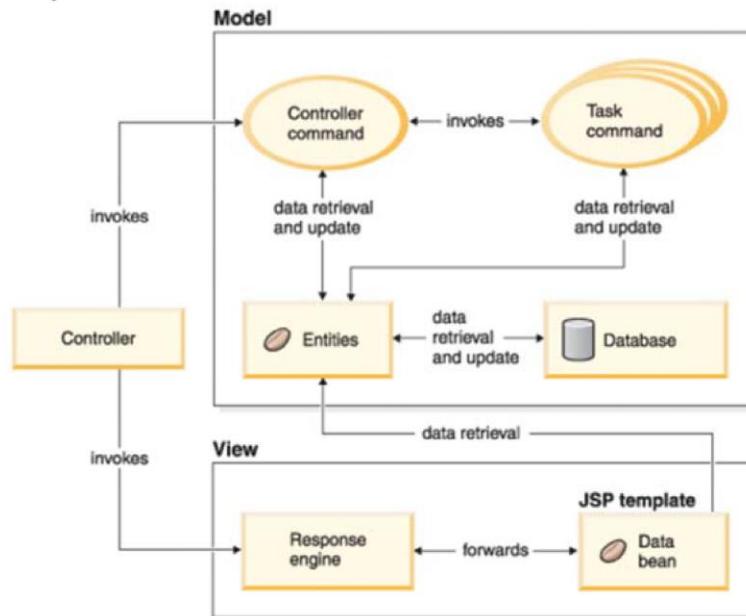
The *model* (for example, the data information) contains only the pure application data. It contains no logic describing how to present the data to a user.

The *view* (for example, the presentation information) presents the model's data to the user and knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.

The *controller* (for example, the control information) exists between the view and the model. Listens to events triggered by the view and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.

Model-View-Controller design pattern (2 of 2)

- Following diagram shows how the MVC design pattern applies to WebSphere Commerce



© Copyright IBM Corporation 2016

The above diagram shows how the MVC design pattern applies to WebSphere Commerce.

This pattern is used for both the Web

applications and for the rich clients and can use either Struts or the Web services framework.

Command design pattern

- Commands implement the business rules of your site.
 - There are two types of commands:
 - Controller commands
 - Implement business tasks such as user registration and allocation of inventory.
 - Task commands
 - Implement discrete pieces of a business task such as address verification or ensuring passwords comply to the defined password policies.
- Command beans follow a Command design pattern.
- Every command includes both an interface class and an implementation class
- The key mechanisms that are enabled within this level of indirection include:
 - The ability to invoke an access control policy manager that determines if the user is allowed to invoke the command.
 - The ability to execute a different command implementation for different stores, based on the store identifier.

© Copyright IBM Corporation 2016

WebSphere Commerce Server accepts requests from browser-based thin-client applications; and remote applications. For example, a request may come from a remote procurement system, or from another commerce server.

All requests, in their variety of formats, are translated into a common format by the controller layer. Once the requests are in this common format, they can be understood by WebSphere Commerce commands.

Every command includes both an interface class for example, CategoryDisplayCmd and an implementation class for example, CategoryDisplayCmdImpl. From a caller's perspective, the invocation logic involves setting input properties, invoking an execute() method, and retrieving output properties.

In order to create new command objects, the caller of the command uses the *command factory*. The command factory is a bean that is used to instantiate commands.

Display design pattern

- Display pages return a response to a client.
- Display pages are implemented as JSP pages.
- To support multiple device types, a URL access to a page should use a view name, not the name of the actual JSP file.
 - Due to JSP page representing a particular view:
 - The ability to select the appropriate view is highly desirable
 - For example: two shoppers requesting the home page of a store.
 - > One shopper using a typical Web browser and the other using a cellular phone. Clearly, the same home page should not be displayed to each shopper.
 - > The Web controller's responsibility is to accept the request, then based upon information in the command registration framework, determine the view that each shopper receives.

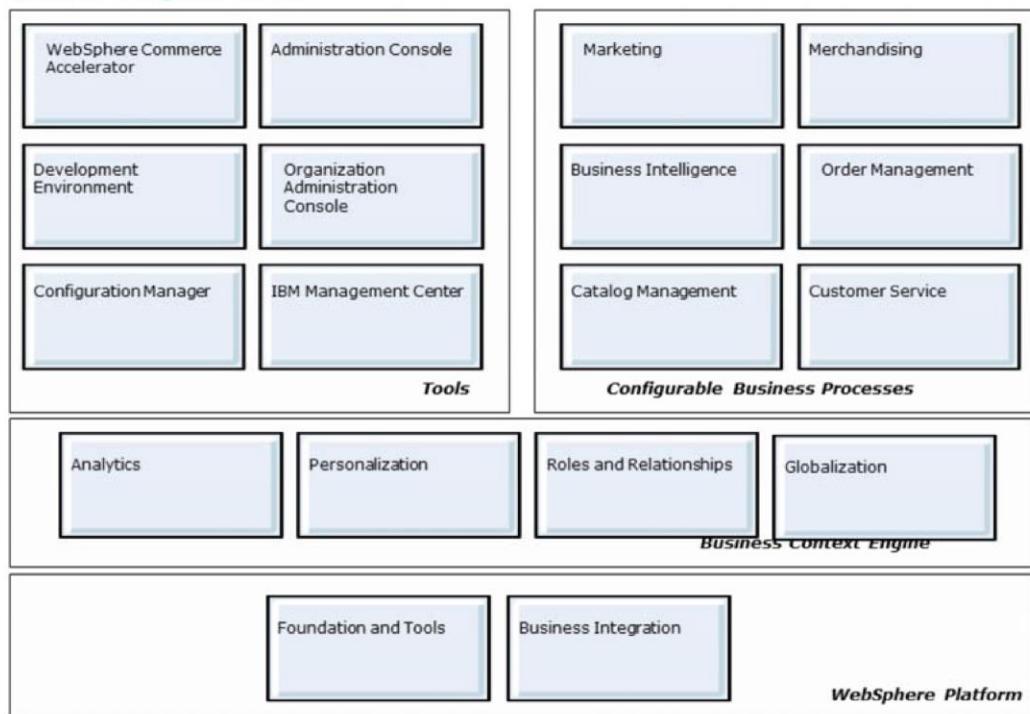
© Copyright IBM Corporation 2016

Display pages return a response to a client. Typically, display pages are implemented as JSP pages.

In order to support multiple device types, a URL access to a page should use a view name, not the name of the actual JSP file.

The main rationale behind this level of indirection is that the JSP page represents a view. The ability to select the appropriate view (for example, based on locale, device type, or other data in the request context) is highly desirable, especially since a single request often has multiple possible views. Consider the example of two shoppers requesting the home page of a store, one shopper using a typical Web browser and the other using a cellular phone. Clearly, the same home page should not be displayed to each shopper. The Web controller's responsibility is to accept the request, then based upon information in the command registration framework, determine the view that each shopper receives.

Core components



© Copyright IBM Corporation 2016

Development layers of WebSphere Commerce

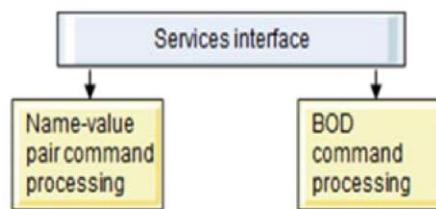
After completing this topic, you should be able to describe the following components:

- Business logic layer:
 - Name-value pair processing commands.
 - BOD processing commands.
- Persistence layer:
 - Access beans and Enterprise JavaBeans.
 - Data service layer.

© Copyright IBM Corporation 2016

Business logic layer

- Business Logic Layer supports two methods of command processing:
 - Traditional name-value pair processing.
 - SOA-compliant processing of Business Object Documents (BODs).
- Both methods use the WebSphere Commerce command framework.
- Name-value pair processing:
 - Controller commands.
 - Task commands.
- BOD processing (Recommended):
 - Get, Change, Process, Sync



© Copyright IBM Corporation 2016

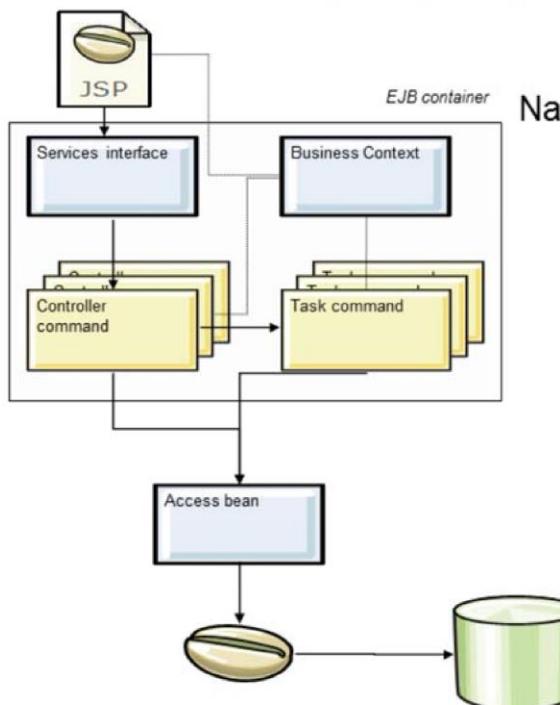
The business logic layer provides services to return data or run business logic. Business logic in the Business Object Documents (BOD) command framework is organized into service modules. The SOI implementation of the business logic layer is fronted by services, which transform the OAGIS messages (BODs) to name-value pairs for processing. This makes for easy integration with existing WebSphere Commerce or customized commands.

Business logic layer supports two methods of command processing:

1. Name-value pair processing.
2. Business Object Documents (BODs).

Users are suggested to use BOD method of command processing.

Differences in business logic layer processing (1 of 2)



Name-value pair processing:

Request for execution from the Presentation Layer.
Business token is passed to the Business Context service from the Data bean.

Façade calls appropriate controller command (or commands) which call task command (or commands).

Controller and task commands call Access beans, which delegate to EJB beans.

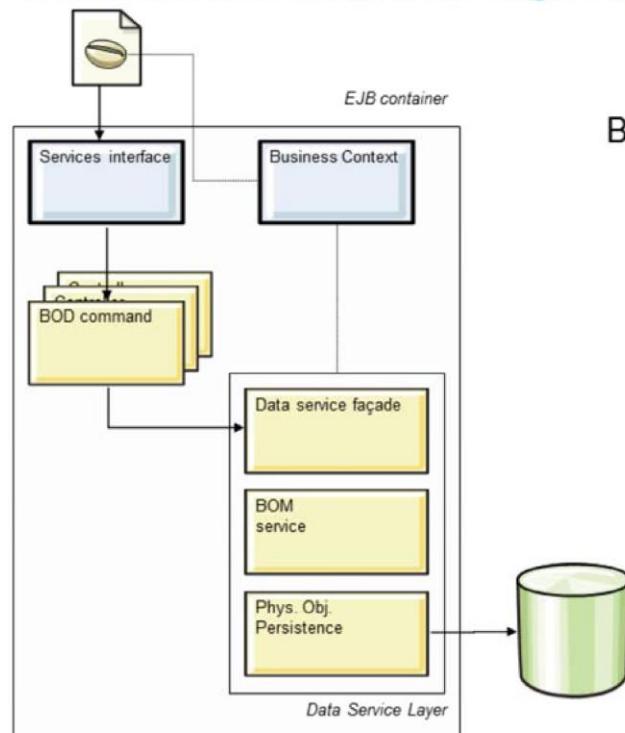
© Copyright IBM Corporation 2016

One can think of the differences in business logic processing as direct and indirect. Name-value pairs, although they offer a level of indirection, are a more direct approach for handling business logic. They require the developer to build controller and task commands (as Java classes) which encapsulate the business logic. Either controller or task commands might then call access beans, which wrap the entity bean. Access beans provide a facade like Java for an EJB, and are therefore much easier to use than calling an EJB directly.

The Business Context service is a useful service for extracting and conceptualizing data, which might not necessarily have an influence over the business logic, such as language information or environment variables, passed from the request. This separation allows the business logic to operate in a way that is independent of language, store, entitlement criteria (such as gold and platinum-level memberships).

The business logic, and Business Context Service, is discussed in greater detail later in the course.

Differences in business logic layer processing (2 of 2)



BOD processing:

- Requests for execution from the Presentation Layer are in SDO format.
- Leverages the Business Context service.
- Contains Pre-built BOD processing commands.
- Commands call the Data Service Layer to access the database.

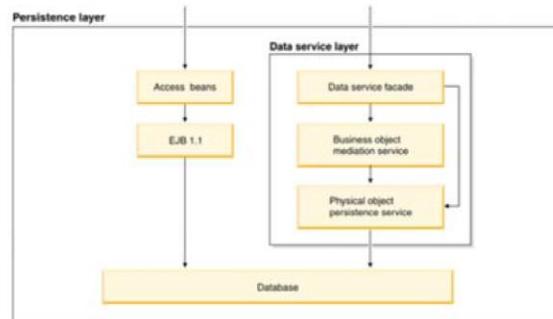
© Copyright IBM Corporation 2016

The Business Object Document (BOD) processing is a more indirect, and hence flexible, method of business logic processing. Requests for execution from the presentation layer are in service data objects (SDO) format.

The BOD processing commands are, usually, pre-built. However, they are flexible enough to include extensions, and perform typical sorts of functions, such as retrieving data, processing instructions, modifying data, or synchronizing systems (Get, Process, Change, or Sync verbs).

Persistence Layer

- The persistence layer is responsible for storing the application state
- There are two frameworks used, depending on the business logic framework:
 - NVP-commands (controller commands) use EJBs (stateless and container-managed entity beans)
 - BOD commands uses the Data Service Layer
 - SQL queries are isolated within the data service layer

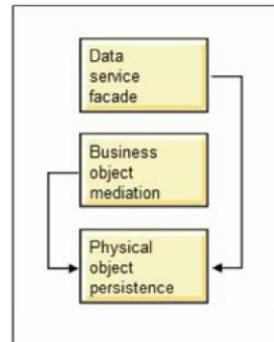


© Copyright IBM Corporation 2016

The persistence layer maps these structured objects to the persistence implementation to perform the data retrieval or updates. The persistence layer accepts structured data objects (SDOs) which are transformed (mediated) into objects called physical SDOs. Physical SDOs are stored in the data service layer. All persistence-specific assets, such as SQL queries, are isolated within the data service layer. To simplify maintenance, the query template file stores SQL.

Data service layer

- Abstraction for data access.
- Independent of the physical schema.
- Purpose:
 - Functions independent of the framework.
 - Transforms data that is retrieved from database into Java objects (implemented as SDO).
 - Offers bidirectional transformation between physical SDOs (schema) and logical SDO (classes).
 - Allows user to perform CRUD operations on physical or logical SDOs.
- Services of the DSL:
 - **Data service façade:** Interface for accessing data.
 - **Business object mediation:** Initializes classes (mediators) that transform physical to logical data.
 - **Physical object persistence:** Provides mediators access to physical data, translating XPath to SQL.



© Copyright IBM Corporation 2016

The data service layer (DSL) provides an abstraction layer for data access that is independent of the physical schema.

The purpose of the data service layer is to provide a consistent interface (called the data service facade) for accessing data, independent of the object-relational mapping framework (such as EJB, DAS, or JPA). The abstracted mapping framework is used to transform the data that is retrieved from the database into a collection of Java objects. These objects are implemented as physical service data objects (SDOs).

The data service layer performs bidirectional transformation between physical SDOs and logical SDOs. It allows you to perform Create, Retrieve, Update, and Delete (CRUD) operations on the logical SDOs. Alternately, the data service layer also lets you do CRUD operations directly on the physical data, bypassing the logical schema altogether.

The DSL consists of three pieces: the business object mediation service (BOM), the physical persistence service, and the data service facade.

The business object mediation service initializes mediators. These mediators are classes that transform between the logical and physical representations of the domain model. It allows the business logic layer to deal only with the logical representation of the data.

The mediators access the physical data through the physical persistence service. This service translates XPath queries to SQL queries.

WebSphere Commerce data model

- WebSphere Commerce Data Model is designed for data integrity and persistence.
- WebSphere Commerce is a database-driven application:
 - 700+ database tables
 - 700+ EJB
- Tables exist to persist instance data, such as:
 - Organizational structure
 - Storefront operational data
 - Access control
 - Many others
- Tables:
 - Use many constraints
 - Use indexes to improve data retrieval.

© Copyright IBM Corporation 2016

To control the behavior of WebSphere Commerce, you control the data. For instance, if you want to get a price on a product or an item, the software retrieves data from the database to determine whether you have a certain contract or promotion. The price is retrieved based on the data.

Many tables and EJB beans come with the framework, so in most cases there are places to put data and ways to retrieve the data.

It is a best practice to use the existing data framework where possible.

Subsystem data models in WebSphere Commerce

- Catalog:
 - Catalogs, categories, products, attributes, groupings.
- Marketing:
 - Promotions, precision marketing.
- Member:
 - Access control, authentication, member (participants)
- Order management:
 - ATP, calculation, pricing, payment, procurement store, shipping, tax.
- System:
 - Collaboration, command, flow, scheduler, messaging, system, user traffic.
- Payment:
 - Tables that are related to the Payments subsystem.
- Trading:
 - Auctions, contract, RFQ
- Workspaces
 - Relationship of database tables that are used in workspace.
- Gift Center (optional):
 - Relationship of database tables pertinent to the IBM Gift Center application.
- Store:
 - Store locator, Commerce Composer
- Search:
 - Search Engine optimization.

© Copyright IBM Corporation 2016

There are more than 700 tables in the WebSphere Commerce schema. The data model is broken out into subsystems.

Catalog:

-Attribute, catalog, catalog entry, merchandising association, package, dynamic kits, product set.

Marketing:

-Collateral and promotions, e-coupon, e-promotions, marketing experimentation, marketing layer.

Member:

-Access control, access logging, authentication, member.

Order management:

-ATP, billing, calculation, country and state code, currency, fulfillment, jurisdiction, Language, Offer, Order, order item, payment, pricing, procurement, quantity unit, receipt, shipping, store, suborder, tax, vendor, views, and temporary tables.

System:

-Collaborative workspace (business edition), command, customer care, flow, job scheduler, messaging, pervasive, staging server, system, user traffic.

Trading:

-Auctions, contract, request for quote (RFQ), winning bids.

Payment:

-Payment subsystem integrates with payment service providers to handle payments.

Workspaces:

-Workspaces are isolated testing environments in which development teams might modify the commerce solution without affecting the live data.

Instructor notes:

Purpose — WebSphere Commerce provides a robust data model with all the subsystems necessary to support an Internet commerce site.

Details — The data model grouping provides a developer with the ability to locate a table through a drill-down method. It is highly recommended that you provide the students with the knowledge of how to locate and use the data model documentation in the information center. Start with the following link and demonstrate how to navigate to a specific data model

Additional information — *WebSphere Commerce Information Center Data Models:*
https://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.database.doc/refs/rdb_datamodel_index.htm

Transition statement — Next: WebSphere Commerce supporting services

WebSphere Commerce supporting services

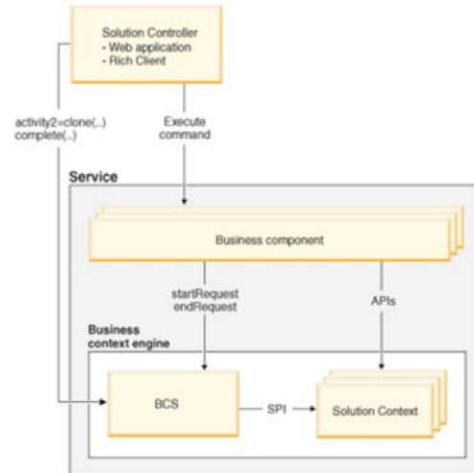
After completing this topic, you should be able to:

- Describe the purpose and function of the business context service.
- Describe the purpose and function of web services in WebSphere Commerce.
- Obtain an overview of security and access control.

© Copyright IBM Corporation 2016

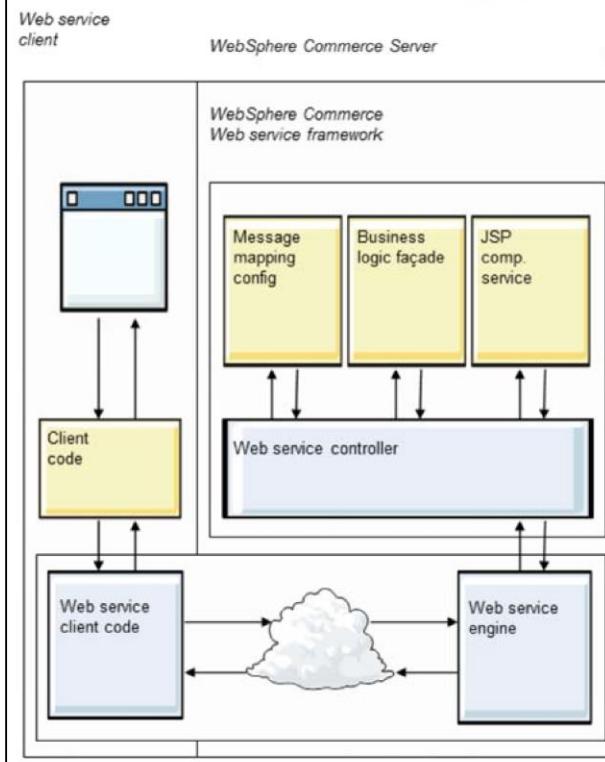
Business context services

- Business contexts:
 - Contexts establish an execution environment that affects the output of a business component.
 - Clients do not invoke contexts directly; business components use the information present in a context to fulfill operations.
- Benefits:
 - Enablement of generic components.
 - Tailored content and experience.
 - Precisely targeted offers.
 - Enforcement of business policies.
 - Appropriate prices, entitlements, and terms for a particular user.



© Copyright IBM Corporation 2016

Web services in WebSphere Commerce



- WebSphere Commerce as a service provider:
 - Enabling business operations as externally accessible web services makes WebSphere Commerce a service provider.
 - Web service deployment models usually have a central server with published WSDL defining the services.
 - External clients, web applications, or rich client applications connect to the central server and invoke services that the publicly available WSDL defines.

© Copyright IBM Corporation 2016

Processing the request

The WebSphere Commerce web service framework uses the approach of mapping the XML (SOAP body) request into the name-value pair parameters that are passed to the service command that is executed.

In terms of overall web service request handling, the WebSphere Application Server web service engine is responsible for delegating the request to the WebSphere Commerce web service framework. The framework is responsible for processing the request and generating the response.

Processing the request consists of:

- Resolving credentials to associate with the request.
- Converting the SOAP body into name-value pairs.
- Mapping to a controller command.
- Executing the command.
- Using the JSP composition service to create the response.

From the high-level perspective, success and application exceptions are handled in a similar fashion. The only difference is the JSP page is used to compose the response: when an exception occurs, the error view determines the JSP page that composes the response.

Creating the response

The WebSphere Commerce web service framework leverages the JSP composition service for creating the XML response to a web service request. This response allows the user to simply modify the JSP template file in order to insert additional nodes in the response instead of having to provide Java code for this capability.

JSP page resolution differs somewhat between the WebSphere Commerce Struts-based Web application and the WebSphere Commerce web services:

- Struts-based Web application:
 - The result of executing the business logic is a set of response properties, one of which is the name of the view to be used to determine which JSP page to invoke.

- The invoked JSP page generates an HTML response.
- Web services:
 - The view for determining the response is specified in the mapping configuration. The view in the response properties is only used if no view is found in the mapping configuration.
 - The view for web service response uses a JSP page to generate an XML response, which is returned as the response message to the web service request. The Web service engine handles adding the necessary SOAP envelope to the response, and the web service framework provides the capability of converting the XML JSP response to the SOAP element.

Advantages of web services in WebSphere Commerce

- Uses WebSphere Application Server web service engine:
 - Hides the complexity of SOAP.
- Reuses existing WebSphere Commerce assets:
 - Existing programming model.
 - Existing integration technology.
- Designed for customization:
 - XML request message mapping.
 - JSP response building.
- Inbound and outbound web service implementations.

© Copyright IBM Corporation 2016

Elements of access control

- **Users:**
 - People that use the system.
 - Grouped into relevant access groups (UserGroup).
 - Roles are used to determine membership in an access group.
 - Roles are assigned to users on a per organization basis.
- **Actions:**
 - Activities that Users can perform on a Resource.
 - Actions can be grouped into relevant groups (ActionGroup).
- **Resources:**
 - Entities, such as JSPs and commands, that are protected
 - Can be grouped into relevant groups (ResourceGroup).
- **Relationships:**
 - Define connection between Users and Resources (such as Owner, Editor, Reader).

© Copyright IBM Corporation 2016

Access control in a WebSphere Commerce application 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. 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 that is used in a store is a view. A view is invoked to display a store page to customers. The views that are 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, **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 might require that a relationship between the user and the resource are satisfied. For example, users might only be allowed to display the orders that they create.

Data Load in review

- Introduced in V7 to reduce total cost of data loading.
- Streamline and load data in a single command, that is, CSV file to database
- Loading based on business objects.
- Benefits:
 - More scalable.
 - Better performance.
 - Business rule enforcement.
 - Better diagnostics and error reporting.
- In V8 business objects supported:
 - Catalog
 - Inventory
 - Price and Catalog filter
 - Member
 - Location
 - Commerce Composer
 - Promotions
 - Marketing

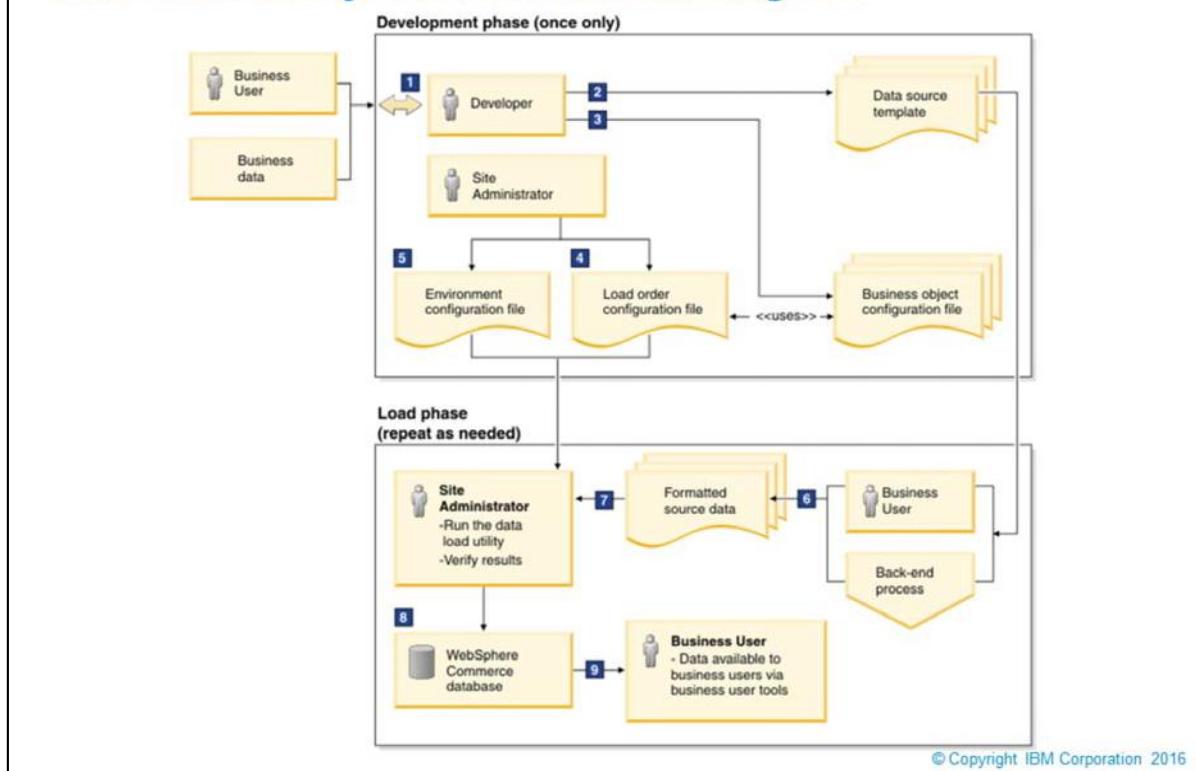
© Copyright IBM Corporation 2016

The Data Load utility streamlines the process and reduces the total cost of loading data. A single command reads the input data and writes it to the database. There is no requirement to generate intermediate files. The solution is based on the WebSphere Commerce business object model, and you need to understand only the WebSphere Commerce business object schema instead of the physical database schema. You need to have CSV input files, and then map the columns in the CSV file to the components of the logical business objects using XPath notation.

The Data Load utility performs the following functions in a single operation:

- a. Reads data from a source.
- b. Transforms the source data to WebSphere Commerce business objects.
- c. Allocates and resolves WebSphere Commerce business objects to physical data.
- d. Loads the physical data into the database.

Data Load utility: User interaction diagram



1. The business user provides the developer with the business data.
2. The developer creates a data source template, which defines how source data must be formatted before being loaded.
3. The developer also creates the business object configuration file, which defines how the Data Load utility maps the input data to the business object and how to transform the business object to physical data.
4. The site administrator uses the business object configuration file to define and create the load order configuration file.
5. The site administrator sets the store and database settings in the environment configuration file.
6. The business data is formatted according to the rules of the data source template before being loaded into the database.
7. The formatted source data is provided to the site administrator.
8. The site administrator runs the Data Load utility along with the three configuration files (environment, load order, and business object configuration files) to load the formatted source data into the WebSphere Commerce database. After running the utility, the site administrator also verifies the results of the load.
9. The business data that the business user manages is available in WebSphere Commerce.

Command-line utilities

- Acpload
- Dataload
- Dataextract
- Dbclean
- Di-buildindex
- Di-preprocess
- Massload (deprecated)
- Stagingprop

© Copyright IBM Corporation 2016

WebSphere Commerce provides utilities for preparing and loading data into a WebSphere Commerce database. These utilities provide various functions that can be strung together in the required sequence to solve your particular commerce data management problems. The loading utilities are flexible enough to handle customizations that are made to the WebSphere Commerce schema.

The utilities use 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.

Use the loading utilities to do the following tasks:

- Load large amounts of data and to update data in your WebSphere Commerce database.
- Import data from multiple input sources in the form of ASCII and XML files into WebSphere Commerce.
- Transform data from ASCII to XML format and back again.
- Map data from one XML format to another.
- Aggregate data from multiple input streams into one aggregated database.

acupload

The acupload utility loads the XML files that contain the main access control policies into the appropriate databases.

dataload

This utility loads data from a source file into a target database. Loading the file populates and updates the WebSphere Commerce database.

dataextract

The Data Extract utility, which uses the Data Load utility framework, extracts data from the WebSphere Commerce database into an output file.

dbclean

The Database Cleanup utility (dbclean) removes unused or obsolete objects from the database.

di-buildindex

The di-buildindex utility is a wrapping utility that updates the information in the Master Index with the Data Import Handler (DIH) service to build the index. The information is updated either partially through delta index updates or completely through full index builds.

di-preprocess

The di-preprocess utility extracts and flattens WebSphere Commerce data and then outputs the data

into a set of temporary tables inside the WebSphere Commerce database. The data in the temporary tables is then used by the index building utility to populate the data into search indexes with the Data Import Handler (DIH).

massload

This utility loads an XML input file into a target database. Loading the XML file updates the WebSphere Commerce database. The massload utility allows column-level updates to a table. You can also use this utility to delete data from a database.

stagingprop

The stagingprop utility propagates staged data and managed files from the production-ready data to the production server. It consolidates the changed data from the production-ready database, and then it propagates the necessary changed data into the production database if the connection is available.

Load package reference:

<http://publib.boulder.ibm.com/infocenter/wchelp/v8r0m0/topic/com.ibm.commerce.data.doc/concepts/cmluse.htm>

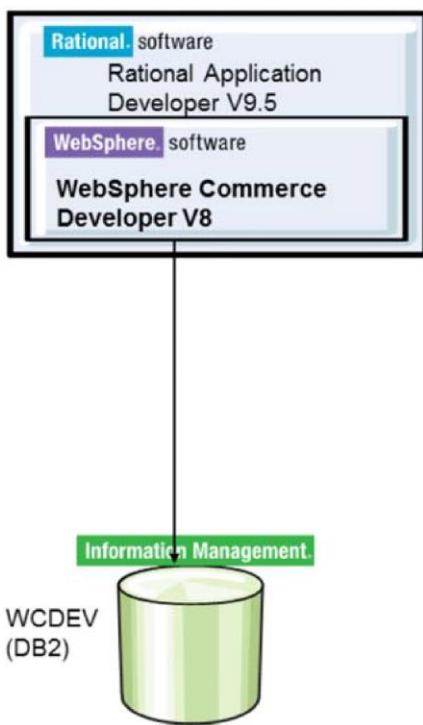
WebSphere Commerce development environment

After completing this topic, you should be able to:

- Describe the WebSphere Commerce development environment.
- Describe the default workspace.
- Identify the tools that are available with WebSphere Commerce Developer.
- Explain the problem determination and troubleshooting process.
- Identify the tiers of WebSphere Commerce for troubleshooting.
- Analyze the logs and trace files in WebSphere Commerce.
- Identify the function of IBM Support Assistant in maintaining WebSphere Commerce.

© Copyright IBM Corporation 2016

WebSphere Commerce development environment (1 of 2)



- Rational Application Developer V9.5
 - Eclipse-based development environment.
- WebSphere Commerce Developer V8
 - Leverages tools from Rational Application Developer.
 - Leverages WebSphere Application Server V8 test server.
- Development database
 - Stores development artifacts.
 - Can be DB2, Oracle, or Apache Derby (default).

© Copyright IBM Corporation 2016

Here is the recommended hardware for an instance of WebSphere Commerce Developer:

- Intel Pentium 4 1.6 GHz processor
- 3 GB RAM
- 13 GB hard disk drive space that is divided as follows:
 - Rational Application Developer Version 7.5.5.1 (10 GB).
 - WebSphere Commerce Developer (3 GB).
- DVD-ROM drive.
- Microsoft Windows:
 - Windows 7 (Professional, Ultimate) (Requires WebSphere Commerce Fix Pack 2 or higher).
 - Vista (Enterprise, Business, or Ultimate edition).
 - XP Professional, Service Pack 3.
 - 2003 Server (Standard or Enterprise edition, SP2).
 - 2008 Server (Standard, Enterprise, or Datacenter edition).

The default development repository is Apache Derby, but a developer might choose to host the repository on DB2 Universal Database V9.5.0.4 or V9.7.0.1 (or higher) or Oracle 11g, either locally or remotely.

Steps for changing the Development database for WebSphere Commerce Developer:

https://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.install.doc/tasks/tig_devchgdb.htm

WebSphere Commerce development environment (2 of 2)

- Installing WebSphere Commerce V8 Developer provides a fully configured Rational Application Developer workspace.
- The following projects are included:
 - **WC**: contains the core WebSphere Commerce EAR file.
 - **CommerceAccelerator**: contains assets for Commerce Accelerator.
 - **LOBTools**: customizable assets for IBM Management Center for WebSphere Commerce.
 - **OrganizationAdministration**: assets for the Organization Administration Console.
 - **Rest**: assets for the REST services.
 - **Search-Rest**: assets for search related REST services.
 - **SiteAdministration**: assets for the Administration Console.
 - **Stores**: module for sample store assets.
 - **WebSphereCommerceServerExtensionsData**: create custom enterprise beans and other data assets.
 - **WebSphereCommerceServerExtensionsLogic**: create new logic, such as Java classes.
 - **WebServicesRouter**: create custom web service assets.

© Copyright IBM Corporation 2016

WebSphereCommerceServerExtensionsData, -ExtensionsLogic, and WebServicesRouter are initially empty. All custom code is recommended to be placed in these projects (unless those customizations are for one of the four consoles). These folders protect your custom code when fix packs and feature packs are applied.

It is not an accepted best practice to modify the WC project.

To work on several development projects at a time, multiple workspaces can be created by installing WebSphere Commerce Developer multiple times.

Instructor

notes:

Purpose —

Details —

Additional information —

Transition statement — Next: WebSphere Commerce tools

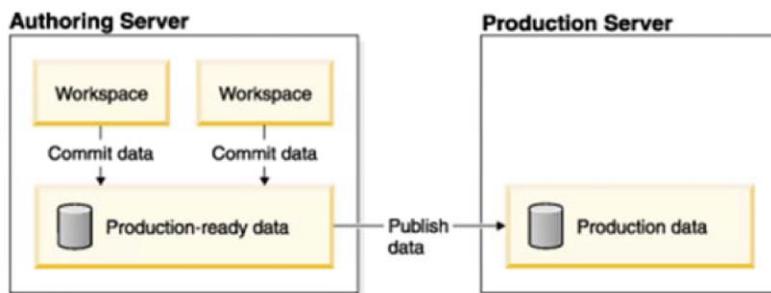
WebSphere Commerce tools

- WebSphere Commerce Administration Console:
 - Administration functions for maintaining a site or any number of stores.
- WebSphere Commerce Organization Administration Console:
 - User Administration functions, maintain security.
- WebSphere Commerce Accelerator:
 - User Console for making high-level modifications to Storefronts.
- IBM Management Center for WebSphere Commerce:
 - Suite of tools to support Merchandising and Marketing tasks

© Copyright IBM Corporation 2016

Workspaces

- Secure work area for developers to make and preview changes to managed assets.
- Does not affect production environment:
 - Similar to having a private copy of managed production assets.
- Workspaces offer the following features:
 - Task groups and tasks for dividing work within workspaces.
 - Defined roles for managing workspaces, approvals, and content.
 - Separate tool for creating and administering workspaces.
 - Locking policies to control changes.
 - Various forms of commit and publish.
 - Controls how data is moved from development to production.



© Copyright IBM Corporation 2016

You can apply changes and preview changes without affecting managed assets outside the workspace. Once you are happy with the changes you made in a workspace, you can commit the changes to the production database, and begin to see the effects on your site.

Workspaces are only available in WebSphere Commerce Professional and Enterprise editions

Workspaces are not used in this course. 6T223/6T222/WC730: *WebSphere Commerce Administration* offers hands-on experience with managing and administering workspaces.

WebSphere Commerce problem determination

- Available resources:
 - WebSphere Commerce Knowledge Center:
 - Contains migration, installation, configuration information and more.
 - Contains several self-help tutorials.
 - IBM Education Assistant:
 - Provides self-help modules.
 - Offers some voice recordings and visual assistance.
 - Technical documents:
 - Supplemental technical information.
 - IBM Developer Works:
 - Offers specific examples.
 - Redbooks:
 - Highly technical explanations of specific tasks

© Copyright IBM Corporation 2016

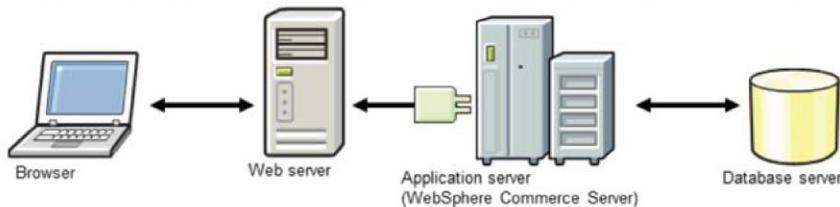
Self-sufficiency saves you time, money, and frustration. Businesses do not want to become dependent on IBM Support or Services.

Effective problem determination saves time, working through the issue yourself before contacting IBM. When determining problems, it is important to narrow down the specific issue or progress the problem determination so IBM Support can pick up where you left off.

As you understand what causes problems, you learn to avoid problems altogether, or learn how to solve them quicker.

Problem areas for examination

- Runtime issue could be anywhere along the path:
 - **Browser:** Client-side JavaScript, Ajax requests, cookie acceptance, cookie, or URL limits, windows.
 - **Web Server:** Virtual hosts, SSL certificate, rewrites, redirects, ports.
 - **WebSphere Application Server plug-in:** Mapped modules, cluster, load balancing, transports, virtual hosts.
 - **Application Server (WC & Solr):** Servlet filters, Commerce servlets, caching.
 - **Database:** connections, SQL, contention, data itself.
 - **External systems:** messaging channels, external integration points



© Copyright IBM Corporation 2016

The basic topology of a WebSphere Commerce solution includes these components. It becomes more complex as other elements, such as a firewall and clusters, are introduced.

Problems might occur at any one of these elements, and even more so when other elements are introduced. When determining a problem, you need to know what to look for at each tier. When focusing on WebSphere Commerce, component-specific Must Gathers are a good place to start. *SystemOut.log* and *trace.log* can point you to the next step:

- No request at all in log (check the web server logs or browser).
- See the logged request, but input seems wrong (check browser).
- Application exceptions (application itself is the issue, or the data).
- Rollback, timeout, locking exceptions (consider the database).
- Simply wrong results (check the application code, or the data and configuration).

WebSphere Commerce logs

- Web server log:
 - `httpd.conf` defines the virtual hosts.
 - `error.log` contains errors encountered by web server.
 - `access.log` contains all requests made to and served by web server.
- WebSphere Application Server plug-in:
 - Check the `plugin-cfg.xml` file.
- Application server and WebSphere Commerce default files:
 - `native_stderr.log`: contains text that is written to the `stderr` stream, contains verboseGC data useful for performance analysis.
 - `native_stdout.log`: contains text that is written to the `stdout` stream, contains verboseGC data on Solaris OS.
 - `startServer.log`: written when starting the server.
 - `stopServer.log`: written when stopping the server.
 - `SystemErr.log`: any system error while the server is running.
 - `SystemOut.log`: system output file while the server is running.
 - `activity.log`: logs continuous activity.
 - `trace.log`: If trace is enabled, it logs the components trace messages while the service is running.

© Copyright IBM Corporation 2016

IBM Support Assistant for WebSphere Commerce

- Ties to many key sites such as WebSphere Commerce Zone.
- Offers problem determination, product flashes, and technotes.
- Alerts user of APARs, fixes, and utilities.
- News and preventive service planning.
- Contains:
 - Product documentation.
 - Publications.
 - Commerce news.
 - Preventive service planning.
 - Upcoming Webcasts.
- Automated Commerce data collectors.

© Copyright IBM Corporation 2016

IBM Support Assistant data collectors

- Data collection can be tedious for some components.
- Automated collectors within ISA help this process:
 - Commerce collectors are one aspect of the Commerce plug-in download for ISA.
- In-line with Commerce “Must Gather” documents for IBM Support.
 - Other products include WebSphere Application Server, DB2, and WebSphere Portal.
- Speeds up initial data collection and ensures that a complete set of data is collected.
- WebSphere Commerce collectors for IBM Support Assistant gather all relevant logs from application and configuration files.

© Copyright IBM Corporation 2016

Eventually, IBM Support Assistant can FTP these collected files to the PMR resources.

Discontinued functionality in version 8 (1 of 3)

- WebSphere Commerce search runtime programming model
 - The BOD-based WebSphere Commerce search runtime programming model is deprecated.
 - This includes search configurations on the WC EAR, and catalog storefront services.
- WebSphere Commerce search profiles
 - Extending search profiles is deprecated in Version 8
- OMS/ERP
 - Order Management System (OMS)/ Enterprise Resource Planning (ERP) integrations are deprecated.
- Triple DES for encryption
 - Triple DES for encryption is deprecated for WebSphere Commerce Version 7 Feature Pack 8
- Massload utility
 - The massload utility was deprecated in WebSphere Commerce Version 7 Feature Pack 6.
- Configuration noun
- PriceTCMasterCatalogWithOptionalAdjustment

© Copyright IBM Corporation 2016

Discontinued functionality in Version 8.0 (2 of 3)

- Editions

- WebSphere Commerce Express was an entry-level offering of WebSphere Commerce.

Recommended action: The following WebSphere Commerce editions are available: WebSphere Commerce Professional provides a powerful customer interaction platform to help midsize companies offer personalized, cross-channel shopping.

- WebSphere Commerce Enterprise provides a sophisticated platform for high-volume B2C and B2B business models and multiple sites.

- Operating systems and environments

- Support for operating systems that use 32-bit architecture is discontinued.

Recommended action: Support for 64-bit operating systems continues.

© Copyright IBM Corporation 2016

Discontinued functionality in Version 8.0 (3 of 3)

- WebSphere Commerce Patterns
 - WebSphere Commerce provides virtual system patterns and virtual application patterns are discontinued.
- Starter stores
 - The following starter stores and add-on store SAR files were removed:
 - Brazil
 - Elite
 - Madisons
 - MayUJoy
- Sample stores
 - In WebSphere Commerce Version 8.0 the following sample stores were removed:
 - Advanced B2B direct sample store
 - Consumer direct sample store
 - Demand chain sample stores

© Copyright IBM Corporation 2016

Unit summary

This unit was designed to enable you to:

- Start a development project that implements WebSphere Commerce V8.
- Describe new features and functions in WebSphere Commerce V8.
- Describe the runtime architecture of WebSphere Commerce.
- Understand the Application Layers
- Understand WebSphere Commerce functional overview
- Understand WebSphere Commerce Design Patterns
- Describe the development model for WebSphere Commerce.
- Summarize the Business Logic and Persistence Layers.
- Troubleshoot and debug WebSphere Commerce applications.

© Copyright IBM Corporation 2016

References

- WebSphere Commerce V8 Information Center:
https://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/landing/wc_welcome.html
- Spring 4.1.7
<http://docs.spring.io/spring/docs/4.1.7.RELEASE/spring-framework-reference/htmlsingle/>
- Apache Struts:
<http://struts.apache.org>
- Open Application Group (OAGi) Business Object Document architecture:
<http://www.oagi.org>
- Dojo Foundation Toolkit 1.8:
<http://www.dojotoolkit.org>
- Service Data Objects (SDO):
<http://www.osoa.org>

© Copyright IBM Corporation 2016

References

- IBM Support Assistant:
<http://www.ibm.com/software/support/isa/>
- IBM Education Assistant for WebSphere Commerce:
http://publib.boulder.ibm.com/infocenter/iedusasst/v1r1m0/index.jsp?topic=/com.ibm.iea.wcs/plug_in_coverage.html
- IBM Support Portal:
<http://www.ibm.com/support/entry/portal/>
- WebSphere Commerce Zone:
<http://www.ibm.com/developerworks/websphere/zones/commerce>
- IBM Redbooks:
<http://www.redbooks.ibm.com>

© Copyright IBM Corporation 2016

Additional training

For additional training needs, you have several options available for registering.

You can register	At
On site	This or any of our training facilities
Online	http://ibm.com/training/
Email	ibmswedu@us.ibm.com
By telephone	1-800-IBM Teach (1-800-426-8322)
By fax	208-692-6130

© Copyright IBM Corporation 2016

For additional training needs, you have several options available for registering like Onsite, Online , Email , by telephone or by Fax .

Unit 2

WebSphere Commerce V8 Data Model, Schema and Configuration Files

© Copyright IBM Corporation 2016

This section describes an overview of the WebSphere Commerce V8 Data Model, Schema and configuration files

Unit objectives

This lecture is designed to enable you to:

- Understand the WebSphere Commerce Database schema
- Understand the WebSphere Commerce Development Database
- Understand WebSphere Subsystem Data models
- Describe WebSphere Commerce Organization Structures
- Understand WebSphere Commerce configuration files and command line utilities
- Understand Access control and policies

© Copyright IBM Corporation 2016

This course is designed to enable you to understand the Introduction to WebSphere Commerce Database schema, development database and provides an overview of WebSphere subsystem data models. Also understand the WebSphere Commerce programming model, describe WebSphere Commerce organization structures and provide the overview of configuration files and command line utilities, access control and policies.

Introduction to WebSphere Commerce Database Schema (1 of 4)

- Designed for data integrity and optimal performance
- Provides several hundred tables that store WebSphere Commerce instance data
- Constraints are widely used in the database model to maintain data integrity
- Limited number of SQL-based database stored procedures are used for data intensive activities

© Copyright IBM Corporation 2016

WebSphere Commerce Database schema is Designed for data integrity and optimal performance. It Provides several hundred tables that store WebSphere Commerce instance data. To maintain data integrity, and to ease maintenance referential integrity, constraints are widely used in the database model. Indexes are used carefully on Commerce tables to avoid over-indexing and to provide a good balance between data retrieval and data manipulation activities. A limited number of SQL-based database stored procedures are used for data intensive activities.

Additional details on WC database schema can be found here:
http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.database.doc/concepts/cdb_schema_overview.htm

Introduction to WebSphere Commerce Database Schema (2 of 4)

- The supported extensions to the WebSphere Commerce schema:
 - Tables
 - Physical properties of a database table such as the table space, or physical property change is allowed
 - To support new EJB beans, or JSP pages new tables can be added
 - Do not drop or rename an existing table that can cause code breakage and migration concerns.
 - Columns
 - Increase the size of a column or change the data type to another compatible type
 - Performance impact may occur, if the size of the column is increased.
 - To customize:
 - Issue DDL statements
 - Change all JSP pages that use the columns
 - The following changes to columns are not supported
 - Changing the data type of a column to a non-compatible type
 - Adding a column
 - Dropping a column
 - Renaming a column

© Copyright IBM Corporation 2016

This section lists supported extensions to the WebSphere Commerce schema:

- Tables
- Columns
- Primary and foreign keys
- Indexes
- Triggers
- Stored procedures
- Functions

Tables:

Physical properties of a database table such as the table space, or physical property change is allowed.

To support new EJB beans, or JSP pages new tables can be added. When you add a table, consider optimistic locking. After you migrate to a new release of WebSphere Commerce, any customized tables must be reapplied.

Do not drop or rename an existing table. This action can cause code breakage and migration problems.

Columns:

You can increase the size of a column or change the data type to another compatible type. There might be a performance impact if the size of the column is increased. To customize, issue DDL statements, then change all JSP pages that use the columns. Data might need to be moved, for example DB2 long to CLOB.

The following changes to columns are not supported :

Changing the data type of a column to a non-compatible type, adding a column, dropping a column, renaming a column

Introduction to WebSphere Commerce Database Schema (3 of 4)

- Primary and foreign keys
 - Foreign keys can be added between a new custom table and an existing table, or between two custom tables
 - The following changes are not supported:
 - Changing primary keys
 - Removing primary keys
 - Changing existing foreign keys
 - Add new foreign keys between existing tables
- Indexes
 - The following changes are supported
 - Adding new indexes.
 - Altering an existing index
 - Adding a column or more to the end.
 - Dropping a column or more.
 - Dropping an existing index.

© Copyright IBM Corporation 2016

Primary and foreign keys:

Foreign keys can be added between a new custom table and an existing table, or between two custom tables. The cascade delete condition can be changed. If the tables are not used, existing foreign keys can be removed.

The changes which are not supported are changing primary keys, removing primary keys, changing existing foreign keys and add new foreign keys between existing tables.

Indexes:

If you drop an index, you might encounter a database performance issue when new fixes or features are added to your site. When you apply a maintenance package, you can introduce new database queries that rely on the dropped index in your instance. Without the index in place, the query performance can be negatively impacted. When you do drop an index, ensure that you monitor your database performance closely after applying a maintenance package. If your database performance is negatively affected, consider adding the dropped index back onto the appropriate table.

The following changes are not supported:

- Adding unique indexes
- Changing the uniqueness
- Functions on columns and indexes.
- Over-indexing

Introduction to WebSphere Commerce Database Schema (4 of 4)

- Triggers
 - Customize triggers by using DDL statements and DB procedural language
 - Additional staging, performance, and business auditing triggers can be added
- Stored procedures
 - Adding new stored procedures for customized code.
 - The following changes are not supported:
 - Changing an existing procedure
 - Adding new stored procedure to existing code
 - Dropping an existing stored procedure
- Functions
 - WebSphere Commerce schema does not contain any custom functions
 - Custom functions are not supported
 - Adding column functions such as AVG, Max, and Min are allowed

© Copyright IBM Corporation 2016

Triggers:

You can customize triggers by using DDL statements and DB procedural language. Any customization to triggers might have a performance impact as the new triggers could slow down DDL statements on other tables.

Additional staging, performance, and business auditing triggers can be added.

The following changes are not supported:

- Changing existing triggers
- Changing the timing
- Dropping existing triggers

Stored procedures:

Adding new stored procedures for customized code is allowed.

The following changes are not supported: Changing an existing procedure, Adding new stored procedure to existing code, Dropping an existing stored procedure

Functions :

WebSphere Commerce schema does not contain any custom functions. Custom functions are not supported.

Adding column functions such as AVG, Max, and Min are allowed

WC Development Database (1 of 3)

- The development environment that is configured to work with Apache Derby by default
- Other Databases supported are
 - DB2 Universal Database for Windows, running locally or remotely.
 - Oracle Database for Windows, running locally or remotely.
 - For IBM i OS operating system DB2 for i running remotely.
- Reference on how to change database type:
 - https://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.install.doc/tasks/tigdevchgdb.htm?lang=en-us
- The options that are available to use different database for
 - Production
 - Development
- Derby development database has all of the starter store archives pre-published

© Copyright IBM Corporation 2016

The development environment is initially configured to work with Apache Derby, which is included with WebSphere Commerce Developer. You can configure IBM WebSphere Commerce Developer to use the other databases.

DB2 Universal Database for Windows, running locally or remotely. Oracle Database for Windows, running locally or remotely.

IBM i DB2 for i running remotely.

For information about configuring IBM WebSphere Commerce Developer for the other databases refer knowledge center here: https://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.install.doc/tasks/tigdevchgdb.htm?lang=en-us

Use different database management systems for development and production. The Derby development database in the initial installation of IBM WebSphere Commerce Developer has all of the starter store archives pre-published.

WC Development Database (2 of 3)

- The database can be accessed
 - While the Test server is running:
 - Use the below URL to access the database
 - <http://localhost/webapp/wcs/admin/servlet/db.jsp>
 - Above URL works for any valid development database type.
 - Test server is idle
 - Use native database tools to access the database
 - For Derby:
 - Use the *ij.bat* utility to start an SQL session
 - Issue the command *connect '..\db\mall'*; to connect to the WebSphere Commerce database
 - Issue *exit* command to exit
 - Note: Only one process at a time can connect to the database when Derby is used

© Copyright IBM Corporation 2016

Accessing the development database directly:

We can access the development database while the test server is running or idle.

WebSphere Commerce Test Server is running:

For all development database types, to issue SQL statements against the WebSphere Commerce development database while the WebSphere Commerce Test Server is running, use the following URL from the same machine that is running WebSphere Commerce Developer:

<http://localhost/webapp/wcs/admin/servlet/db.jsp>

This URL will provide an entry field where you can issue SQL statements that will run against your development database. This URL works for any valid development database type.

WebSphere Commerce Test Server is not running:

Use native database tools to access the database, for example a DB2 command window.

To issue SQL statements against the WebSphere Commerce development database when the WebSphere Commerce Test Server is not running, use the *ij.bat* utility to start an SQL session:

1. Start a command prompt session.
2. Navigate to the WCDE installdir\bindirectory.
3. Issue the following command to start an SQL session:*ij.bat*
4. Issue the following command to connect to the WebSphereCommerce database: *connect '..\db\mall'*; You can now issue SQL statements that will run against your development database. Ensure that you end each of your SQL statements with a semicolon.

5. To end your session, issue the following command: exit; Note: Only one process at a time can connect to the database.
Therefore, you must ensure that the WebSphere Commerce Test Server is stopped before beginning an ij.bat session and, conversely, that the ij.bat session is ended before starting the WebSphere Commerce Test Server.

WC Development Database (3 of 3)

- Resetting the Apache Derby database to contain only bootstrap data
 - restoreDefault.bat
 - Script to restore the development environment to its default setup
 - Script resets WebSphere Commerce Developer to its default settings.
 - resetdb.bat
 - Script resets the database to its bootstrap configuration
 - resetstores.bat
 - Script resets the Stores Web module in your workspace to its bootstrap configuration
 - The Stores Web module does not contain starter store resources

© Copyright IBM Corporation 2016

The initial installation of WebSphere Commerce Developer contains pre-published starter stores using Apache Derby as the development database type. You can reset your database and workspace so that they do not contain any starter store resources.

Before you begin

Ensure that the WebSphere Commerce server is stopped.

Procedure:

- 1) Go to the WCDE_installdir\bindirectory.
- 2) Run the restoreDefault.bat script. When prompted, confirm that you want to restore the development environment to its default setup by typing YES. This script resets WebSphere Commerce Developer to its default settings.
- 3) Run the resetdb.bat script. When prompted, confirm that you want to reset the database by typing YES. This script resets the database to its bootstrap configuration. This is applicable to Derbyonly.
- 4) Run the resetstores.bat script. When prompted, you must confirm that you want the stores information in the database to be reset by typing YES. This script resets the Stores Web module in your workspace to its bootstrap configuration. After you run the script, the Stores Web module does not contain starter store resources.
- 5) If you applied a WebSphere Commerce fix pack, run theupdatedb.bat script to update the database to the appropriate level.
- 6) Use the Administration Console to publish only those starterstores that are of interest to you.

Summary of Subsystem Data Models in WC

- Catalog:
 - Catalogs, categories, products, attributes, groupings.
- Marketing:
 - Promotions, rules server, product advisor, WebSphere Commerce Analyzer.
- Member:
 - Access control, authentication, member (participants)
- Order management:
 - ATP, calculation, pricing, payment, procurement store, shipping, tax.
- System:
 - Collaboration, command, flow, scheduler, messaging, system, user traffic.
- Payment:
 - Tables that are related to the Payments subsystem.
- Trading:
 - Auctions, contract, RFQ
- Workspaces
 - Relationship of database tables that are used in workspace.
- Gift Center (optional):
 - Relationship of database tables pertinent to the IBM Gift Center application.
- Store:
 - Store locator, Commerce Composer
- Search:
 - Search Engine optimization.

© Copyright IBM Corporation 2016

There are more than 700 tables in the WebSphere Commerce schema. The data model is broken out into subsystems.

Catalog:

-Attribute, catalog, catalog entry, merchandising association, package, dynamic kits, product set.

Marketing:

-Collateral and promotions, e-coupon, e-promotions, marketing experimentation, marketing layer.

Member:

-Access control, access logging, authentication, member.

Order management:

-ATP, billing, calculation, country and state code, currency, fulfillment, jurisdiction, Language, Offer, Order, order item, payment, pricing, procurement, quantity unit, receipt, shipping, store, suborder, tax, vendor, views, and temporary tables.

System:

-Collaborative workspace (business edition), command, customer care, flow, job scheduler, messaging, pervasive, staging server, system, user traffic.

Trading:

-Auctions, contract, request for quote (RFQ), winning bids.

Payment:

-Payment subsystem integrates with payment service providers to handle payments.

Workspaces:

-Workspaces are isolated testing environments in which development teams might modify the commerce solution without affecting the live data.

Instructor notes:

Purpose — WebSphere Commerce provides a robust data model with all the subsystems necessary to support an Internet commerce site.

Details — The data model grouping provides a developer with the ability to locate a table through a drill-down method. It is highly recommended that you provide the students with the

knowledge of how to locate and use the data model documentation in the information center.

Start with the following link and demonstrate how to navigate to a specific data model

Additional information — WebSphere Commerce Information Center Data Models:

https://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.database.doc/refs/rdb_datamodel_index.htm

Details of Subsystem Data Models (1 of 9)

- Catalog data models
 - Relationship between database tables related to the Catalog subsystem
 - Attachment data model
 - Relationship between database tables that contain information about attachment
 - Tables : ATCHTGT, ATCHTGTDSC
 - Attribute data model
 - Relationship between tables that contain information about product attributes
 - Tables : ATTR, ATTRVAL, ATTRVALDESC
 - Attribute dictionary data model
 - Relationship between database tables that contain information about attribute dictionaries.
 - Tables : ATTRDICT, ATTRDESC
 - Catalog data model
 - The relationship between database tables that contain information about a catalog.
 - Tables : CATALOG, STOREENT
 - Catalog entry data model
 - Shows the relationship between database tables that contain information about a catalog entry.
 - Tables : CATENTRY, CATENTDESC

© Copyright IBM Corporation 2016

Data model

Any given database data model displays the relationship among database tables in the schema

The catalog data model shows the relationship between database tables related to the Catalog subsystem.

Attachment data model:

The Attachment shows the relationship between database tables that contain information about attachment.

ATCHTGT : This table holds information about the target, which is used to hold a collection of assets, for use as an attachment target.

ATCHTGTDSC : This table holds the description of an attachment target.

Attribute data model:

The attribute data model shows the relationship between database tables that contain information about product attributes.

ATT : This table holds catalog entry attributes used for descriptive or SKU resolution purposes.

ATTRVAL : This table holds the values assigned to attributes.

Attribute dictionary data model:

The attribute dictionary data model shows the relationship between database tables that contain information about attribute dictionaries.

ATTRDICT : Table used to store the attribute dictionaries configured for the site. Each attribute

dictionary is owned by a store, but can be used by children stores through store relationships. There can be only one attribute dictionary owned by any given store.

ATTRDESC : This table holds the language sensitive attribute description. Before loading attributes in multiple languages, the attributes must already exist in the database with default data.

Catalog data model:

The catalog data model shows the relationship between database tables that contain information about a catalog.

CATALOG : This table holds the information related to a catalog.

STOREENT : Each row of this table represents a StoreEntity. A StoreEntity is an abstract superclass that can represent either a Store or a StoreGroup.

Catalog entry data model:

The catalog entry data model shows the relationship between database tables that contain information about a catalog entry.

CATENTRY : This table holds the information related to a catalog entry. Examples of catalog entries include products, items, packages, and bundles.

CATENTDESC : This table holds language-dependent information related to a catalog entry.

Here are the references and information about WebSphereCommerce database schema, Cross-reference of data beans, EJB beans, and tables, Cross-reference of commands, tasks, and tables, Cross reference: Commands to beans to database tables, and Data models.

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.database.doc/concepts/cdb_schema_overview.htm

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rxejb.htm

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rgxxref.htm

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rgxref.htm

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.database.doc/refs/rdb_datamodel_index.htm

Details of Subsystem Data Models (2 of 9)

- Catalog data models
 - Catalog group data model
 - Relationship between database tables that contain information about catalog groups
 - Tables : CATGROUP, CATGRPDESC
 - Catalog filter data model
 - Relationship between database tables that contain information about catalog filtering
 - Tables : CATFILTER, CATFLTDSC
 - Merchandising association data model
 - Relationship between database tables that contain information about merchandising associations
 - Tables : MASSOCCECE, MASSOCTYPE
 - Dynamic kit data model
 - Relationship between database tables that contain information about configurable products. These configurable products are Dynamic Kit catalog entries
 - Tables : DKPREDEFCONF, DKPDCCATENTREL
 - Product set data model
 - Relationship between database tables that contain information about product sets.
 - Tables : PRSETCEREL, PRODSETDSC

© Copyright IBM Corporation 2016

Catalog group data model:

catalog group data model shows the relationship between database tables that contain information about catalog groups.

CATGROUP : This table hold the information related to a catalog group. A catalog group is similar to a generic category that can contain both other catalog groups and also catalog entries.

CATGRPDESC : This table holds the language-dependent information related to a catalog group.

Catalog filter data model:

catalog filter data model shows the relationship between database tables that contain information about catalog filtering.

CATFILTER : Each row of this table represents a catalog filter.

CATFLTDSC : Each row of this table represents a language specific description of a certain catalog filter.

Merchandising association data model:

The merchandising association data model shows the relationship between database tables that contain information about merchandising associations.

MASSOCCECE : This table holds the merchandising associations that exist between CatalogEntries.

MASSOCTYPE : This table holds all of the possible types of associations that can exist between CatalogObjects.

Dynamic kit data model:

Dynamic kit data model shows the relationship between database tables that contain information

about configurable products. These configurable products are Dynamic Kit catalog entries.

DKPREDEFCONF : Contains predefined configurations of dynamic kit catalog entries.

DKPDCCATENTREL : Contains relationships between predefined configurations and the dynamic kit catalog entries for which they are built.

Product set data model:

product set data model shows the relationship between database tables that contain information about product sets.

PRSETCEREL : This table holds the expanded (published) form of a ProductSet.

PRODSETDSC : This table holds the language-dependent information related to product sets.

Details of Subsystem Data Models (3 of 9)

- Marketing data models
 - Shows the relationship between database tables related to the Marketing subsystem.
 - Marketing data model
 - Relationship between database tables that contain information about marketing activities
 - Tables : DMACTIVITY, DMELEMENT
 - E-mail activity data model
 - Relationship between database tables that contain information about email activity
 - Tables : EMLCONTENT, EMLMSG, EMLMCREL
- Promotions data models
 - Promotion data model
 - Relationship between database tables that contain information about promotions
 - Tables : PX_PROMOTION, PX_DESCRIPTION, PX_RWDOPTION
 - The collateral and promotions data model
 - Relationship between database tables that contain information about collateral and promotions.
 - Tables : COLLATERAL, COLDESC, COLTYPE

© Copyright IBM Corporation 2016

The marketing data model shows the relationship between database tables related to the Marketing subsystem.

Marketing data model:

The marketing data model shows the relationship between database tables that contain information about marketing activities.

DMACTIVITY : A definition of the interaction with a customer to market content.

DMELEMENT : Campaign elements associated with a marketing activity.

Email activity data model:

The e-mail activity data model shows the relationship between database tables that contain information about e-mail activity

EMLCONTENT : Stores the content of the e-mail template.

EMLMSG : This table stores e-mail message templates. Each row corresponds to an e-mail message template which can be used within an e-mail activity.

EMLMCREL : Stores the relationship between EMLMSG and EMLCONTENT table.

Promotion data model:

The promotion data model shows the relationship between database tables that contain information about promotions.

PX_PROMOTION : This table contains promotions.

PX_DESCRIPTION : This table contains promotion description information.

PX_RWDOPTION : This table contains reward option information associated with a promotion

application that will be used at evaluation time.

Collateral and promotions data model:

The collateral and promotions data model shows the relationship between database tables that contain information about collateral and promotions.

COLLATERAL : This table contains data describing the advertisements used by marketing campaigns.

COLLDESC : This table holds language-dependent information related to a Collateral.

COLLTYPE : This table defines all possible types of Collateral. Examples of Collateral types are image, flash and text.

Details of Subsystem Data Models (4 of 9)

- Member data models
 - Shows the relationship between database tables related to the Member subsystem
 - Member data model
 - Relationship between database tables that contain information about users, organizational entities, user groups, and roles
 - Tables : MEMBER, MBRGRP
 - Access control data model
 - Relationships between the access control policy tables.
 - An access control policy is composed of a member group, resource group, and action group
 - Tables : ACOPOLICY, ACPOLDESC, ACRESGRP
 - Authentication data model
 - Relationship between database tables that contain information about security policies within the system.
 - Tables : USERS, ROLE, PLCYACCT

© Copyright IBM Corporation 2016

Member data models:

The member data model shows the relationship between database tables related to the Member subsystem.

Member data model:

The member data model shows the relationship between database tables that contain information about users, organizational entities, user groups, and roles.

MEMBER : Stores the list of members (participants) of the WebSphere Commerce system. A member is either a user, an organizational entity or a member group.

MBRGRP : This table stores member groups defined in the WebSphere Commerce system. A member group is a group of members.

Membership is restricted to users within a member group.

Access control data model:

The access control data model shows relationships between the access control policy tables. An access control policy is composed of a member group, resource group, and action group.

ACOPOLICY : This table stores all the access control policies in the system. Every policy refers to an action group, a member group, a resource group and optionally, a relationship.

ACPOLDESC : This table stores locale-specific information for the ACOPOLICY table.

ACRESGRP : This table stores all the access control resource groups in the system. The conditions column stores an XML document containing the constraints and attribute value pairs used for grouping the resources.

Authentication data model:

The authentication data model shows the relationship between database tables that contain

information about security policies within the system.

USERS : This table contains all users of the WebSphere Commerce system: registered users, guest users, and generic users.

ROLE : This table stores the roles defined in WebSphere Commerce. Once a role is created, you cannot change the name or description of a role using a graphical user interface tool.

PLCYACCT : This table stores the user account policies that consist of an account lockout policy and a password policy.

Details of Subsystem Data Models (5 of 9)

- The payment data models
 - Database tables that are related to the payments subsystem in Payments subsystem data model
 - Shows the relationship between database tables that contain information about the components that make up the Payments subsystem.
 - Tables : EDPATMPAY, EDPPAYINST, EDPORDER
 - Payment batch data model
 - Relationship between database tables that contain information about handling batch processing of payments.
 - Tables : PPCBATCH
 - Payment merchant support data model
 - Relationship between database tables that contain information about payment merchant support
 - Tables : MERCHANT, MERCHCONFINFO
 - Payment processing data model
 - Relationship between database tables that contain information about payment processing
 - Tables : PPCPAYTRAN, PPCPAYMENT, PPCCREDIT

© Copyright IBM Corporation 2016

Payment data models:

The payment data models show the database tables that are related to the payments subsystem in Payments subsystem data model:

The Payments subsystem data model shows the relationship between database tables that contain information about the components that make up the Payments subsystem. These components store the information to determine how payments are handled by the Payment Rule Engine or the Payment Plug-in Controller module.

EDPATMPAY : This table stores the value object Payment Instruction in WebSphere Commerce which contains the detailed information required by plug-ins to process financial transactions. The amount in the payment instruction can be used by plug-ins to determine what is the maximum target amount the payments subsystem intends to consume collectively.

EDPPAYINST : This table stores the information of the value object container Payment Instruction in Payment Rules.

EDPORDER : This table stores the information of the value object container Credit for credit-related transactions in the payments subsystem.

Payment batch data model:

The Payment batch data model shows the relationship between database tables that contain information about handling batch processing of payments.

PPCBATCH : The batch object representing the batch transaction

Payment merchant support data model:

The Payment merchant support data model shows the relationship between database tables that contain information about payment merchant support

MERCHANT : This table stores the information of the value object container Credit for credit-related transactions in the payments subsystem.

MERCHCONFINFO : This table stores the NVPs for properties of merchant information for a specific merchant configuration.

Payment processing data model:

The Payment processing data model shows the relationship between database tables that contain information about payment processing.

PPCPAYTRAN : This table stores the information of a financial transaction processed by a payment plug-in.

PPCPAYMENT : This table stores the information of the value object container Payment for payment-related transactions in the payments subsystem.

PPCCREDIT :This table stores the information of the value object container Credit for credit-related transactions in the payments subsystem.

Details of Subsystem Data Models (6 of 9)

- Store data models
 - Store data model
 - Relationship between database tables that contain information about stores
 - Tables : STORE, STOREREL, STOREENT
 - Store locator data model
 - Relationship between database tables that contain information for the store locator functionality
 - Tables : STLOC, STLOCATTR
 - Commerce Composer data model
 - Relationship between database tables that contain information about Commerce Composer
 - Tables : PLWIDGETDEF, PLWIDGETDESC, PLSTOREWIDGET, PLWIDGET
- Trading data models
 - Contract data model
 - Relationship between database tables containing information about contracts for trading positions
 - Tables : CONTRACT, STORECNTR, CATCNTR, ACCOUNT
 - Request for Quote (RFQ) data model
 - Relationship between database tables that contain information about RFQs.
 - Tables : RFQ, RFQRSPPROD, TRADING, RFQCATEGORY

© Copyright IBM Corporation 2016

The store data models show the relationship between database tables that are related to stores.

Store data model:

The Store data model shows the relationship between database tables that contain information about stores.

STORE : Each row of this table represents a store. A store is a store entity.

STOREREL : Each row of this table describes a relationship between stores.

STOREENT : Each row of this table represents a StoreEntity. A StoreEntity is an abstract superclass that can represent either a Store or a StoreGroup.

Store locator data model:

The Store locator data model shows the relationship between database tables that contain information for the store locator functionality.

STLOC : Represents store location information.

STLOCATTR : Holds language specific store searchable attributes.

Commerce Composer data model:

Commerce Composer data model relationship between database tables that contain information about CommerceComposer.

PLWIDGETDEF : Contains information about all widgets registered on the system.

PLWIDGETDESC : Contains language specific descriptions for each widget in PLWIDGETDEF.

PLSTOREWIDGET : Each row specifies the relationship between a store and available widgets. Each available widget is allowed to be added to page layouts within the specified store.

PLWIDGET : This table contains the association of widgets to page layouts.

Trading data models:

The trading data models show the relationship of database tables in the Trading subsystem.

Contract data model:

contract data model shows the relationship between database tables containing information about contracts for trading positions.

CONTRACT: Each row of this table represents a Contract. AContract is part of a Store, and represents terms and conditions that may be associated with OrderItems. Such as prices, minimum quantities, and who can use the Contract.

STORECNTR : Each row of this table indicates that a Contract is deployed in a Store.

CATCNTR : Each row of this table associates a catalog with a contract.

ACCOUNT : Each row of this table represents a business account between a Buyer organization and a Seller organization. A business account can be used to organize various trading agreements, and to specify special trading terms and conditions.

Request for Quote (RFQ) data model:

The Request for Quote (RFQ) data model shows the relationship between database tables that contain information about RFQs.

RFQ : This table holds basic RFQ data.

RFQRSPPROD : RFQ response and product relationship table. This table stores the products included in a specific RFQ Response.

TRADING : Each row in this table represents a TradingAgreement. **RFQCATEGORY** : This table holds categories created in each RFQ.

Details of Subsystem Data Models (7 of 9)

- System data models
 - Relationship of database tables used for system functions
 - Command data model
 - Relationship between database tables that contain information about the system commands
 - Tables : STOREENT, CMDREG
 - Scheduler data model
 - Relationship between database tables that contain information about the scheduler
 - Tables : SCHCONFIG , SCHORDERS
 - Messaging data model
 - Relationship between database tables that contain information about messaging
 - Tables : MSGSTORE, MSGTYPES, PROFILE
 - Mobile device data model
 - Relationship between database tables that contain information about mobile (pervasive) device options within the system
 - Tables : USERPVCDEV, PVCSESSION, PVCDEVMDL
 - Views and temporary tables data model
 - Information about views and temporary tables
 - View : INVSTFFMVW, RCPTSTVW

© Copyright IBM Corporation 2016

System data models:

The system data models show the relationship of database tables used for WebSphere Commerce system functions.

Command data model:

The command data model shows the relationship between database tables that contain information about the system commands.

STOREENT : Each row of this table represents a StoreEntity. A StoreEntity is an abstract superclass that can represent either a Store or a StoreGroup.

CMDREG : This command registry allows the server to look up an implementation and default parameters for a command interface.

Scheduler data model:

The Scheduler data model shows the relationship between database tables that contain information about the scheduler.

SCHCONFIG : This table contains all scheduled job entries. SCHORDERS : This table contains the entries for scheduled orders.

Messaging data model:

Messaging data model shows the relationship between database tables that contain information about messaging.

MSGSTORE : This table holds the messages stored as a result of using a transacted send. Temporary storage space only.

MSGTYPES : This table is used by the messaging system to store header information about the

available message types.

PROFILE : This table is used by the messaging system to store profile data. Specifically, it defines what transports are to be used for particular stores, message types, and priorities. It also holds other administrative information.

Mobile device data model:

Mobile device data model shows the relationship between database tables that contain information about mobile (pervasive) device options within the system.

USERPVCDEV : This table contains information for pervasive computing devices used by WebSphere Commerce registered users.

PVCSESSION : This table manages the session of various pervasive computing devices.

PVCDEVMDL : This table stores device model information. Records will be looked up by the value returned by the WebSphere Commerce pervasive computing adapter. Each subclass of **PVCAdapterImpl** is responsible to get a device model name from the HTTP request.

Views and temporary tables data model:

The views and temporary tables data model shows information about views and temporary tables.

INVSTFFMVW : This is a view derived from the RECEIPT and ITEMFFMCTR tables which contains the existing quantity available for an item at a Store and FulfillmentCenter.

RCPTSTVW : A view which summarizes the quantity of inventory available from the RECEIPT table for an item owned by a Store across all FulfillmentCenters.

Details of Subsystem Data Models (8 of 9)

- Content Management data model
 - The relationship between database tables that contain information about content management
 - Tables : CMMETADATA, CMWORKSPACE, CMFTSKMBREL
- Content management system integration data model
 - Relationship between database tables used by the solution
 - Tables : CMFEEDLOG
- Content versioning data model
 - Relationship between database tables that contain information about the content versioning
 - Tables : CMACTVERSN, CMVERSINFO
- Folder data model
 - Relationship between the database tables that include information about folder and folder items
 - Tables : FOLDER, FOLDERITEM

© Copyright IBM Corporation 2016

Content management data model:

The Content Management data model shows the relationship between database tables that contain information about content management.

CMMETADATA : Records the business objects that have changed and the associated workspaces, task groups, and tasks that have made that change. This table will be used for resource locking and for determining the business objects that will be committed and published to production.

CMWORKSPACE : Contains a list of valid workspaces that may be assigned to an available schema pool.

CMFTSKMBREL : Holds the task member-specific information. This relationship defines the tasks assigned to members. For example, workspaces contributor tasks, approval tasks will be stored in this table.

Content management system integration data model:

The content management system integration data model shows the relationship between database tables used by the solution.

CMFEEDLOG : This table logs information about each feed that has been processed. It also records a hash code for the record to compare against to determine if there is an update to the record.

Content versioning data model:

The content versioning data model shows the relationship between database tables that contain information about the content versioning.

CMACTVERSN : This table is part of the Content Version schema, and alias in the Commerce base schema. This table contains all content active version.

CMVERSINFO : This table is part of the Content Version schema, and alias in the Commerce base

schema. This table contains the content version meta information.

Folder data model:

The folder data model shows the relationship between the database tables that include information about folder and folder items.

FOLDER : Holds information about folders. Folders are a generic implementation of a foldering concept that allow for any type of object to be organized into a hierarchy. The same object can be in multiple folders but a sub folder can only have a single parent.

FOLDERITEM : Holds the relationships between folders and objects in folders. The same object can be in different folders and there will be one row in this table for each folder that object is contained in.

Details of Subsystem Data Models (9 of 9)

- IBM Gift Center data model
 - Relationship between database tables that contain information about IBM Gift Center
 - Tables : GRGFTREG, GRRGSTRNT, GRPERATTR
- WebSphere Commerce search engine optimization (SEO)
 - Relationship between database tables that contain information about WebSphere Commerce search engine optimization (SEO)
 - Tables : SEOPAGEDEF, SEOURLKEYWORD
- The WebSphere Commerce search data model
 - Relationship between database tables that are related to WebSphere Commerce search
 - Tables : SRCHCONF, SRCHTERMASSOC, SRCHSTAT

© Copyright IBM Corporation 2016

IBM Gift Center data model:

The IBM Gift Center data model shows the relationship between database tables that contain information about IBM Gift Center.

GRGFTREG : This table stores gift registry and wish list profile information.

GRRGSTRNT : This table stores information about a gift registry registrant.

GRPERATTR : This table stores personalization attributes for gift registry and wish list items, such as monogramming.

WebSphere Commerce search engine optimization (SEO) data model:

The WebSphere Commerce search engine optimization (SEO) shows the relationship between database tables that contain information about WebSphere Commerce search engine optimization (SEO).

SEOPAGEDEF : This table holds the SEO page definition for a particular pageName at a site/store level

SEOURLKEYWORD : This table holds the store and language specific SEO URL keyword information

WebSphere Commerce search data model:

The WebSphere Commerce search data model shows the relationship between database tables that are related to WebSphere Commerce search.

SRCHCONF : The configuration table for WebSphere Commerce search integration.

SRCHTERMASSOC : This table contains search term associations to suggest additional, different, or replacement products in search results. Search term associations include synonyms, replacement terms, and landing pages.

SRCHSTAT : This table is used for capturing search related statistics at runtime.

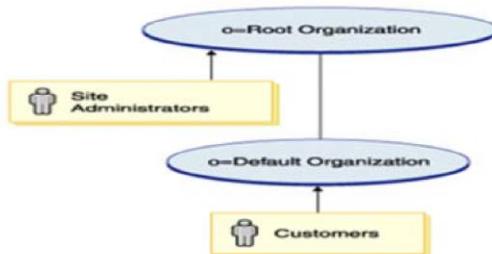
Overview of Business Model

© Copyright IBM Corporation 2016

This section describes an overview of Programming Model and Business Model.

Organization Structures

- Provides a framework for the actors, or entities, in a business scenario
- Organized in a hierarchical structure, similar to typical organizational hierarchies with entries for 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 bus
- Assign all actors a position
- Allows users to administer site, update catalog, creating promotions
- Organizational units in the framework act as owners
- MemberRegistrationAttributes.xml file is used to assign roles



© Copyright IBM Corporation 2016

IBM WebSphere Commerce Version 8.0 is a powerful customer interaction platform for online and cross-channel commerce, built for a wide range of environments. Recognized as an industry-leading e-commerce solution, IBM WebSphere Commerce supports all the business models of a company, while providing a rich, differentiated customer experience through a single platform.

With powerful ready-to-use capabilities and easy-to-use business user tools, IBM WebSphere Commerce delivers a proven, flexible solution that scales to meet your business requirements no matter your industry, size of company, or selling model. State-of-the-art tools help your staff create and manage precision marketing campaigns and promotions and efficiently manage catalogs, products, merchandising, and connectivity to external systems.

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.

Assign all actors in your business scenario a position in your WebSphere Commerce organization structure.

The MemberRegistrationAttributes.xml file is used to dynamically assign WebSphere Commerce roles to authenticated users. This dynamic assignment occurs during logon, registration, and single sign-on to WebSphere Commerce.

What are Extended Sites?

- Extended Sites can be defined in two ways:
- Business Definition:
 - A business model offered by WebSphere Commerce which allows a seller to have multiple sites aimed at different audiences
- Technical Definition:
 - A comprehensive inheritance framework for WebSphere Commerce websites
- Assets can be shared/inherited
- Catalog Assets: categories, products, items
- Storefront Assets: JSPs, properties files, images
- Server Assets: Controller commands
- Marketing Assets: Business-user-managed content
- Promotions

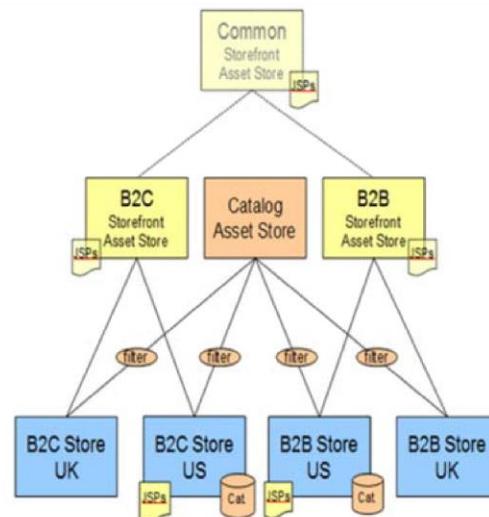
© Copyright IBM Corporation 2016

Extended Sites, one of the predefined business models in WebSphere Commerce, is used when a seller can have many sites that are aimed at different audiences. For example, a seller can create sites that are based on geographical regions, while some customized sites can be created for individual large customers. All these different sites can share assets such as the catalog. Each site can select the subset of the catalog that is presented, and adjust the prices as necessary.

The extended sites business model allows a business, or enterprise to use multiple strategies to make its products available to customers. Many different sites can be presented to the market and each is perceived by customers as unique. The business can customize sites by: Geography, brand, market segment, or customer.

How can it be used?

- Inheritance between stores
 - Asset stores (profile stores)
 - Customer-facing stores
- Inheritance relation is known as the *Store Path*
 - Defined in the STOREREL table
- Multiple inheritance
 - Catalog filtering
 - Adding customer-facing stores without publishing
 - Potential for multiple levels



© Copyright IBM Corporation 2016

Extended site stores can be created as either consumer direct or B2B direct stores. In the Consumer Direct business model, an enterprise sells goods or services to the general public. In variations of this business model, the set of consumers might be restricted. For example, the consumers might be employees of a company. With the B2B Direct business model, an enterprise sells to other businesses, either users of the products or, in some cases, resellers of the products.

There are some common scenarios within these business models in which the selling enterprise needs to create multiple sites. Such scenarios include unique sites for different geographical areas, different brands, different markets, and for large customers

When to use Extended Sites?

- Extended sites should be used whenever two or more sites share some common assets (code, products, etc.)
- The main business scenarios for Extended Sites include:
 - Multiple Brands
 - Multinational Sales
 - Multiple Retail Geographies
 - Multiple Market Segments
 - Enterprise Account
 - Franchising
 - Partner Sites
 - Multiple customers

© Copyright IBM Corporation 2016

Few scenarios:

Sites for different brands :

An extended site can customize its presentation to suit the requirements of each brand it carries in its catalog, such as site look-and-feel, flow, and product display pages. Each brand can have its own unique marketing strategy, such as catalog up-sells, cross-sells, and promotions.

Sites for different market segments:

In the extended sites business model, different market segments like B2B and B2C customers can be addressed by different sites. In addition, unique sites can be created for industry segments such as Education, Travel, Transportation, and Industrial. These sites are based on the same set of catalog data, but each site needs to filter the portions of the catalog that are applicable to it. As with branding, marketing campaigns and messages can be unique to the site, but a set of marketing messages can be common to the extended sites.

Sites for different customers:

Each customer in the extended sites business model can have its own business rules, such as shipping and billing options, and contractual arrangements that affect pricing or product entitlement. A customer can be granted one of the predefined sets of terms and conditions, and additions or exclusions are also applied on top of the predefined contract. In addition, a customer can negotiate special bids with unique prices, or might have entitlements to made-to-order products that are available only to that customer.

B2C Business Model

- Supports commerce transactions involving products, services, or information between businesses and consumers.
- Customers buy directly from the business, usually a retailer, or manufacturer who sells their goods directly to consumers through their own retail outlet.
- As of V8, B2C business model is supported only within the Extended Sites business model, a standalone B2C store is not supported.
- For example
 - A business that sells to consumers directly through a catalog would be considered a B2C business.



© Copyright IBM Corporation 2016

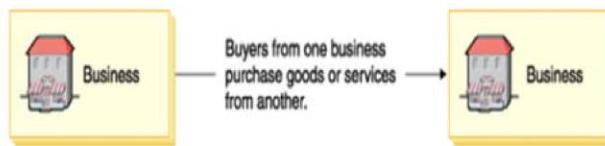
WebSphere Commerce supports commerce transactions involving products, services, or information between businesses and consumers. Consumers typically purchase goods or services directly from a business in this B2C scenario.

In a typical B2C business, customers buy directly from the business, usually a retailer. 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 B2C business.

B2B Direct Business Model

- Supports commerce transactions involving products, services, or information between businesses or parties.
- Direct transactions occur between buyers, suppliers, manufacturers, resellers, distributors, and trading partners.
- As of V8, B2B business model is supported only within the Extended Sites business model, a standalone B2C store is not supported.
- For example
 - Businesses purchase goods or services directly from another business



© Copyright IBM Corporation 2016

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

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, can sometimes implement sites that are based on the B2B direct business model. This occurs in such cases where governments provide goods and services directly to businesses.

Overview of Configuration Files and Command Line Utilities

© Copyright IBM Corporation 2016

This section describes an overview of Configuration files and command line utilities.

Updating the WebSphere Commerce Configuration File

- Runtime applications are configured in an XML file.
- This XML file is inside Enterprise Edition EAR.
- The WebSphere Commerce configuration file is **wc-server.xml**.
 - Location : WC_eardir/xml/config/, workspace_dir\WC\xml\config
- Update the configuration files:
 - The Configuration Manager can modify configuration parameters
 - Use the GUI
 - Edit manually
 - > Edit the file
 - > Deploy the file to the WebSphere Commerce EAR.

© Copyright IBM Corporation 2016

Many aspects of the WebSphere Commerce runtime application are configured in an XML file. This XML file is inside the WebSphere Commerce Java Platform, Enterprise Edition EAR.

The WebSphere Commerce configuration file is `wc-server.xml` located under `WC_eardir/xml/config/` and `workspace_dir\WC\xml\config` directories.

To update the configuration files:

The Configuration Manager can be used to modify most WebSphere Commerce configuration parameters.

However, some parameters must be changed through the WebSphere Commerce configuration file.

To determine whether to configure an individual component through the Configuration Manager or through the WebSphere Commerce configuration file, see the documentation for that component.

Option A: Use the GUI. For instructions on how to use the GUI, see Modifying a WebSphere Commerce instance.

Option B: Edit manually

1. Edit the file. Modify the master copy of the file per whatever instructions you are following. The master configuration file that is managed by the Configuration Manager can be found at the following location:

`WC_installdir/instances/instance_name/xml/instance_name.xml`

`WC_userdir/instances/instance_name/xml/instance_name.xml`

2. Deploy the file to the WebSphere Commerce EAR.

After manually changing the master configuration file, ensure that you properly deploy it to the WebSphere Commerce Java Platform, Enterprise Edition application that is running on WebSphere Application Server.

Updating the XML configuration files by using the configMigration utility

- Use configMigration utility to update XML configuration files that were migrated from version 7.0 to version 8.0.
- Utility upgrades configuration files that preserve customizations.
- configMigration utility applies all of the version 8.0 XML configuration file changes to your version 7.0
- On WebSphere Commerce Version 8.0 environment, back up important configuration files
- For more details refer to the Knowledge Centre :

http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.migrate.doc/tasks/tmg_upconfig_prod.htm?lang=en

© Copyright IBM Corporation 2016

Use the configMigration utility to update XML configuration files that were migrated from version 7.0 to version 8.0. The utility upgrades the configuration files while it preserves your customizations.

In a previous migration step, you deployed your customizations from your version 8.0 development environment to your runtime environment by using WCBD. That deployment contained the various XML configuration files that were modified specifically for your customizations. In their current state, those XML files are at the WebSphere Commerce Version 7.0 level. Those XML configuration files must be updated to WebSphere Commerce Version 8.0. The configMigration utility applies all of the version 8.0 XML configuration file changes to your version 7.0 XML configuration files, leaving your customizations intact.

The configMigration utility updates the configuration files.

Before you begin, On your WebSphere Commerce Version 8.0 environment, back up the important configuration files:

For more details refer to the Knowledge Centre :

http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.migrate.doc/tasks/tmg_upconfig_prod.htm?lang=en

Procedure:

1.Go to the WC_installdir/bin directory. 2.Run the following command:

Windows : config_ant.bat -buildfile WC_installdir/migration/components/common/xmlUpdate /configMigrationTasks.xml -

DinstanceName=instanceName

AIX/Linux:config_ant.sh -buildfile WC_installdir/migration/components/common/xmlUpdate /configMigrationTasks.xml -

DinstanceName=instanceName

For IBM i OS operating system:config_ant.sh - buildfile

WC_installdir/migration/components/common/xmlUpdate

/configMigrationTasks.xml -

DinstanceName=instanceName

Where:instanceName=The name of your WebSphere Commerce instance, for example, demo.

3. Review the following log file after the utility completes:

WC_installdir\logs\WCIM\configMigration.MM.dd.YYYY. mm.ss.log

Note:

configMigration utility tool is provided in WebSphere Commerce version 8.0.1 onwards.

Additional Migration utilities

- Management Center Migration Utility
 - A command-line based tool that helps migrate OpenLaszlo-based customizations from WC 7 FEP 2 and onwards to V8 based Management Center code
 - Custom OpenLaszlo source code and definitions files -> Version 8 CMC definitions
 - Struts configurations are ported to Spring configurations
 - Images, resource bundles, configuration files are ported
 - `migrateLOBTools.bat version7LOBToolsdirectory featurePackNumber [Edition]`
 - Eg. migrateLOBTools c:\IBM\WCDE_ENT70\workspace\LOBTools 8 pro
- Staging Environment Migration
 - V7 staging environments can be migrated to V8 with minimal downtime (Includes live production)
 - A documented sequence of migrations steps that can be configured for staging or authoring instances
 - Refer to the knowledge Center topic: Migrating runtime environments (with staging servers)
 - http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.migrate.doc/tasks/tmg_server_stage.htm?lang=en

© Copyright IBM Corporation 2016

WebSphere Commerce Loading Utilities

- Multiple utilities for preparing and loading data into a WebSphere Commerce database
- Data Load utility : Main loading utility for administrators
- Business users can load data into WebSphere Commerce by using the upload feature in Management Center
- Data Extract utility : Extract data from the WebSphere Commerce database into an output file
- Mass load utilities (deprecated)
- A full list of all Commerce utilities can be found here:
 - http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.admin.doc/refs/rovutilities.htm

© Copyright IBM Corporation 2016

WebSphere Commerce provides utilities for preparing and loading data into a WebSphere Commerce database. These utilities provide various functions that can be strung together in the required sequence to solve your particular commerce data management problems. The loading utilities are flexible enough to handle customizations that are made to the WebSphere Commerce schema.

The utilities use 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.

The Data Load utility is a business object-based loading utility that offers the following advantages:

The Data Load utility is easier to use. You do not need to understand the details of your WebSphere Commerce database. You must understand only the business object model, and how to map your data to the business object model.

The Data Load utility transaction boundary is based on business objects as opposed to being based on table objects. The Data Load utility is scalable and has better performance than the mass load utility.

The Data Load utility has a much improved summary report, and provides a better error report to assist with diagnosing problems when errors occur

Data Load utility reference:

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.data.doc/concepts/cmlobatchoverview.htm

The Data Extract utility is a command-line utility that you can use to extract data from the WebSphere Commerce database into an output file.

Reference:

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.data.doc/concepts/cmldataextractutilityover.htm

Overview of WebSphere Commerce Access Control

© Copyright IBM Corporation 2016

This section describes an overview of WebSphere Commerce Access control.

References:

Defining access control policy elements using XML

[http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0
/com.ibm.commerce.admin.doc/tasks/taxdefineacpelementxml.htm](http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.admin.doc/tasks/taxdefineacpelementxml.htm)

Elements of Access Control

- Users:
 - People that use the system.
 - Grouped into relevant access groups (UserGroup).
 - Roles are used to determine membership in an access group.
 - Roles are assigned to users on a per organization basis.
- Actions:
 - Activities that Users can perform on a Resource.
 - Actions can be grouped into relevant groups (ActionGroup).
- Resources:
 - Entities, such as JSPs and commands, that are protected
 - Can be grouped into relevant groups (ResourceGroup).
- Relationships:
 - Define connection between Users and Resources (such as Owner, Editor, Reader).

© Copyright IBM Corporation 2016

Access control in a WebSphere Commerce application 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. 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 that is used in a store is a view. A view is invoked to display a store page to customers. The views that are 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, **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 might require that a relationship between the user and the resource are satisfied. For example, users might only be allowed to display the orders that they create.

Types of access control

- Two types of authentication:
 - WebSphere Application Server (broad)
 - WebSphere Commerce Server (fine)
- Two types of access control:
 - Command-level (broad)
 - Also known as role-based.
 - Specifies that users who are assigned to a particular role can execute certain commands.
 - For example, guest users can execute the commands that are contained in action group X.
 - Resource-level (fine)
 - Specifies the relationship that a user must have with a resource before an action can be performed.
 - For example, users can display only an order they created.

© Copyright IBM Corporation 2016

In a WebSphere Commerce application, there are two main levels of authorization and authentication. WebSphere Application Server performs the first level of access control. In this respect, WebSphere Commerce uses WebSphere Application Server to protect enterprise beans and servlets. The second level of access control is the fine-grained access control system of WebSphere Commerce.

The WebSphere Commerce access control framework uses access control policies to determine whether a user is permitted to act on a resource. This access control framework provides fine-grained access control. It works with, but does not replace, the access control that the WebSphere Application Server provides.

There are two types of access control, both of which are policy-based:
command-level access control and **resource-level access control**.

Command-level (also known as role-based) access control uses a broad type of policy. You can specify that all users of a particular role can execute certain types of commands. For example, you can specify that users with the Account Representative role can execute any command in the AccountRepresentativesCmdResourceGroup resource group.

Another example policy is to specify that all seller administrators can perform an action that is specified in the SellerAdministratorsViews group on any resource that the ViewCommandResourceGroup specifies.

A command-level access control policy always has the ExecuteCommandActionGroup as the action group for controller commands. For views, the resource group is always ViewCommandResourceGroup.

Command-level access control protects all controller commands. In addition, command-level access control must protect any view that can be called directly, or that a redirect from another command (in contrast to being launched by forwarding to the view) can launch.

Command-level access control determines whether the user is allowed to execute the particular command within the store you specify. If a policy allows the user to execute the command, a subsequent resource-level access control policy could be applied to determine whether the user can access the resource in question.

Consider the case when a seller administrator attempts to perform an administrative task. The first level of access control checking determines whether this user is allowed to execute the particular seller administration command within the current store. The user is allowed to run the command if the command belongs to the ViewCommand resource group, and the user has the seller administrator role in the current store's organization. In such cases, the command might then trigger a resource-level access control check if needed. In order to satisfy this check, a resource-level policy would have to grant access. Such a policy might state that seller administrators are only permitted to perform administrative tasks on resources that the organization for which the user is a seller administrator owns.

Reference : http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/caxunderstandaccessmain.htm

Access Control Policies

- Over 300 policies included OOB.
- Access control policies authorize access groups to perform particular actions on particular resources.
- All access control policies that are provided with WebSphere Commerce are assigned to 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. The policy group must be subscribed to by the organization that owns the resource.
- Unless authorized through one or more access control policies, users have no access to any functions of the system

© Copyright IBM Corporation 2016

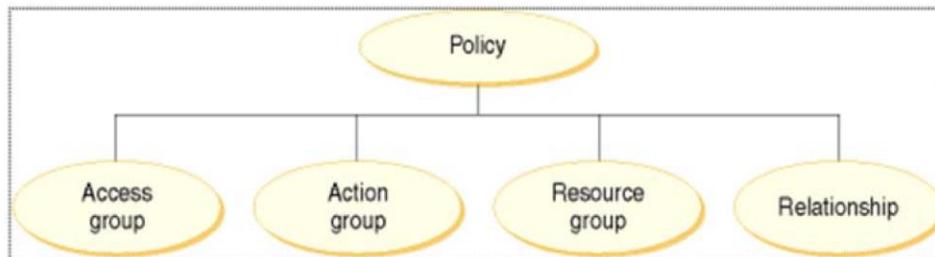
Although organizations own access control policy groups, they are not automatically applied to the organization. An organization must subscribe to a policy group in order for the access control policies to apply to the organization. If the organization has child organizations, all policy groups the parent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

In previous versions of WebSphere Commerce, a policy that is 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. Organizations can subscribe to policy groups.

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 in the command context. If there is no store in the command context, then Root Organization is used as the owner.

Access Control Policies and Entities

- Access group (ACRELGRP, MBRGRP, ROLE) – Users to which the policy applies.
 - That is, RegisteredCustomers, StoreAdministrators, Customer Service Representatives, AccountRep > BuyerOrganizationalEntity, ContractAdministrator > Participant
- Action group (ACACTGRP) – Group of actions that the users perform on resources.
 - That is, Approve, AllUserViews, DoEverything
- Resource group (ACRESGRP) – Resources that the policy controls.
 - That is, CatalogResourceGroup, StoreAdministratorsCmdResourceGroup
- Relationship (ACRELATION) (optional) – A set of relationships that are associated to the resource.
 - That is, Creator, owner, approver, NotDescendant



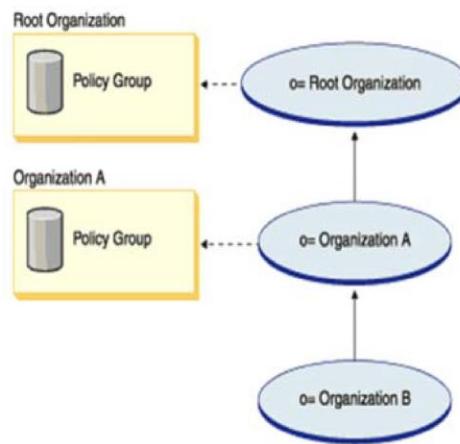
© Copyright IBM Corporation 2016

An access control policy authorizes a group of users to perform a set of actions on a set of resources within WebSphere Commerce. Unless authorized through one or more access control policies, users have no access to any functions of the system.

The policies are what grant users access to your site. Unless they are authorized to perform their responsibilities through one or more access control policies, users have no access to site functions.

Access control inheritance from direct parent

- Organizations can subscribe to policy groups.
- Organization A policy group is applied to Organization A.
- Organization B did not implicitly apply any policy groups.
- Access control framework searches up the organization tree.
- Organization A is the parent to Organization B.
- Organization A policy group is applied to Organization B.

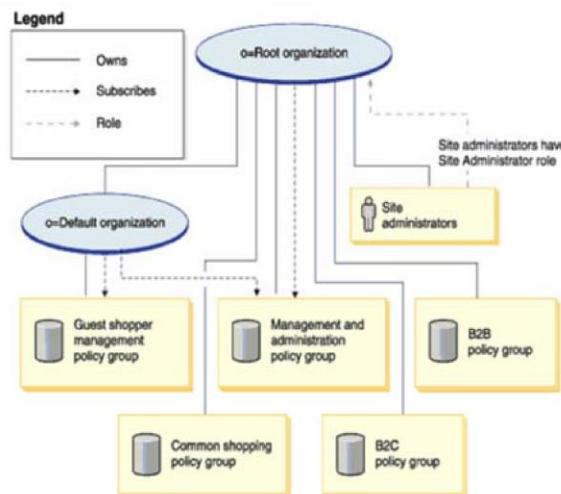


© Copyright IBM Corporation 2016

Organization B does not subscribe to any policy groups; the access control framework begins searching 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 Organizations A and B.

Basic access policy structure

- The root organization owns the following default policy groups:
 - Management and Administration
 - Common Shopping
 - B2C
 - B2B
- The default organization owns the guest shopper management policy group.



© Copyright IBM Corporation 2016

The root organization owns the following default policy groups:

- Management and Administration
- Common Shopping
- B2C
- B2B

However, the root organization subscribes only 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.

Default access control policies

- The default policies that are shipped with WebSphere Commerce are organized into the following categories:
 - Role-based policies: The role-based policies for each default role. These policies (command-level policies) define who can execute each command.
 - Resource-level policies: The resource-level policies, which are grouped by business area. These policies define the actions that a group of users can perform on specific resources. Each business area is organized by the type of resource they regulate:
 - Data resources:
Business objects that can be manipulated such as an order or a bid.
 - Data Bean resources:
Contain information about business objects. Data beans display object information about a web page.

© Copyright IBM Corporation 2016

The default policies that are shipped with WebSphere Commerce are organized into the following categories:

Role-based policies: The role-based policies for each default role. These policies (command-level policies) define who can execute each command.

Resource-level policies: The resource-level policies, which are grouped by business area. These policies define the actions that a group of users can perform on specific resources. Each business area is organized by the type of resource they regulate:

Data resources:

Business objects that can be manipulated such as an order or a bid.

Data Bean resources:

Contain information about business objects. Databeans display object information about a web page.

Access control policy utilities

- Access control records created in the database by utilities in WebSphere Commerce or directly through SQL:
 - `acugload` (load access groups).
 - `acupload` (load policy groups).
 - `acpnlsload` (load NLS descriptions).
 - `acextract` (extract policies and relationships).
- Policies are defined in a specific XML format:
 - See examples in <WCDE_HOME>\xml\policies\xml
 - Create XML files that conform to DTDs in \xml\policies\dtd
- Can load through the SAR publish process:
 - Edit `accesscontrol.xml` in \WEB-INF\stores\StoreAssetsDir\data
 - Descriptions using `accesscontrol_</language>.xml`
- View policies using Organization Administration Console

© Copyright IBM Corporation 2016

The access control policy loading process reads the XML files that contain the access control policy information and access group definitions, and loads them into the appropriate databases. The policy and access group information that is in the XML files is loaded at installation; however, you must reload the files if they are changed.

Database user, who has read, write, and execute permissions updates the access control.

The SiteAdministratorsCanDoEverything policy is a special default policy that grants super-user access to administrators with the site administrator role. In this policy, a site administrator can act on any resource, even if those actions or resources are not defined. It is important to be aware of this fact when assigning this role to users.

BecomeUserCustomerServiceGroupExecutesBecomeUserCmdsResourceGroup policy is a special policy that allows certain administrative users to run specified commands on behalf of other users. This policy is needed, for example, when a customer requests a customer service representative to create an order on the customer's behalf. In this case, the customer service representative is able to run the command such that it appears as if the customer himself runs the command.

An important task for administrators is to keep multiple machines in sync; access control policies are no different. The acextract utility is crucial for extracting policy data for subsequent load onto another machine. There are two levels of authorization check:

- WebSphere Application Server
- WebSphere Commerce (fine-grained)

There are two types of policy-based control in WebSphere Commerce:

- Command-level (role-based, broad)
- Resource-level (resource-based, fine)

Controller commands require command-level access policies.

Defining access control policy elements using XML

- Organization Administration Console allows to make simple changes to access control policies and their parts
- Edit XML files and load them into database
- Create customized commands, entity beans, and JSP templates that can be protected by access control policies
- Finish code customization
- Edit the XML files for access control to establish the protections required
- Manipulate the XML files to perform the authorization tasks

© Copyright IBM Corporation 2016

The Organization Administration Console allows you to make simple changes to access control policies and their parts. To make more sophisticated changes, you need to edit the XML files directly, and then load them into the database.

Before you begin making changes to the XML files for access control, refer the topics WebSphere Commerce Developer Access Control and Creating an access control policy in knowledge center. These topics provide a technical overview of access control and explains how to create customized commands, entity beans, and JSP templates that can be protected by access control policies.

Once you have finished the code customization, you can edit the XML files for access control to establish the protections required.

The following changes can only be made by editing and then loading the appropriate XML files:

Creating or modifying an action
Creating or modifying a relationship

Creating or modifying a relationship group
Creating or modifying a resource
Creating or modifying attributes

Creating or modifying access groups using complex criteria
Creating or modifying resource groups using complex criteria
Creating a role-based policy for views

Changing the action group in a role-based policy for views
Creating or modifying a policy group

Associating policies with policy groups

You can manipulate the XML files to perform the following authorization tasks:

Protecting views

Protecting controller commands Protecting resources

Protecting data beans

Grouping resources by attributes Defining
relationships

Defining relationship groups Defining
access groups Defining policies

Reference for more details here: http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.admin.doc/tasks/taxdefineacpelementxml.htm

Access control, example policy

- Common change, adding a view to an action group procedure:
 - Create an action definition in the XML file that has the view name:
- <Action Name=" MyNewView CommandName=" MyNewView"> </Action>
 - Create an action group to be associated with the new role:
- <ActionGroupName=" XYZViews" OwnerID="RootOrganization"> </ActionGroup>
 - Associate the new action with the new action group:
- <ActionGroupAction Name=" MyNewView"/>
- Example:

```
<Policies>
  <Action Name="SpecialOrderView"
    CommandName="SpecialOrderView">
    </Action>
  <ActionGroup Name="AllSiteUsersViews"
    OwnerID="RootOrganization">
    <ActionGroupAction Name="SpecialOrderView"/>
  </ActionGroup>
</Policies>
```

© Copyright IBM Corporation 2016

Common change, adding a view to an action group procedure: Create an action definition in the XML file that has the view name: <Action Name=" MyNewView CommandName=" MyNewView"> </Action>

Create an action group to be associated with the new role:

```
<ActionGroupName=" XYZViews" OwnerID="RootOrganization">
</ActionGroup>
```

Associate the new action with the new action group: <ActionGroupAction Name=" MyNewView"/>

Loading access control policy

- Create or copy access control policy file into the directory:
 - <WC_HOME>\xml\policies\xml
- Run the access control policy load program from:
 - <WC_HOME>\bin
 - Must be logged in as administrator.
 - Example:
 - Windows:
`acupload.cmd database_name database_user database_user_password
policies_xml_file schema_name`
 - > database_name Required: Name of the database in which to load the policy.
 - > database_user Required: Name of the database user who can connect to the database.
 - > database_user_password Required: The associated password for the database user.
 - > policies_xml_file Required: The input policy XML file that specifies what policy data to load into the database.
 - > schema_name Optional: The name of target database schema. This name is normally the same as database_user.
 - Check for errors in log file.
 - messages.txt
 - acupload.log

© Copyright IBM Corporation 2016

Loading an access control policy is simple. The steps include:

1. Create or copy the access control policy file that you created into the WebSphere Commerce home directory\xml\policies\xml directory.

2. Run the acupload script. This script loads the following elements:

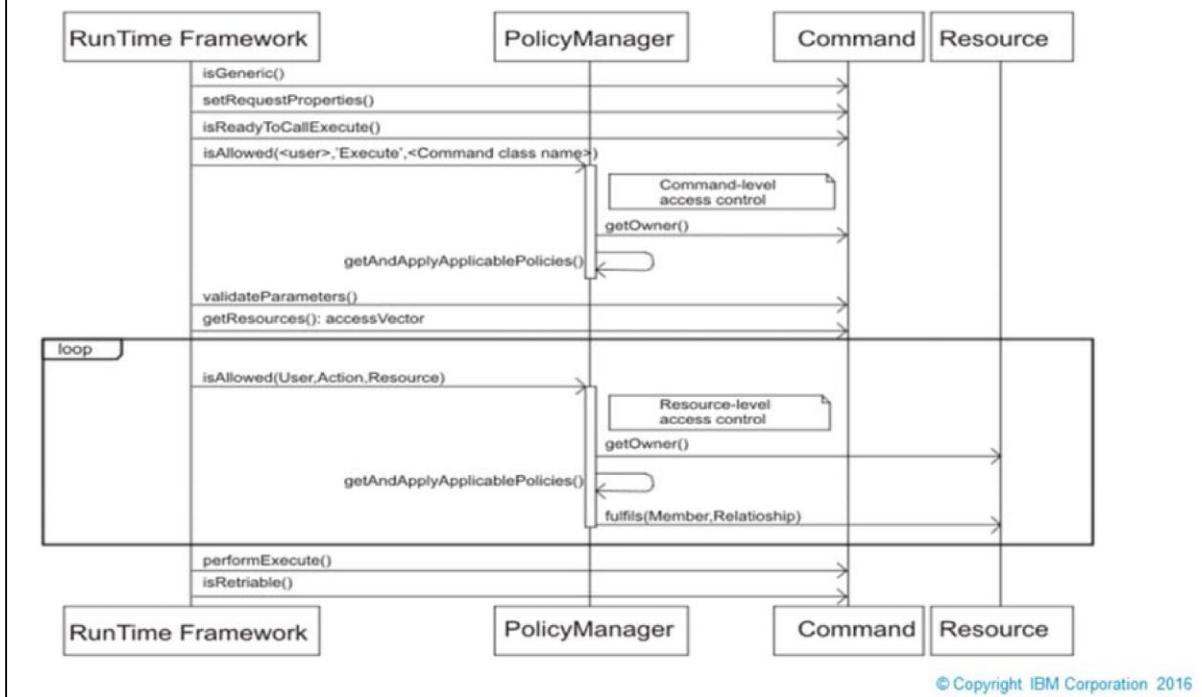
<Action>, <ActionGroup>, <Attribute>, <ResourceCategory>,
<ResourceGroup>, <Relation>, <RelationGroup>, <Policy>,
<PolicyGroup>.

3. Check the error log file, messages.txt and acupload.log, for any errors.

The loading process reads the XML files that contain the access control policy information and access group definitions and loads them into the appropriate databases.

Access control policies, entities

- The access control policy framework controls how access works in WebSphere Commerce.



© Copyright IBM Corporation 2016

The access control policy framework controls how access control works in WebSphere Commerce.

This diagram shows actions that the access control *policy manager* does. The access control policy manager is the access control component that determines whether the current user is allowed to perform the specified action on the specified resource. It determines access by searching through the policies in groups to which the resource owner subscribes. If the resource owner does not subscribe to any policy groups, it searches through the policies in groups to which the resource owner's closest ancestor subscribes. If at least one policy grants access, then permission is granted.

- isGeneric()** Determines whether a generic user needs to be converted to a guest user.
- setRequestProperties()** passes the request parameters to the command. The request parameters can be used in subsequent access control calls (for example, by returning requestparameter-specific data from getOwner() or getResources()).
- isAllowed()** The runtime components determine whether the user has command-level access for either the controller command or view.
- getOwner()** The access control policy manager determines the owner of the command-level resource.
- getAndApplyApplicablePolicies()** The access control policy manager finds and processes the applicable policies based on the specified user, action, resource, and current store.
- validateParameters()** Checks and resolves the initial parameters.
- getResources()** Returns an access vector that is a vector of resource- action pairs.
- isAllowed()** The runtime components determine whether the user has resource level access to all of the resource-action pairs that the getResources() specify.
- getOwner()** The resource returns the memberID of its owner. The memberID determines which policies apply.
- getAndApplyApplicablePolicies()** The access control policy manager searches for applicable policies and then applies them.

- fulfils()** If an applicable policy has a relationship or relationship group specified, a check is done on the resource to see whether the member satisfies the specified relationship, with regards to the resource.
- performExecute()** The business logic of the command.
- isRetriable()** Checks whether the command can be tried again if the performExecute() call shows an ECSysException.

Access control in controller commands

- Access control is enabled by extending the base ControllerCommand and ControllerCommandImpl classes that implement the AccCommand interface, which in turn extends Protectable.
- A simple implementation of the getResources() method follows:

```
public AccessVector getResources() throws ECException {
    String methodName = "getResources";
    ECTrace.entry(25L, getClass().getName(), "getResources");
    ECTrace.exit(25L, getClass().getName(), "getResources");
    return null;
}
```

© Copyright IBM Corporation 2016

Access control is enabled by extending the base ControllerCommand and ControllerCommandImpl classes that implement the AccCommand interface, which in turn extends Protectable.

A simple implementation of the getResources() method follows:

```
public AccessVector getResources() throws ECException { String
    methodName = "getResources";
    ECTrace.entry(25L, getClass().getName(), "getResources");
    ECTrace.exit(25L, getClass().getName(), "getResources"); return
    null;
}
```

Unit summary

This unit was designed to enable you to:

- Understand the WebSphere Commerce Database schema
- Understand the WebSphere Commerce Development Database
- Understand WebSphere Subsystem Data models
- Describe WebSphere Commerce Organization Structures
- Understand WebSphere Commerce configuration files and command line utilities
- Understand Access control and policies

© Copyright IBM Corporation 2016

Reference

- **WebSphere Commerce database schema**
 - http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.database.doc/concepts/cdb_schema_overview.htm
- **Data models**
 - http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.database.doc/refs/rdb_datamodel_index.htm
- **Cross-reference of data beans, EJB beans, and tables**
 - http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rgxejb.htm
- **Cross-reference of commands, tasks, and tables**
 - http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rgxxref.htm
- **Cross reference: Commands to beans to database tables**
 - http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rgxref.htm

© Copyright IBM Corporation 2016

Additional training

For additional training needs, you have several options available for registering.

You can register	At
On site	This or any of our training facilities
Online	http://ibm.com/training/
Email	ibmswedu@us.ibm.com
By telephone	1-800-IBM Teach (1-800-426-8322)
By fax	208-692-6130

© Copyright IBM Corporation 2016

For additional training needs, you have several options available for registering like Onsite, Online , Email , by telephone or by Fax .

Unit 3

IBM Training



WebSphere Commerce V8 Developer Toolkit

© Copyright IBM Corporation 2016
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

This presentation provides an overview of the WebSphere Commerce Version 8 development Toolkit.

Unit objectives

- Overview of WebSphere Commerce development environment
- WC Tools
- Understand the co-existence of V7 and V8 toolkits
- Understand the process to install and configure developer toolkit on single or multiple workstations
- Publishing a given store archive
- Workspaces Overview
- Understand logging, tracing, and troubleshooting
- Demonstration of exploring the toolkit environment

© Copyright IBM Corporation 2016

At the end of this presentation, you should understand the WebSphere Commerce development environment. This is followed by an understanding of the co-existence of Version 7 and Version 8 toolkits.

You should also understand the process to install and configure the developer toolkit environment on single or multiple workstations. You will learn how to publish a given store archive. Then, understand about logging, tracing, and troubleshooting in WebSphere Commerce. You will receive an overview of the WebSphere Commerce developer tools.

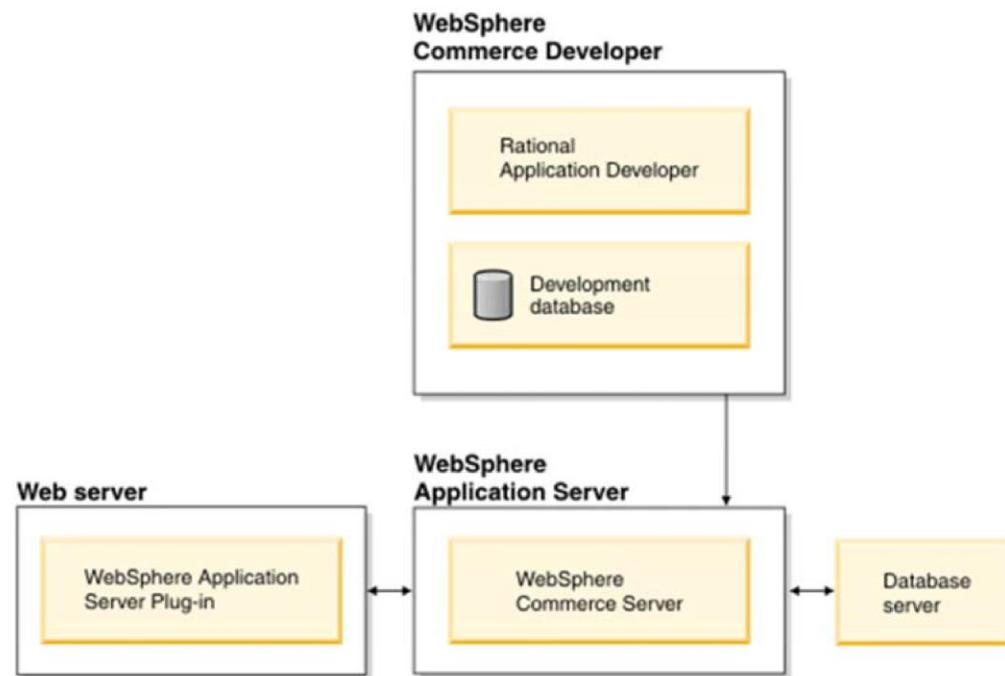
Lastly, you will be presented with a demonstration that explores the toolkit environment.

Overview of WebSphere Commerce development environment

© Copyright IBM Corporation 2016

This section describes an overview of the WebSphere Commerce Version 8 development environment.

WebSphere Commerce common architecture



© Copyright IBM Corporation 2016

The Web server is the first point of contact for incoming HTTP requests for your e-commerce application. In order to interface efficiently with the WebSphere Application Server, it uses the WebSphere Application Server plug-in to manage the connections between the two components.

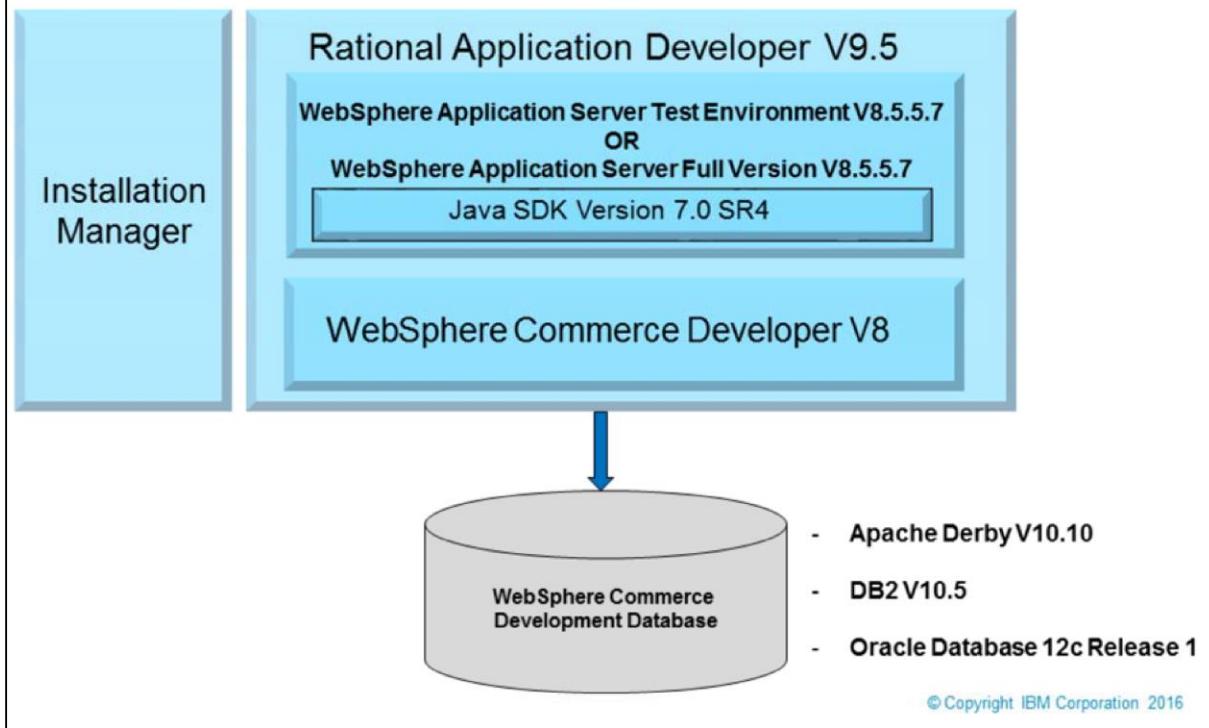
The WebSphere Commerce Server runs within the WebSphere Application Server, allowing it to take advantage of many of the features of the application server. The database server holds most of your application's data, including product and customer data. In general, extensions to your application are made by modifying or extending the code for the WebSphere Commerce Server. In addition, you may have a need to store data that falls outside of the realm of the WebSphere Commerce database schema within your database.

Developers use Rational Application Developer to perform the following tasks:

- Create and customize storefront assets such as JSP and HTML pages
- Create and modify business logic in Java
- Create and modify access beans and EJB entity beans
- Test code and storefront assets
- Create and modify Web services

The WebSphere Commerce development environment uses a development database. Developers can use their preferred database tools (including Rational Application Developer) to make database modifications. WebSphere Commerce supports a one to one mapping between the WebSphere Commerce instance and the WebSphere Commerce database. Running multiple WebSphere Commerce instances against the same database is not supported.

WebSphere Commerce development environment



Development environment consists of:

- Rational Application Developer or Rational Software Architect Version 9.5 or higher
- A 64-bit WebSphere Application Server. You can install either one of the following
 - WebSphere Application Server Test Environment V8.5.5.7 or higher
 - WebSphere Application Server Full Version V8.5.5.7 or higher

If you are installing the WebSphere Application Server full version (WebSphere Application Server Version 8.5.5), then you need to update to WebSphere Application Server Fix Pack 7.

Refer to the Knowledge Center for the current suggested versions for all required software to get more information.

WebSphere Commerce development environment

- 64-bit Rational Application Developer V9.5
 - Eclipse-based development environment
 - Rational Software Architect V9.5 can be installed instead of Rational Application Developer
- WebSphere Commerce Developer V8
 - Leverages tools from Rational Application Developer
 - Leverages WebSphere Application Server V8 test or full server environment
- Development database
 - Stores development artifacts
 - Can be DB2, Apache Derby (default) or Oracle
- Hardware requirements
 - 64-bit – All supported Windows operating systems
 - Disk space – 50 GB
 - Memory – 4 GB

© Copyright IBM Corporation 2016

64-bit Rational Application Developer V9.5 is an Eclipse-based development environment. You have the option to install Rational Software Architect V9.5 instead of Rational Application Developer.

WebSphere Commerce Developer V8 Leverages tools from Rational Application Developer. Also, Leverages WebSphere Application Server V8 test or full server environment.

WebSphere Commerce Development database is used to store development artifacts. Database can be IBM DB2, ApacheDerby which is installed by default or Oracle.

WebSphere Commerce requires 64-bit machine which is supported on all Windows operating systems. It requires minimum disk space of 50 GB and memory of 4 GB.

WebSphere Commerce Workspace (1/2)

- Installing WebSphere Commerce V8 Developer provides a fully configured Rational Application Developer workspace.
- The following projects are included:
 - **CommerceAccelerator**: contains assets for Commerce Accelerator.
 - **LOBTools**: customizable assets for IBM Management Center for WebSphere Commerce.
 - **OrganizationAdministration**: assets for the Organization Administration Console.
 - **SiteAdministration**: assets for the Administration Console.
 - **Stores**: module for sample store assets.
 - **WebSphereCommerceServerExtensionsData**: create custom enterprise beans and other data assets.
 - **WebSphereCommerceServerExtensionsLogic**: create new logic, such as Java classes.
 - **WebServicesRouter**: create custom web service assets.

© Copyright IBM Corporation 2016

WebSphereCommerceServerExtensionsData, -ExtensionsLogic, and WebServicesRouter are initially empty. All custom code is recommended to be placed in these projects (unless those customizations are for one of the four consoles). These folders protect your custom code when fix packs and feature packs are applied.

It is not an accepted best practice to modify the WC project.

To work on several development projects at a time, multiple workspaces can be created by installing WebSphere Commerce Developer multiple times.

WebSphere Commerce Workspace (2/2)

- **WC**: contains the core WebSphere Commerce EAR file.
- **Rest** : Customize REST services by creating or modifying assets in this module.
- **Search** : The enterprise archive (EAR) for the WebSphere Commerce search application.
- **Search-Rest** : Customize WebSphere Commerce search-based REST services by creating or modifying assets in this module.

© Copyright IBM Corporation 2016

WebSphere Commerce development database

- Apache Derby (default)
- Can use DB2 or Oracle
- Accessing DB
 - While WCS server is running
 - Use <http://localhost/webapp/wcs/admin/servlet/db.jsp>

CATEENTRY_ID	MEMBER_ID	ITEMSPC_ID	CATENTTYPE_ID	PARTNUMBER	MPPARTNUMBER	MNAME	MARKFORDELETE	URL	FIELD1	FIELD2
0	0	NULL	'ProductBean'	'0'	NULL	NULL	0	NULL	NULL	NULL
10001	7000000000000000000002	NULL	'ProductBean'	'AuroraWMDRS-1'	NULL	'Hermitage Collection'	0	NULL	NULL	NULL
10002	7000000000000000000002	NULL	'ProductBean'	'AuroraWMDRS-2'	NULL	'Versatil'	0	NULL	NULL	NULL
10003	7000000000000000000002	NULL	'ProductBean'	'AuroraWMDRS-3'	NULL	'Luisi Valente'	0	NULL	NULL	NULL

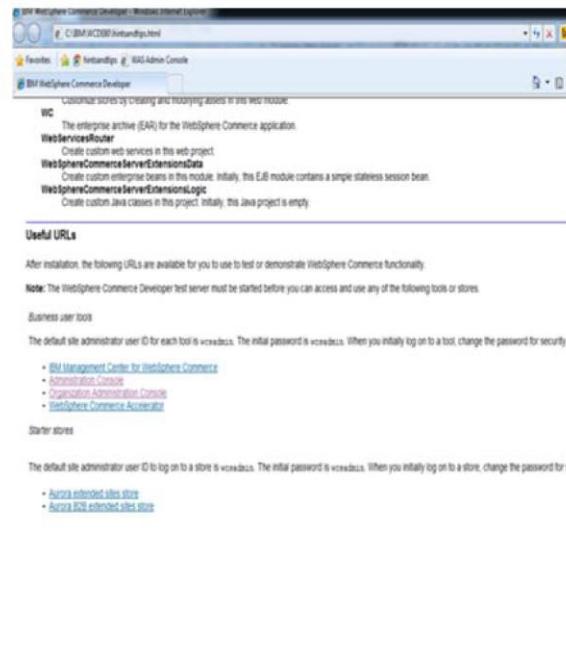
- While WCS server is not running
 - Use command prompt
 - Can use other clients like SQL squirrel etc.

© Copyright IBM Corporation 2016

Talk about the in built DB access and access via clients – easy way.

WebSphere Commerce tools

- Open the bookmarked tools URL using
C:\IBMWCD80\hintsandtips.html
- Tools Available
 - WebSphere Commerce Administration Console
 - WebSphere Commerce Organization Administration Console
 - WebSphere Commerce Accelerator
 - IBM Management Center for WebSphere Commerce

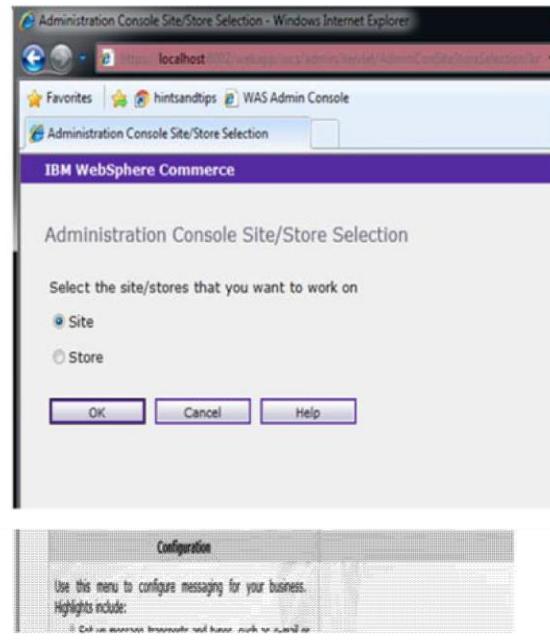


© Copyright IBM Corporation 2016

Give overview of each tool. Some are explained in detail in next slides

WebSphere Commerce Administration Console

- Site Administrators use the Administration Console to control a site or store by completing administrative operations and configuration tasks such as publishing stores, refreshing the registry, and scheduling jobs.
- Maintain security settings for your site.
- Track messages and transactions for your business.
- Configure messaging for your business.
- Manage access control policies for the site.



Site Administrators use the Administration Console to control a site or store by completing administrative operations and configuration tasks such as publishing stores, refreshing the registry, and scheduling jobs.

Maintain security settings for your site

- Set up account-related, password, and account lockout policies

Track messages and transactions for your business

- View unsent messages
- View archived messages
- Run business auditing reports (store-level)

Configure messaging for your business

- Set up message transports and types, such as e-mail or plain text
- Schedule common jobs that the system should run at particular intervals
- Update registry components
- Enable or disable WebSphere Commerce components for the current session
- Configure e-mail activity templates
- View the product information for the version of WebSphere Commerce you are using

Deploy stores to the site

- Publish a store archive file to deploy stores to the site

Manage access control policies for the site

- Set up access control policies to define which users can perform which tasks.

WebSphere Commerce Administration Console

Site

Security	Monitoring
Use this menu to maintain security settings for your site. Highlights include: ▪ Set up account-related, password, and account lockout policies	Use this menu to track messages and transactions for your business. Highlights include: ▪ View unsent messages ▪ View archived messages

Configuration	Store Archives
Use this menu to configure messaging for your business. Highlights include: ▪ Set up message transports and types, such as e-mail or plain text ▪ Schedule common jobs that the system should run at particular intervals ▪ Update message components	Use this menu to publish a store for your site. Highlights include: ▪ Publish a store archive file to deploy a store to the site ▪ Check the status of a publishing job.

Store

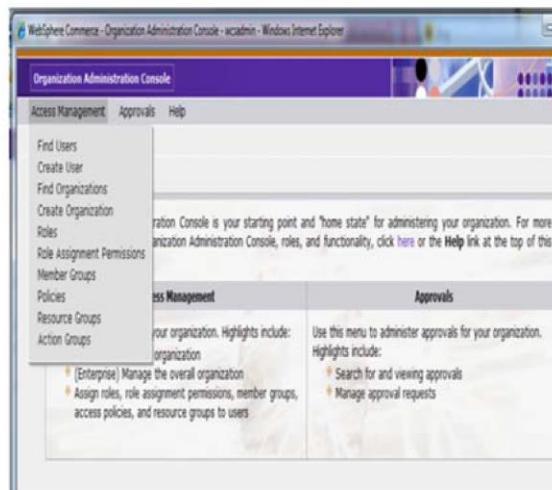
Access Management	Monitoring
Use this menu to manage access control for your store. Highlights include: ▪ Set up access control policies to define which users can perform which tasks	Use this menu to track messages and transactions for your business. Highlights include: ▪ View unsent messages ▪ View archived messages ▪ Run business auditing reports

Configuration	
Use the menu to configure messaging for your business. Highlights include: ▪ Click on external hyperlinks and hover over links to learn more	

© Copyright IBM Corporation 2016

Websphere Commerce Organization Admin Console

- Enables administrators to manage organizations, business users and member groups.
- Work with organizations.
- Work with users.
- Work with roles.
- Create and manage member groups.
- Create and manage access control policies, resource groups and action groups.
- Work with approvals.



Work with organizations.

An organization is a type of member in WebSphere Commerce. Organizations are setup in a hierarchical manner and subscribe to access control policy groups that determine the authorization policies that apply to the assets owned by that organization. Organizations can be created and modified. Organizations can be configured to have different types of approvals. Roles can be assigned to organizations, which determines the roles that users can have for that organization. The access control policy groups that an organization subscribes to can be modified.

A Site Administrator can manage all organizations. A Seller Administrator, Buyer Administrator and Channel Manager can manage organization where they have the role. A Seller Administrator, Customer Service Supervisor and Customer Service Representative can manage (buyer) organizations that have registered to the (seller) organization where they have the role.

Work with users.

A user is any individual that accesses the WebSphere Commerce system. The Organization Administration Console is intended to manage business users. That is, users that are not under the Default Organization and that have a profile type of 'B'. The system is designed so that B2C users exist under the Default Organization and have a profile type of 'C'. In the Organization Administration Console, business users can be created and updated. Roles can be assigned to these users. Users can be assigned to Member Groups. Customers can be assigned to a Customer Service Representative, Customer Service Supervisor or a Seller.

A Site Administrator can manage all users. A Seller Administrator, Buyer Administrator and Channel

Manager can manage users under an organization where they have the role. A Seller Administrator, Customer Service Supervisor and Customer Service Representative can manage users under (buyer) organizations that have registered to the (seller) organization where they have the role.

Work with roles.

Roles are assigned to users to give them authority to perform various operations in the site. New roles can be created in the Organization Administration Console by a Site Administrator. The Site Administrator can also configure which roles can be assigned by users with a particular role. For example, by default the Customer Service Representative role is configured to only be able to assign the Registered Customer role to other users. Roles can be assigned to users and organizations.

Create and manage member groups.

A member group is a grouping of users and organizations -- used for various business purposes. You can create member groups for Access groups, Approval groups, Customer price groups, Customer territory groups, Customer service representative groups, Price override groups, Registered customer groups, or Customer segment groups. Site administrators and Seller Administrators have authority to manage member groups.

Create and manage access control policies, resource groups and action groups.

You can use the Organization Administration Console to make simple changes to access control policies. If extensive changes are required, you should use the acpload utility that uses XML files. Only Site Administrators can manage Access Control Policies.

You cannot use the Organization Administration Console to:

- Define new actions, resources, attributes, relationships, relationship groups.
- Define complex implicit resource groups, and complex implicit access groups.
- Assign a new policy to a policy group.

Work with approvals.

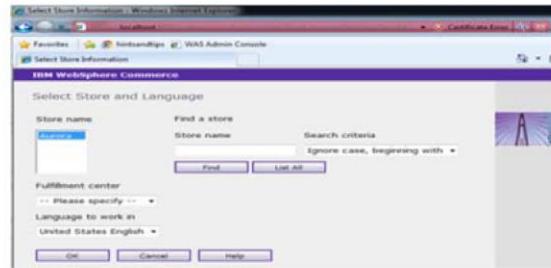
In WebSphere Commerce, there are five processes that might require approval:

- contract submission
- order process
- reseller and buyer registration
- RFQ response
- user registration

A Site Administrator, Seller Administrator, Buyer Administrator, Channel Manager, Customer Service Supervisor and Customer Service Representative can specify the approvals that are needed for an organization. A Site Administrator, Seller Administrator, Buyer Administrator and Channel Manager are able to process approval requests.

WebSphere Commerce Accelerator

- Maintain online stores.
- Select the store, language and Fulfillment center during logon.
- Change the above using Select option after login.
- Managing Store – Shipment, taxes, Inventory, Pricing negotiation etc
- Finding Customer
- Order Management
- Returns and Refunds



Store

Use this menu to administer your store. Highlights include:

- Maintain tax, shipping and fulfillment center settings

Merchandise

Use this menu to manage your inventory. Highlights include:

- Maintain inventory and vendor information

Operations

Use this menu to provide the best customer service. Highlights include:

- Maintain customer registration information
- Manage the end-to-end order process
- Monitor order management, operational, and customer service reports to track store sales

Payments

Use this menu to manage payments for your store. Highlights include:

- Listing all outstanding payment transactions

© Copyright IBM Corporation 2016

Managing Store

The Seller sets up store-level functions within the site. The Seller can access all menu items under the **Store** menu.

- Managing shipment
- Setting up tax
- Inventory adjustment
- Fulfillment
- Negotiating return
- Return reason
- Email notification configuration

Finding Customer

If customers register with the store, you can maintain their registration profiles to track information. Maintain information such as the customer's logon ID, logon password, authentication information, store account status, title, name, preferred language, or currency.

- Finding customer
- Registering Customer
- Customer Summary

Order Management

A typical order includes one or more products, billing and shipping addresses, payment details, and the total cost (including shipping charges and taxes, as applicable). Comments or price adjustments can also be included in an order.

Both registered and non-registered customers can place orders at a store. A *registered customer* provided information to create a profile with the store. The customer has a logon ID and password, which is required to submit orders at the store. Registered customers can also contact the store to inquire about their orders.

A *non-registered customer* is one who does not have a logon ID and password for shopping at the store and is considered a guest.

- Order Processing
- Order and Order item status
- Creating order
- Finding order
- Changing order
- Cancelled order

Returns and Refunds

A *return* is a record of a customer's request for a *refund*, to *return* merchandise, or both. The term is used whether an RMA is issued in advance of merchandise receipt or the transaction is begun at the time of merchandise receipt.

- Finding return
- Creating return
- Changing return
- Return summary

Instructor

notes:

Purpose

— Details

—

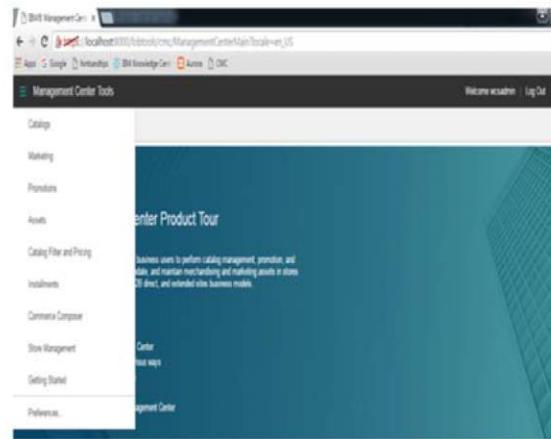
**Additional
information —**

Transition statement

— Next: CMC

Websphere Commerce Management Center

- The IBM Management Center for WebSphere Commerce, or Management Center, is a suite of tools to support store management, merchandising, and marketing tasks for business users (Sellers, Marketing Managers, Product Managers, Catalog Managers, and others).
- Tools Available
 - Catalog
 - Promotion
 - Marketing
 - Asset
 - Installation
 - Store Management
 - Commerce Composer
 - Catalog Filter and Pricing
 - Workspace Management



© Copyright IBM Corporation 2016

Catalog tool

- Create and manage master and sales catalogs, including:
 - defining categories
 - creating products and SKUs
- categorizing SKUs
- bundling SKUs
- managing dynamic kits
- Enrich and manage product content, and create merchandising associations.
- Create, restore, view, and delete versions of catalog objects (sales catalog, category, and catalog entries).
- Create and manage search term associations, and associate search terms with landing pages.
- Manage product attributes with the Management Center attribute dictionary.
- Manage search engine optimization (SEO) properties for categories and catalog entries.

Promotion tool

- Create and manage promotions for your merchandise.
- Review and manage promotions.
- Import promotion codes for use with your promotions.
- Export promotion codes for a promotion.

Marketing tool

- Design creative marketing campaigns that can contain web and email activities.
- Create experiments to test your campaign activities' effectiveness.
- Create and manage marketing content.
- Define the flow of marketing activities.
- Review and manage marketing activities.
- Create and manage search rules.

- Create and manage search rule experiments.

Asset tool

- Browse attachments, managed files, and managed directories.
- Create, change, and delete attachments.
- Create, change, and delete managed files.
- Create and delete managed directories.
- Upload files, such as product images and marketing content.

Catalog filter and pricing

- Create, and manage catalog filters.
- Create and manage price lists and price rules.
- Upload catalog filters.
- Upload price lists.

Commerce Composer

- Create pages for your store.
- Create layouts for pages by using predefined widgets and templates.
- Assign layouts to store pages.
- View layouts.

Store management

- Update store profile information such as the name or description of a store, contact information, store location, and supported languages and currencies.
- Change the style of a store, including the store layout and color.
- Specify the functions that are available in your store, such as which optional fields to display on store pages.
- View your changes in the store.
- Open or close a store.
- Update the URL keyword of a static store page for Search Engine Optimization (SEO) friendly URLs.

Installation tool

- Create installment rules to offer customers different installment payment options when they purchase items from your store catalog.
- Review how installment rules are scheduled in relation to one another in a calendar tool.
- Review the list of active and inactive installment rules.

Workspace management

- Manage content within a workspace by selecting tasks, managing content within tasks, adding comments, and approving or rejecting tasks.
- Work on assigned tasks, complete tasks, and submit tasks for approval.
- View, open, and compare business object change history records to quickly see what modifications were made.
- Undo change history records to correct mistakes or to approve a task without needing to reject it.
- View the change history for a task group after the task group is approved or canceled.

Cache monitor

- Dynamic Cache Monitor application for displaying cache statistics, Edge Side and disk statistics, cache entries, dependency IDs, and cache policy information.
- URL is [https://admin host name:<port>/cachemonitor](https://admin_host_name:<port>/cachemonitor)

The screenshot shows the 'Cache Monitor' interface for a 'baseCache'. The left sidebar lists navigation options: Cache Statistics, Edge Statistics, Cache Contents, Cache Contents: PushPull Table, Dependency IDs, Disk Offload, Cache Policies, MBean Statistics, and Compare Cache Contents. The main area displays 'Cache Statistics' with buttons for Refresh Statistics, Reset Statistics, and Clear Cache. A table provides detailed statistics:

Statistic	Value
Cache Size	2000
Used Entries	2
Cache Hits	528
Cache Misses	40
LRU Evictions	0
Explicit Removals	9
Default Priority	3
Servlet Caching Enabled	Yes
Disk Offload Enabled	No

© Copyright IBM Corporation 2016

Development environment tips

- **Start WebSphere Commerce Developer**
 - From the command line:
 - Go to WCDE_installdir\bin.
 - Run the startWCToolkit.bat command
 - Browse the **Start** menu to manually start IBM Rational Application Developer for WebSphere Software and display the WebSphere Commerce Developer workspace.
- **Starting the test server**
 - Shut down any applications that use ports 80, 443, 8000, 8002, 8004, 8006, and 8007. These ports are required by the test server.
 - In the **Servers** view, right-click **WebSphere Commerce Test Server** and choose **Start**. Initialization is complete when you see text in the console that is similar to the following message: ADMU3000I: Server server1 open for e-business
 - If you want to debug, choose **Debug** instead of **Start**.

© Copyright IBM Corporation 2016

Development environment tips

- Changing the Database type
 - WCDE_installdir\bin\setdbtype.bat script
- Removing published Starter Stores
 - WCDE_installdir\bin\resetstores.bat
- Restoring Derby database to bootstrap
 - WCDE_installdir\bin\resetdb.bat
- Restoring Developer to default set up
 - WCDE_installdir\bin\restoreDefault.bat

© Copyright IBM Corporation 2016

Removing published starter stores

When you initially install WebSphere Commerce Developer, the Aurora extended sites starter store is published. If you decide that you do not want the starter store, use the WCDE_installdir\bin\resetdb.bat command to restore the database to the bootstrap configuration. Use the WCDE_installdir\bin\resetstores.bat command to reset the Stores web module back to its bootstrap state. Ensure that you create a backup before you run these scripts. The resetdb.bat script applies only to the Apache Derby database. If you want to publish a store, use the administration console.

Changing the database type

By default, WebSphere Commerce Developer is configured to use Apache Derby as the development environment database. If you want to use a different database, you must install your preferred database and change the database type from Apache Derby to your preferred database. Use the WCDE_installdir\bin\setdbtype.bat script to change the database type. This script configures your environment to use an existing database and optionally create a new bootstrap database. Running the setdbtype.bat script with no arguments displays the usage.

Ensure that IBM Rational Application Developer for WebSphere Software is not active when you run the setdbtype.bat script. If IBM Rational Application Developer for WebSphere Software is active, rerun the setdbtype.bat script after you stop IBM Rational Application Developer for WebSphere Software.

When you are creating a bootstrap database, run the WCDE_installdir\bin\resetstores.bat command to reset the Stores web module back to the bootstrap state.

Restoring WebSphere Commerce Developer to the default setup

Run the WCDE_installdir\bin\restoreDefault.bat script to restore the WebSphere Commerce

Developer to the default configuration. The restoration is done at the maintenance level of the installation and not at the base level.

Understand the co-existence of V7 and V8 toolkits

© Copyright IBM Corporation 2016

This section provides an understanding on the co-existence of WebSphere Commerce Version 7 and Version 8 toolkits.

Co-existence of V7 and V8 toolkits (1)

- Only V7 toolkit can co-exist with V8 toolkit
- Rational Application Developer must be installed in separate directories
- Installation Manager manages both installations
 - Manages the installations as separate package groups
- Install WebSphere Application Servers in separate directories

© Copyright IBM Corporation 2016

Note: Installation Manager may replace the WC 7 toolkit desktop shortcuts to workspace with WC 8.

IBM WebSphere Commerce Developer tool kit V8 can coexist on the same machine with only IBM WebSphere Commerce Developer Version 7.0 or any version 7 fix pack.

Coexistence with other versions of IBM WebSphere Commerce Developer is not supported.

RationalApplication Developer must be installed in separate directories.

Installation Manager manages both installation by treating separate package groups for V7 and V8.

Similar to Rational Application Developer, WebSphere Application Servers also needs to be installed in separate directories.

Co-existence of V7 and V8 toolkits (2)

- Database considerations
 - Apache Derby installs are separate for both toolkits, so there are no issues
 - DB2 – only one version of DB2 can be installed.
 - V8 toolkit requires a 64-bit DB2 installation.
 - Whereas V7 toolkit requires a 32-bit DB2 installation.
 - Both can not co-exist.
 - For more details refer to the Knowledge Center topic titled – “Multiple DB2 copies on the same computer (Windows)” : https://www-01.ibm.com/support/knowledgecenter/SSEPGG_10.5.0/com.ibm.db2.luw.qb.server.doc/doc/c0022772.html
 - Oracle – Refer to Oracle online documentation
<https://docs.oracle.com/database/121/>

© Copyright IBM Corporation 2016

This section describes about database considerations.

Apache Derby database is installed separately for both toolkits, so that there will not be any issues.

For DB2, only one version of DB2 can be installed. Version 8 toolkit requires a 64-bit DB2 installation whereas Version 7 toolkit requires a 32 bit installation, both cannot co-exist.

DB2 does not support multiple versions to be installed on Windows, because the DB2 32-bit and 64-bit registries are stored in different locations. Once the DB2 64-bit version is installed, the 32-bit version is removed from the system. For more information, refer to the Knowledge Center topic titled "Supported Databases".

For the Oracle database, refer to the online documentation.

Understand the process to install and configure developer toolkit on single or multiple workstations

© Copyright IBM Corporation 2016

This section gives an overview for Understanding the process to install and configure developer toolkit on single or multiple workstations.

High level steps for a single development workstation (1)

- Verify the hardware, software and operating system requirements
- Install and prepare 64-bit Installation Manager V1.8.4
- Install Rational Application Developer or Rational Software Architect Version 9.5
 - Apply fix pack 1 to upgrade the version to 9.5.0.1
- Install and prepare a 64-bit WebSphere Application Server
 - You can install either one of the following
 - WebSphere Application Server Test Environment V8.5.5.7
 - WebSphere Application Server Full Version V8.5.5.7
- Install a 64-bit version of SDK Java Technology Edition Version 7.0 SR4 or any later Version 7.0 Service Refresh

© Copyright IBM Corporation 2016

This covers High level steps for a single development workstation. Firstly verify the hardware, software and operating system requirements.

Install and prepare 64-bit Installation Manager Version 1.8.4 or higher

Install Rational Application Developer or Rational Software Architect Version 9.5 or higher.

Apply fix pack 1 to upgrade the version to 9.5.0.1. Refer to the Knowledge Center for downloading RationalApplication Developer Fix Pack 1.

Install and prepare a 64-bit WebSphere Application Server. This can be either WebSphereApplication Server Test Environment Version 8.5.5.7 or WebSphere Application Server Full Version 8.5.5.7 or higher.

Install a 64-bit version of SDK Java Technology Edition Version 7.0 S R 4 or any later Version 7.0 Service Refresh.

High level steps for a single development workstation (2)

- Apply fixes to Rational Application Developer
- Install DB2 or Oracle if Apache Derby is not the choice of database
- Install WebSphere Commerce Developer
 - Apply fix pack 1 from Fix Central.
- Run search pre-process and build index utilities to update the Solr server index
- Publish the Aurora starter store, if DB2 or Oracle is the choice of database
 - Starting from WebSphere Commerce version 8, only extended sites store archives are supported.
 - If database of choice is Derby, then the extended sites store archive is already published
- Verify the stores and tools are functioning properly

© Copyright IBM Corporation 2016

Note:

1. Refer to <http://www-01.ibm.com/support/docview.wss?uid=swg24041495> for downloading WebSphere Commerce Developer Fix Pack 1.
2. Use can IBM Installation Manager to install the above fix pack instead of IBM Update Installer 8.0 for WebSphere Commerce V8.0.

Apply fixes to Rational Application Developer. Then Install DB2 or Oracle if Apache Derby is not the choice of database. Install WebSphere Commerce Developer. Apply fix pack 1 from Fix Central.

Refer to the Knowledge Center for downloading WebSphere Commerce Developer Fix Pack 1.

You can use IBM Installation Manager to install the above fix pack instead of IBM Update Installer 8.0 for WebSphere Commerce Version 8.0

Run search pre-process and build index utilities to update the Solar server index.

Publish the Aurora starter store, if DB2 or Oracle is the choice of database.

Starting from WebSphere Commerce version 8, only extended sites store archives are supported.

If database of choice is Derby, then the extended sites store archive is already published.

Make sure to verify that the stores and tools are functioning properly.

Switching the database

- By default, WebSphere Commerce Developer is configured to use Apache Derby as the development environment database
- If you want to use a different database, you must install your preferred database and change the database type from Apache Derby to your preferred database
- Sequence of steps to switch the database type
 - Run the `WCDE_installdir\bin\restoreDefault.bat` script to restore the WebSphere Commerce Developer to the default configuration.
 - Use the `WCDE_installdir\bin\resetstores.bat` command to reset the Stores web module back to its bootstrap state.
 - Use the `WCDE_installdir\bin\setdbtype.bat` script to change the database type
 - Publish the store archive of choice using the WebSphere Commerce Administration console
 - *Optional:* Run `di-preprocess` and `di-buildindex` to update search index

© Copyright IBM Corporation 2016

This section covers about Switching the database.

By default, WebSphere Commerce Developer is configured to use Apache Derby as the development environment database

If you want to use a different database, you must install your preferred database and change the database type from Apache Derby to your preferred database.

Sequence of steps to switch the database type

Run the `WCDE_installdir\bin\restoreDefault.bat` script to restore the WebSphere Commerce Developer to the default configuration.

Use the `WCDE_installdir\bin\resetstores.bat` command to reset the Stores web module back to its bootstrap state.

Use the `WCDE_installdir\bin\setdbtype.bat` script to change the database type

Publish the store archive of choice using WebSphere Commerce admin console

Run `di-preprocess` and `di-buildindex` to update search index which is optional.

High level steps for multiple development workstations

- Install and configure IBM Packaging utility
- Create a central repository
- Make your central repository available over the local intranet
- Maintain your central repository
 - Ensure the central repository contains fixpacks released every two weeks
 - Ensure the central repository contains mod packs released every quarter
- Install software with the IBM Installation Manager or Enterprise Deployment Utility
- Verify that the stores and tools are functioning properly

© Copyright IBM Corporation 2016

This unit covers High level steps for multiple development workstations.

Firstly Install and configure IBM Packaging utility. Then Create a central repository and make your central repository available over the local intranet.

Maintain your central repository by ensuring that central repository contains fixpacks released every two weeks and mod packs released every quarter.

Install software with the IBM Installation Manager or Enterprise Deployment Utility. Verify that the stores and tools are functioning properly.

Publishing a given store archive

© Copyright IBM Corporation 2016

This section talks about Publishing a given store archive.

Store archives

- A store archive file (.sar) is a ZIP archive file that contains all the assets necessary to create a store or site.
 - For example, Extended Sites store archive is packaged as ExtendedSites.sar.
- Store archive files are used to package and deliver organization structures, predefined user roles, and access control policies necessary to create the environment for your store or site.
- The archive files are used to package collections of assets that are used to create a catalog or storefront.
- The default location for the store archive is
`<WC_installdir>/starterstores`

© Copyright IBM Corporation 2016

A store archive file (.sar) is a ZIP archive file that contains all the assets necessary to create a store or site.

Store archive files are used to package and deliver organization structures, predefined user roles, and access control policies necessary to create the environment for your store or site.

The archive files are used to package collections of assets that are used to create a catalog or storefront.

The default location for the store archive is Websphere Commerce installation directory/starterstores

Verifying the store publish

- Go to **Store Archives > Publish Status** option.
- Check whether the sar file publish is successful
- Launch the stores using the links giving in hints and tips file
- Ensure all parts of the homepage are loaded properly
- Browse the catalog and optionally, place a guest order
- Open business user tooling like IBM Management Center for WebSphere Commerce

© Copyright IBM Corporation 2016

In order to check if the store was published successfully or not. Go to Store Archives and select the Publish Status option.

Check whether the sar file publish is successful.

Launch the stores using the links giving in hints and tips file. Ensure all parts of the homepage are loaded properly.

Browse the catalog and optionally, place a guestorder.

Open a business user tool. For example, IBM Management Center for WebSphere Commerce.

Workspaces Overview

© Copyright IBM Corporation 2016

This section provides an overview to Understand the Workspaces in WebSphere Commerce Version 8.

Workspaces overview

- Access-controlled work area without affecting the current site
- Provides to have own copy of managed assets
- Option to change and preview available without affecting assets outside the workspace
- Features
 - Task groups and tasks for dividing work within workspaces
 - Defined roles for managing workspaces, workspace-related approvals, and contributing content
 - Locking policies to control changes within a workspace
 - Various forms of commit and publish to control how data is moved from task groups to the production-ready data and the production data

Available only in

- WebSphere Commerce Professional
- Enterprise editions
- Workspace Management tool
 - Present in Management Center
 - Options available are
 - manage and edit workspaces
 - manage and edit tasks and tasks group

© Copyright IBM Corporation 2016

A workspace is an access-controlled work area where you can make and preview changes to managed assets, without affecting what is currently running on your site. It Provides to have own copy of managed assets.

There is an option to make and preview changes without affecting managed assets outside the workspace.

Features available are Task groups and tasks for dividing work within workspaces ,Defined roles for managing workspaces, workspace-related approvals, and contributing content, Locking policies to control changes within a workspace and Various forms of commit and publish to control how data is moved from task groups to the production-ready data and the production data. Workspaces are available only on WebSphere Commerce Professional and Enterprise editions only. Workspace Management tool in Management Center to manage and edit workspaces, its tasks groups, and tasks

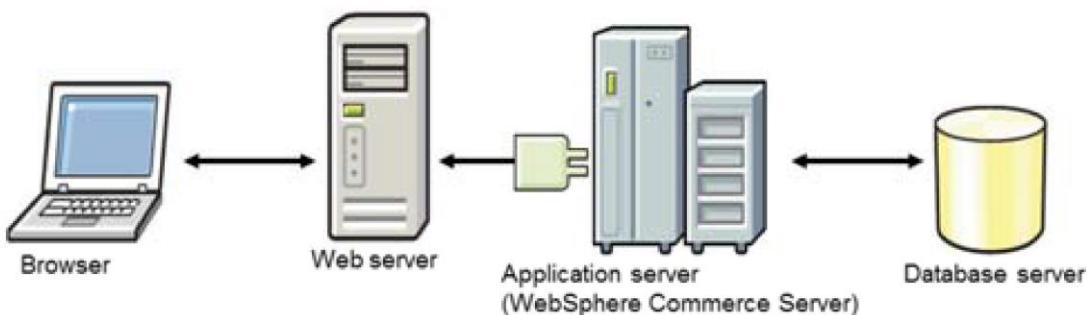
Understand logging, tracing and troubleshooting

© Copyright IBM Corporation 2016

This section provides an overview to Understand logging, tracing and troubleshooting in WebSphere Commerce Version 8.

Problem areas for examination

- Runtime issue could be anywhere along the path:
 - **Browser:** Client side JavaScript, Ajax requests, cookie acceptance, cookie, or URL limits, windows.
 - **Web Server:** Virtual hosts, SSL certificate, rewrites, redirects, ports.
 - **WebSphere Application Server plug-in:** Mapped modules, cluster, load balancing, transports, virtual hosts.
 - **Application Server:** Servlet filters, Commerce servlets, caching.
 - **Database:** connections, SQL, contention, data itself.



© Copyright IBM Corporation 2016

The basic topology of a WebSphere Commerce solution includes these components. It becomes more complex as other elements, such as a firewall and clusters, are introduced.

Problems might occur at any one of these elements, and even more so when other elements are introduced. When determining a problem, you need to know what to look for at each tier. When focusing on WebSphere Commerce, component-specific Must Gathers are a good place to start. *SystemOut.log* and *trace.log* can point you to the next step:

- See the logged request, but input seems wrong (check browser).
- Application exceptions (application itself is the issue, or the data).
- Rollback, timeout, locking exceptions (consider the database).
- Simply wrong results (check the application code, or the data and config).

Troubleshooting Ajax

Example using Firebug to inspect...

- Request



- Request Parameters

The screenshot shows the Firebug Net tab for the AjaxInterestItemAdd request. Under "Params", the following parameters are listed:

- URL: SuccessfulAJAXRequest
- catEntryId: 10701
- catLangId: 10101
- langId: -1
- storeId: 10051

- Response

The screenshot shows the Firebug Net tab for the AjaxInterestItemAdd request. Under "Response", the JSON data is displayed:

```
{  
    "langId": "-1",  
    "storeId": "10051",  
    "URL": "SuccessfulAJAXRequest",  
    "catEntryId": "10701",  
    "catalogId": "10101"  
}
```

© Copyright IBM Corporation 2016

Logging at different Levels

- Running a WebSphere Commerce site involves logging at multiple product levels:
 - Web Server [If web server is configured for toolkit]
 - Application Server
 - Search Server
 - Database
 - Message Server, if any
 - Log files in <WCDE_InstallDir>/logs directory
- Validate the traces are enabled before reproducing the issue
- Ensure there are enough logs defined to capture the issue
- Clean the log directory before reproducing the scenario
- Ensure that the error is captured in the log file

© Copyright IBM Corporation 2016

Logging can be identified at different Levels.

Running a WebSphere Commerce site involves logging at multiple product levels like Web Server, Application Server, Search Server, Database and Message Server. Log files reside in WebSphere Commerce Installation directory / logs folder.

Follow the below steps before reproducing the issue to capture the logs.

Firstly validate whether traces are enabled. Then ensure that, there are enough logs defined to capture the issue.

Clean the log directory before reproducing the scenario. Ensure that the error is captured in the log file.

Tracing

- Easiest way to see which code is involved
 - Which classes, which methods, what inputs, what outputs
- Find the appropriate traces to enable from both Knowledge Center page and MustGather documentation.
- MustGather documents aid in the problem determination process and saves time while working with Problem Management Records (PMRs)
 - Refer to <http://www-01.ibm.com/support/docview.wss?uid=swg21440712>
- Important: Be sure to include traces within your own custom code to aid in serviceability.

© Copyright IBM Corporation 2016

This section describes about tracing in WebSphere Commerce Server.

Tracing provides the Easiest way to see the classes , methods,inputs and outputs in the code which is involved.

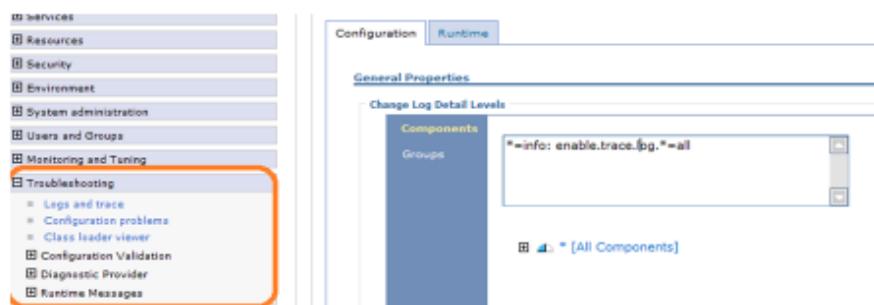
You can find different types of traces detailed information in Knowledge Center page, or MustGather documents.

MustGather documents helps in problem determination and save time resolving Problem Management Records .

Another important point is to include the traces in your own custom code too, so it is serviceable effectively.

Enabling trace

- Use logging and tracing to help diagnose problems, even for custom code
- Set trace levels in the administration console
- You can set the trace active at startup ("Configuration"), or only for the currently running process ("Runtime")



© Copyright IBM Corporation 2016

Working with trace

- Use trace to obtain detailed information about running the application server components, including application servers, clients, and other processes in the environment
- Traces will, by default, be written to trace.log in the application server's log directory
- Typically WASProfile_Home\logs\<servername>
- For example:
- /opt/IBM/WebSphere/AppServer/profiles/demo/logs/server1
- C:\IBM\WCDE_ENT80\wasprofile\logs\server1
- Use a "tail" program to monitor the trace while executing code, if possible
- Never open a SystemOut.log or trace.log in an editor like vi, nano, or emacs!
- These editors create temporary files in the /tmp file system which can impact server functionality
- Instead, use viewing tools like tail, pg, less, or more.

© Copyright IBM Corporation 2016

Troubleshooting

- Problems can arise during any phase of the project:
 - Design and planning
 - Development stage
 - Testing
 - Launch
 - Maintenance or upgrades
 - Runtime staging/QA/production issues
- Important to isolate specific problem, know how and where to find answers
- Commerce **Non-runtime** problems include install, configuration, deployment, feature enablement, development
- Commerce **Runtime** problems include orders, payments, web services, messaging, users, and cookies, performance

© Copyright IBM Corporation 2016

Problems can arise during any phase of the project. It could be during Design and planning, Development stage, Testing, Launch, Maintenance or upgrades, Runtime staging/QA/production issues.

It is important to isolate specific problem, know how and where to find answers.

Some of the Commerce Non-runtime problems include install, configuration, deployment, feature enablement, development.

Some of the Commerce Runtime problems include orders, payments, web services, messaging, users, and cookies, performance.

IBM Support

- Any product behavior that negatively impacts business is a problem
- WC Support Recommends...
 - Learn from the experiences of others. Help yourself by looking up past problems and resolutions
 - Search [IBM Support Portal](#) for technical information, including
 - Flashes and Alerts,
 - Technotes,
 - Known problems and resolutions,
 - Updates and fixes,
 - Version Compatibility,
 - Diagnostic tools
 - Find fixes and updates from [Fix Central](#)
 - Subscribe to automatic email notifications to stay informed about new and updated content.
 - If you need to call Support, **Prepare** before the call

© Copyright IBM Corporation 2016

You can contact IBM Support when there is any problem with the product behavior that negatively impacts the business.

WebSphere Commerce Support Recommends to Learn from the experiences of others. Help yourself by looking up past problems and resolutions . Search IBM Support Portal for technical information, including Flashes and Alerts , Tech notes, Known problems and resolutions, Updates and fixes, Version Compatibility and Diagnostic tools.

Find fixes and updates from Fix Central.

Subscribe to automatic email notifications to stay informed about new and updated content.

If you need to call Support, Prepare before the call.

Unit summary

This course was designed to enable you to:

- Get an overview of WebSphere Commerce development environment
- Understand the co-existence of V7 and V8 toolkits
- Understand the process to install and configure developer toolkit on single or multiple workstations
- Publishing a given store archive
- Workspaces overview
- Understand logging, tracing and troubleshooting
- Demonstration of exploring the toolkit environment

© Copyright IBM Corporation 2016

This course was designed to provide an overview of WebSphere Commerce development environment.

Understand the co-existence of version 7 and version 8 toolkits.

Understand the process to install and configure developer toolkit on single or multiple workstations.

Publishing a given store archive.

Understand more about logging, tracing and troubleshooting.

Describe the WebSphere Commerce developer tools and to provide a Demonstration that explores the toolkit environment.

Additional training

For additional training needs, you have several options available for registering.

You can register	At
On site	This or any of our training facilities
Online	http://ibm.com/training/
Email	ibmswedu@us.ibm.com
By telephone	1-800-IBM Teach (1-800-426-8322)
By fax	208-692-6130

© Copyright IBM Corporation 2016

For additional training needs, you have several options available for registering like Onsite, Online , Email , by telephone or by Fax .

Unit 4

WebSphere Commerce V8 Presentation layer

© Copyright IBM Corporation 2016

This section describes an overview of the WebSphere Commerce V8 Presentation Layer.

Unit objectives

This course is designed to enable you to:

- Describe the behavior of the presentation layer in WebSphere Commerce.
- Understand WebSphere Commerce Struts Framework
- List the process for modifying the behavior of WebSphere Commerce Storefront Assets.
- Understand WebSphere Commerce Tag library JSTL.
- Understand and manipulate WebSphere Commerce data beans.
- Analyze the architecture of the Web 2.0 store.
- Design interactive Storefronts by using the Dojo toolkit and WebSphere Commerce extensions.
- Construct agile storefronts by using Asynchronous Java and XML (AJAX).
- Understand WebSphere Commerce Spring Framework.
- Understand the basics of Commerce Composer.
- Understand the basics of Commerce Management Center Customization.
- Understand the basics of Responsive Web design.

Copyright 2016 IBM Corporation

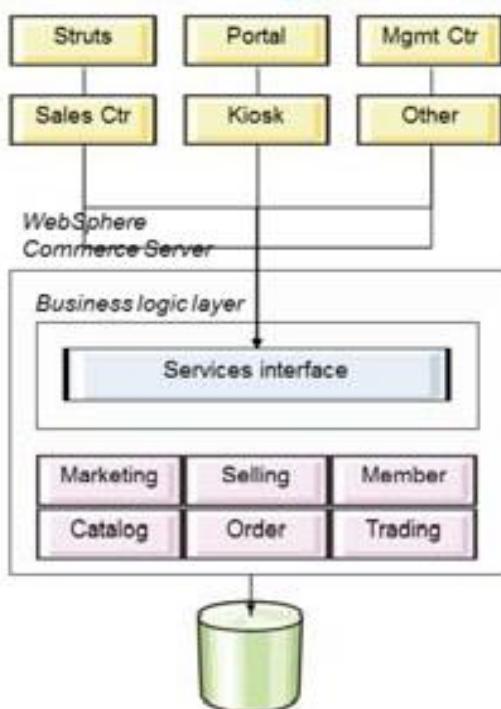
Developing the Web channel

After completing this topic, you should be able to:

- Describe the behavior of the presentation layer in WebSphere Commerce.
- List the process for modifying the behavior of WebSphere Commerce storefront assets.
- Work with JavaServer Pages.
- Globalize storefronts.
- Outline the business processes used in sample stores.
- Prepare property files.
- Manipulate data beans.
- Comply with recommended development strategies and best practices.

© Copyright IBM Corporation 2016

Multi-channel support in WebSphere Commerce



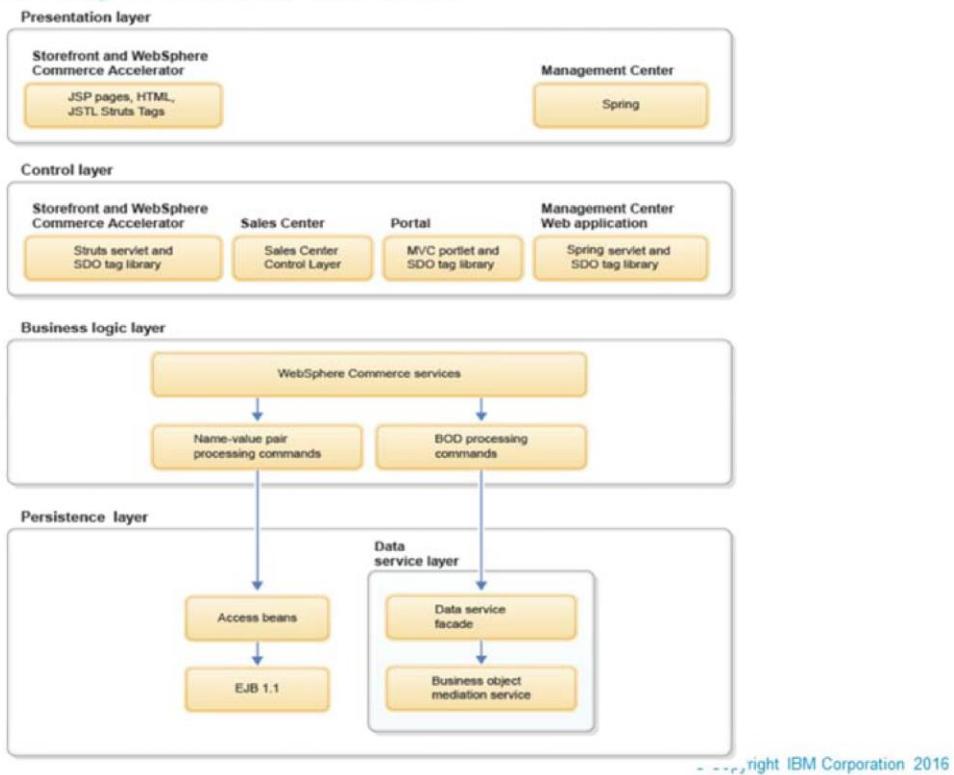
- Presentation layer is decoupled from the business logic layer through the Services interface.
- Multiple types of sales channels might be connected to the WebSphere Commerce Server by using the Services interface.
 - Each sales channel might support its own unique architecture.
- Access to the database from the WebSphere Commerce Server is implied through the persistence layer.

© Copyright IBM Corporation 2016

Many different types of channels can be used in WebSphere Commerce. The web storefront is the most common, but a mobile app is considered a presentation layer channel, because it communicates through a common business logic services interface.

Details — Because each channel in the presentation layer might support its own technology and architecture, there are several technologies and architectures to learn in this course. Management Center and Web 2.0 is discussed in greater detail later in the class, as well as web services, when using WebSphere Commerce as a consumer.

Presentation layer in Architecture



Regardless of the channel, the business logic facade, a generic interface implemented as a stateless session bean, is used by controller calls to invoke controller commands. The command layer is implemented as WebSphere Commerce commands.

Details — Because each channel in the presentation layer might support its own technology and architecture, there are several technologies and architectures to learn in this course. Management Center and Web 2.0 is discussed in greater detail later in the class, as well as web services, when using WebSphere Commerce as a consumer.

Architecture of the Web channel

- Controller
- Actions
 - Struts action (com.ibm.commerce.struts.BaseAction)
- Adapter framework and adapters
 - HTTP browser adapter
 - HTTP PvC adapter
 - HTTP program adapter
- Data Bean Manager
 - Data beans
- Display Pages (JSP)

© Copyright IBM Corporation 2016

WebSphere Commerce uses JavaServer Pages (JSP) to implement the view (presentation) layer of the model-view-controller (MVC) design pattern. The view layer is in charge of retrieving data from the database by using data beans and formatting it to meet the display requirements. The view layer determines whether the request is sent to a browser or streamed out as XML, WML, or other MIME formatted data flow. JSP files present a clean separation between data content and presentation.

Purpose — Describe the use of JSP to create the view layer for the various web modules that make up WebSphere Commerce.

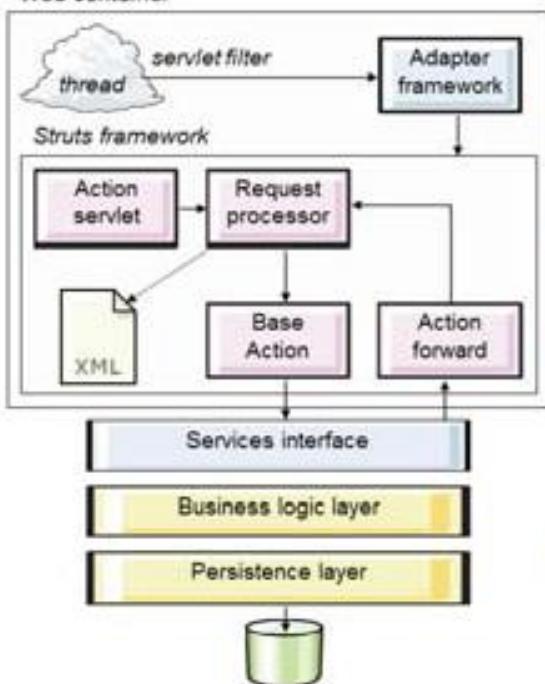
Details — A web module (WAR) combines Java code and JSP files to present a view and encapsulate business process logic to drive the users display.

Additional information — JSP presents a mechanism similar to ASP that allows for the easy embedding of HTML or other markup language with Java code scriptlets. JSP files are first compiled down to servlet classes, whose output is cached for better performance. Essentially JSP files are shorthand for writing display servlets that the View portion of the model-view- controller architecture that is implemented by WebSphere Commerce and many other Java EE applications uses.

Transition statement — How does it all fit together?

Framework interaction in the web channel

Web container



- A request is directed to the presentation layer in the web container via its own thread.
- Specific type of adapter (web adapter, in this example) is determined.
- Struts action invokes a command through the services layer (referred to as the services interface).
- Command executes a controller command which:
 - Accesses the database through the persistence layer.
 - Calls one or more task commands, which can then access the database.
 - A combination of the two.
- Response is returned as a set of properties.

© Copyright IBM Corporation 2016

1. The request is directed to the Presentation layer (web container) in its own thread.
2. The thread that handles the request is dispatched to the WebSphere Commerce servlet filter. The filter passes the request to the adapter framework.
3. The adapter manager determines which adapter supports handling the request, and then returns that adapter to associate with the request. For example, if the request originated from an Internet browser, the adapter manager associates the request with the HTTP browser adapter. The adapter is passed back to the Dynacache servlet container. The filter regains control and passes the request to the servlet engine for processing. Either of the following actions can occur:
 - The request can be cached and the cached response can be returned.
 - If the request is not cached, the Struts action invokes the business logic facade by specifying the interface name of the business logic to invoke and the associated parameters. The business logic facade queries the command registry to determine the appropriate implementation for the store that is associated with the request.
4. The business logic facade invokes the appropriate controller command.
5. The controller command begins execution:
 - The controller command can access the database by using an access bean and its corresponding entity bean.
 - The controller command can invoke one or more task commands. Task commands can then access the database by using access beans and their corresponding entity beans.
 - A combination of a and b.
6. The business logic facade returns a set of properties to the Struts action. One of the elements that is part of the properties is the key to the global forward that represents the response.
7. The action looks up the global forward in the Struts configuration files. It resolves to the right one based on the store configuration. The action forward implementation that is selected is the appropriate one for the device of the request.
8. The Struts request processor executes the action forward that executes the appropriate JSP

- page. Within the JSP page, a data bean is required to retrieve dynamic information from the database. The data bean manager is used to activate the data bean.
9. The access bean from which the data bean is extended accesses the database using its corresponding entity bean. Based on globalization information in the request, the data bean formats the data.

Controller

The controller plays a role in enforcing the programming model for the WebSphere Commerce application. For example, the programming model defines the types of commands that an application can implement. Each type of command serves a specific purpose. Business logic must be implemented in controller commands. The controller expects the controller command to return a view name.

Action

In a typical Struts application, many logic details are handled in Struts actions. Because the WebSphere Commerce framework uses the command pattern, a Struts action is used to pass information back to the session facade.

Adapter framework and adapters

The adapter framework determines which adapter supports handling the request and associates the adapter with that request for response building and session management. WebSphere Commerce adapters are device-specific components that perform processing functions before passing a request to the web controller. Specific adapters that come with WebSphere Commerce are the HTTP browser adapter, HTTP (Pervasive Computing) PvC adapter, and HTTP program adapter.

- HTTP browser adapter:** provides support for requests to invoke WebSphere Commerce commands that are received from HTTP browsers.
- HTTP PvC adapter:** an abstract adapter class that can be used to develop specific PvC device adapters. For example, if you needed to develop an adapter for a particular cellular phone application, you would extend from this adapter.
- HTTP program adapter:** provides support for remote programs that invoke WebSphere Commerce commands. The program adapter receives requests and uses a message mapper to convert the request into a CommandProperty object. After the conversion, the program adapter uses the CommandProperty object and executes the request.

Data bean manager and data beans

WebSphere Commerce data beans that are inserted into JSP pages allow for the inclusion of dynamic content in the page.

The data bean manager activates the data bean so that its values are populated when the following line of code is inserted into the page:

```
com.ibm.commerce.beans.DataBeanManager.activate(data_beans, request, response)
```

The bean is handled by using the wcbase:usebean tag, where **data_beans** is the data bean to be activated, **request** is an HttpServletRequest object, and **response** is an HttpServletResponse object.

Display pages

WebSphere Commerce uses JavaServer Pages for display.

Purpose — WebSphere Commerce uses a combination of architecture and design patterns. Struts, business logic facade, controller, command, MVC, data access objects, EJB, and many others are used to implement the WebSphere Commerce application. WebSphere Commerce provides a set of adapter framework classes to support different device and program behaviors.

Details — Examples of processing tasks that an adapter performs include:

- Instructing the web controller to process the request in a manner specific to the type of device. For example, a pervasive computing (PvC) device adapter can instruct the web controller to ignore HTTPS checking in the original request.

- Transforming the message format of the inbound request into a set of properties that WebSphere Commerce commands can parse.
- Providing device-specific session persistence.

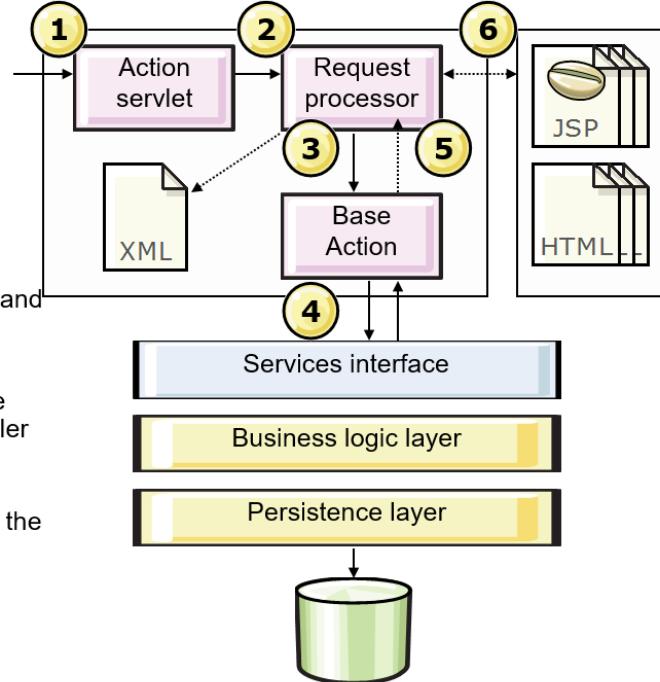
Additional information —

http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/csdruntime.htm

Transition statement — Move on to discuss WebSphere Commerce and the Struts framework.

WebSphere Commerce and the Struts framework

- Action servlet:
 - ECActionServlet implemented for WebSphere Commerce.
 - This servlet must be used.
- Request processor:
 - Basic Struts processor,
 - Can replace this processor to add Tiles support.
- Action mapping:
 - Map request pattern to action and parameters.
 - Three files, base definitions, extensions, and migration.
- BaseAction:
 - Single action, BaseAction, calls to façade with an argument that defines the controller command interface.
 - Controller command returns view name.
 - Return ActionForwards, which determine the JSP to display.
- JSP page:
 - Dynamic pages.



© Copyright IBM Corporation 2016

Why Struts?

- Mature and full-featured framework for Java EE web Application development
- Industry standard for deploying a Java EE MVC (model-view-controller) architecture
- Enforces best practices and design patterns
- Supported by IBM development tools, such as Rational Application Developer, and third-party utilities
- Large developer community base such as books, message groups, and websites that are dedicated to Struts

Advantages

- Form validation
 - Scripted field validation removes extensive parameter checking from the commands.
 - Facilitates greater command reuse since mandatory parameter requirements are configurable
- Tag libraries
 - Bean and HTML
 - Ability to display initial value or redisplay entered values
- Support for Tiles
 - Create page templates for layouts.
 - JSP fragments inserted into the layouts.
- Web application-specific Struts configuration file
 - Restricts the URLs or views that are accepted for the web application
- Module-specific Struts configuration
 - Split application into multiple modules

Additional notes:

Purpose — An introduction to WebSphere Commerce Struts framework. Stress the fact that WebSphere Commerce uses Struts, but controller commands still do the work; those commands are in the model layer. The responsibility of the base action is to delegate to the facade, by using arguments that cause the proper controller command to be executed.

Details —

Steps in request processing:

1. Determines a number of request characteristics, such as locale and content type, necessary for further processing
2. Uses the Struts configuration file of the module to determine the configured action mapping for the request
3. Uses the Struts configuration file of the module to locate or instantiate an appropriate action form for the request, if necessary, and populates and validates it
4. Uses the Struts configuration file of the module to locate or instantiate an appropriate action for the request
5. Passes the request, action form, and action mapping to the action (step 3 in the preceding diagram)
6. Forwards the user to the appropriate view element when the action is completed (step 6 in the preceding diagram)

Additional information — The Struts homepage is <http://struts.apache.org>.

Struts configuration

- *struts-config.xml*
 - Core product configuration.
 - Used for the mappings in the WebSphere Commerce base.
 - This file should not be modified.
- *struts-config-ext.xml*
 - Used to store customized configurations (“extensions”).
 - New mappings are added in this file.

© Copyright IBM Corporation 2016

WebSphere Commerce action mapping

The diagram illustrates the mapping between a URL and an interface name in the `struts-config.xml` file. A user icon is shown at a computer, with a red arrow pointing from the URL "AddressAdd" to the first row of the table. Another red arrow points from the "interface" column to a callout box containing the text: "Interface name that is passed to business logic facade (command alias)". A yellow circle labeled "1" is on the left of the table, and a yellow circle labeled "2" is on the right.

url	storeent_id	interface	https	Authen-ticated
AddressAdd	0	com.ibm.commerce.usermanagement.commands.AddressAddCmd	0	0
AddressAdd	201	com.ibm.commerce.usermanagement.commands.AddressAddCmd	1	1
StoreCatalog Display	0	com.ibm.commerce.catalog.commands.StoreCatalogDisplayCmd	0	0

© Copyright IBM Corporation 2016

In the Struts file, the entry might look like the example shown here:

```
<action-mappings type="com.ibm.commerce.struts.ECActionMapping">
<action path="/AddressAdd"
parameter="com.ibm.commerce.usermanagement.commands.AddressAddCmd"
type="com.ibm.commerce.struts.BaseAction">
    <set-property property="https" value="0:0,201:1"/>
    <set-property property="authenticate" value="201:1"/>
</action>
<action path="/StoreCatalogDisplay"
parameter="com.ibm.commerce.catalog.commands.StoreCatalogDisplayCmd"
type="com.ibm.commerce.struts.BaseAction"/>
</action-mappings>
```

url: the URL name.

storeent_id: is the store reference number for this URI or 0 to mean any store.

https: The value of 1 indicates that the request was expected to be received on a secure channel (HTTPS). A redirect to the SSL port is issued if it was received on an insecure channel (HTTP).

authenticated: The value of 1 indicates that user logon is required. If the user is not logged on when trying to access this URL, the user is redirected to the logon page before being able to proceed. For some actions, partially authenticated users might be entitled access to a resource.

Instructor notes:

Purpose — Explain configuration of Struts configuration in web channel.

Details — Here is a simple example of the use Struts, which is not changed from V6.

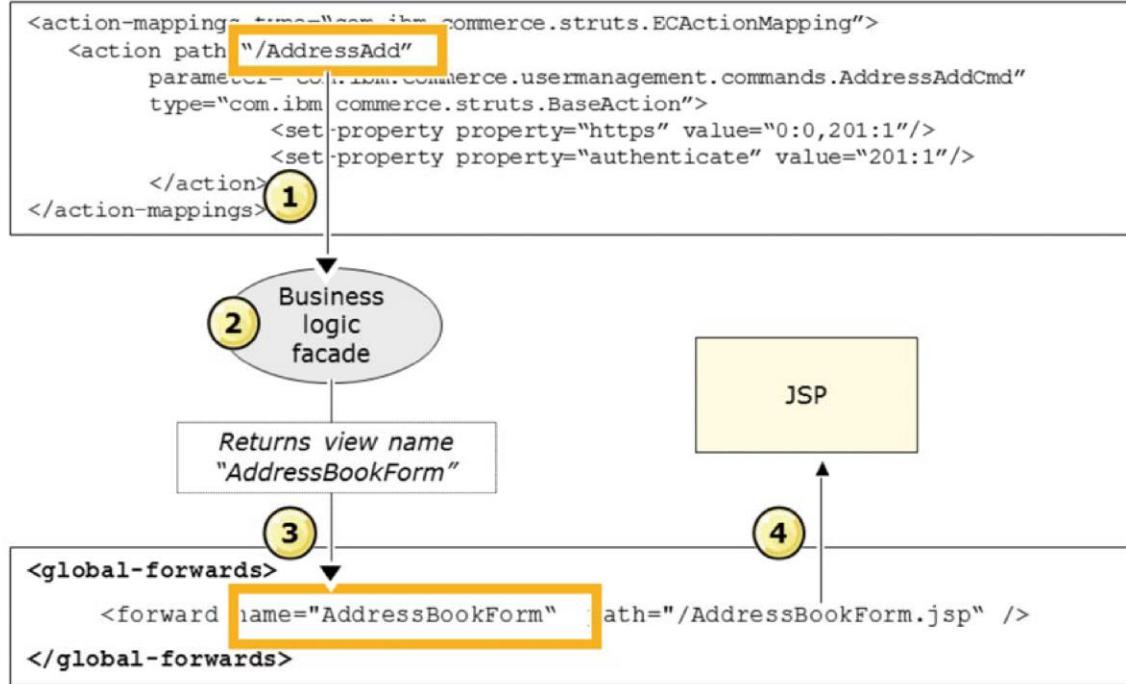
Additional information —

Transition statement — Next: WebSphere Commerce global forwards

Explain about how urls can be mapped to specific stores
Device format ID - The identifier of the device to which the view will be sent. The default device format is -1, which represents an HTTP Web browser.
Store ID - The store reference number for this URI or 0 to mean any store.
PropertiesName-value pairs used by this view, in the form of an HTTP request query string.
HTTPS

- The value of 1 indicates that the request was expected to be received on a secure channel (HTTPS) and a redirect to the SSL port will be issued if it has been received on an insecure channel (HTTP).
Credentials Accepted - The value of P indicates that partially authenticated users are entitled to access this resource.

WebSphere Commerce Global Forwards



© Copyright IBM Corporation 2016

In the AddressAdd example:

1. The URL from the request is mapped in the action mappings to an interface name (a command alias).
2. The command alias is passed through the business logic facade to execute in the business logic layer
3. The return value from the command execution includes the name of a view "AddressBookForm." That view name is mapped in global forwards to a JSP name.
4. The JSP is called, rendered, and returned to the caller.

Development of Struts components

- Customizations in WebSphere Commerce do not often involve customization of Struts components.
 - **ECActionServlet** is developed for WebSphere Commerce, and probably never has to be modified.
 - Request processors might be replaced (to provide Tiles support, for example).
 - Custom request processors might be developed, but it is not common.
 - Development of new Struts actions is rare; **BaseAction** acts as a delegate to the facade, where the business logic is implemented as a set of commands.

© Copyright IBM Corporation 2016

Storefront customization

© Copyright IBM Corporation 2016

This section describes an overview of storefront customization.

Storefront customization strategies

1. Use of existing components.
 - Through the WebSphere Commerce Accelerator and Commerce Management Center.
 - Many features are configurable through administrative tools.
2. Customize existing components.
 - Extensions or modification through WebSphere Commerce Developer.
 - If development of a store began with a starter store, most customizations are performed on existing pages or components.
3. Build new customization components.
 - New features that are developed with the WebSphere Commerce Developer.
 - In some cases, requirements call for the creation of new pages.
 - In many cases, requirements call for the creation of new components for an existing page, like Data beans.

© Copyright IBM Corporation 2016

The next unit covers developing and customizing the store business logic.

Additional notes:

Purpose — Initial topics start at a high level and then drill down to specific customization methods for the store.

Details — It is a common strategy for all development and customization in WebSphere Commerce (and, in fact, all IBM products): Configure what is there, extend if possible, and build if necessary.

Additional information —

Transition statement — Administrative storefront modifications

Administrative Storefront modifications

- WebSphere Commerce Accelerator toolset – Create new stores, suspend and resume existing stores
- The Store Management tool in Management Center replaces all of the store management features in WebSphere Commerce Accelerator

Type	State	Store Identifier	Description
		AuroraB2BESite	AuroraB2BESite
		AuroraB2BStorefrontAssetStore	Commerce Model Store entity
		AuroraESite	AuroraESite
		AuroraStorefrontAssetStore	Commerce Model Store entity
		Extended Sites Catalog Asset Store	Extended Sites Catalog Asset Store
		Extended Sites Hub	Extended Sites Hub

As a Seller, Site Administrator, or Channel Manager, you can use the Store Management tool in Management Center to manage your stores. The Store Management tool in Management Center replaces all of the store management features in WebSphere Commerce Accelerator, except store creation and the ability to suspend and resume stores. Continue to use the Store Creation wizard in WebSphere Commerce Accelerator to create new stores; continue to use WebSphere Commerce Accelerator to suspend and resume stores.

Manage store information

Change general store information, such as:

- Store name
- Description
- Contact information
- Location
- Supported languages
- Supported currencies

Change store style - Change the style of your store, including the store layout and color.

Select store functions - Specify the functions that are supported in your store, such as which optional fields to display on the Registration and Change Personal Information pages.

Select and view a store - Select a store to:

- View the store in a new browser window to determine what changes you want to make in the store.
- View the store in a new browser window after you make changes to the store.

Open or close a store - Specify whether to open or close a store. For example, you might want to close a store to perform maintenance and then open the store once this work is complete.

Find stores - Search for the store you want to work with by store name.

Store management changes between WebSphere Commerce Accelerator and Management Center

https://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.management-center.doc/refs/rstmapping.htm?view=embed

Additional notes:

Purpose — Initial topics start at a high level and then drill down to specific developer customization methods for the store.

Adding pages to the WebSphere Commerce Storefront

- JSPs are called as the result of an action.
 - Actions return action forwards; those forwards map to JSP names.
 - To add a view, create a new JSP and make an entry in the Struts configuration files that map to that JSP.
 - To access that new view, map the return for an action to the new forward.
 - It can be done on a store-by-store basis.
 - Alternatively, a new command or new implementation of an existing command could be developed to return a new forward, mapped to the new JSP.
 - New JSPs often require the modification of access control policies.

```
<forward  
className="com.ibm.commerce.struts.ECActionForward"  
name="StoreView/10001"  
path="/index.jsp"/>
```

© Copyright IBM Corporation 2016

In order to develop a list of the pages that are needed to create your store, you need to know the business and functional requirements of the store. The list of pages also reflects any business processes that are defined.

Once business processes are available, you can create the shopping flow for your store. The shopping flow reflects the requirements and business processes that are defined for your store, illustrating how a customer moves through the store. This flow is in multiple design and implementation artifact deliverables for your development depending on your requirements. They could include:

- Use cases
- Test plan and cases
- Site map
- Flow diagrams
- Sequence diagrams

Next: JSP development in WebSphere Commerce

JSP development in WebSphere Commerce

- These tag libraries are used in JSP development for WebSphere Commerce:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wcbase" %>
```

- Optionally use tags for flow to enable feature administration in WebSphere Commerce.
- Because JSPs in WebSphere Commerce are feature-rich JSPs, it is possible to use almost any available tag library.
- When adding an entirely new page, it is often necessary to add many imports and includes, to add WebSphere Commerce features to the new JSP.
 - For example: `JSTLEnvironmentSetup.jspf`

© Copyright IBM Corporation 2016

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wcbase" %>
These tag libraries are commonly used in WebSphere Commerce JSPs since V5.6. It is a recommended best practice to use the wcbase library instead of the Java standard base:
<wcbase:useBean id="category"
classname="com.ibm.commerce.catalog.beans.CategoryDataBean"
scope="page" />
```

`fmt` — globalization and formatting:

```
<fmt:formatNumber var="totalPages" value="${(numEntries-
numEntries%pageSize)/pageSize+1}"/>
<fmt:parseNumber var="totalPages" value="${totalPages}"/>
<img src=<c:out value="${jspStoreImgDir}" />images/NoImageIcon.jpg"
alt="

```

`core` — for program flow control, if-then, loops, URL building, display output:

```
<c:choose>
  <c:when test="${!empty category.description.fullImage}">
    <img src=<c:out
value="${category.objectPath}${category.description.fullImage}"
" />" alt=<c:out
value="${category.description.shortDescription}" />">
  </c:when>
  <c:otherwise>
    <img src=<c:out
value="${jspStoreImgDir}" />images/NoImageIcon.jpg"
alt="

```

```
</c:choose>
<c:if test="${!empty category.description.shortDescription}">
    <c:out
value="${category.description.shortDescription}"
escapeXml="false"/></span>
</c:if>
```

Additional notes:

Details — Here are some simple examples of the use of the tag libraries within both V6 and V7 that remain largely unchanged.

Accessing request properties

- In a “standard” JSP, *param* and *paramValues* allow access to HTTP request parameters.
- Use Commerce-specific maps to access request parameters.
 - *WCParam* and *WCParamValues* allow access to decrypted HTTP request parameters.

```
<c:out value="${WCParam.catalogId}" />
```

© Copyright IBM Corporation 2016

Next: JSP error handling

JSP error handling

- JSP error handling can be at multiple levels with the store development.
- Intra-page error handling:
 - For detailed error handling, the page handles errors.
- Page level error handling:
 - By defining an error page, a JSP can delegate error-handling to another page.
- Application level error handling:
 - It is possible to define error pages by exception, when an exception of that type occurs, the request is forwarded to a particular JSP.

© Copyright IBM Corporation 2016

Error handling from within the page

For JSP pages that require more intricate error handling and recovery, a page can be written to directly handle errors from the data bean. Either the JSP page can catch exceptions that the data bean throws or it can check for error codes that are set within each data bean, depending on how the data bean was activated. The JSP page can then take an appropriate recovery action, based on the error received.

Error JSP page at the page level

A JSP page can specify its own default error JSP page from an exception that occurs within it through the JSP error tag. The error tag enables a JSP page to specify its own handling of an error. A JSP page that does not contain a JSP error tag has an error fall through to the application-level error JSP page. In the page-level error JSP page, it must call the JSP helper class (`com.ibm.server.JSPHelper`) to roll back the current transaction.

Here is an example of including error handling at the page level:

Create a single error JSP page that handles the errors that occur across all the other JSP pages in the application. To specify a JSP page as an errorHandler page, use this JSP page directive: `<%@ page isErrorPage="true" %>` In the errorHandler JSP page, use `ErrorDataBean` or `StoreErrorDataBean` to retrieve more information about the exception and display messages.

Include the errorHandler JSP page in other JSP pages, by using this JSP directive to specify that if exceptions occur on the current page, forward the request to `errorHandler.jsp:<%@ page isErrorPage="/errorHandler.jsp" %>`

Error JSP page at the application level

An application under WebSphere can specify a default error JSP page when an exception from within any of its servlets or JSP pages occurs. The application-level error JSP page can be used as a mall level or store level (for a single store model) error handler. In the application-level error JSP page, a call must be made to the servlet helper class to roll back the current transaction. The Web Controller is not in the execution path to roll back the transaction. Whenever possible, you must rely on the preceding two types of JSP error handling. Use the application-level error handling strategy only when required.

Include error handler at the application level by using the deployment descriptor (WEB-INF/web.xml) of the Web application. For example, if you want to handle a PaymentException generically at the application level, code a paymentError.jsp file and specify the same in the web.xml file by using the <error-page> tag. Here is an example:

```
<error-page>    <exception-type>PaymentException</exception-type>
<location>/paymentError.jsp</location> </error-page>
```

Instructor notes:

Purpose — JSP error handling

Details — A developer can use a combination of the preceding error handling scopes.

Additional information — When a JSP has errors and is invoked on the server environment, the return is a white page. It can happen with runtime errors, when a page references a null value, division by zero, parsing errors, or any other activity that results in an exception.

A simple technique that is useful is to wrap the page with a try and catch block, for example:

```
<% try { %>
...
JSP code
...
<% } catch (Exception e) { out.println(e); } %>
```

Normally you would not leave these statements in the deployed code. However, the statements can be useful if you have errors that are not apparent in the viewing and debugging of the page during development. The statements are also useful for runtime exceptions that occur in production that cannot be duplicated in the WebSphere Commerce development environment.

Globalization

- Globalization is the proper design and execution of systems, software, services, and procedures. For it to be globalized, one instance of software, executing on a single server or user machine, can process multilingual data and present culturally correct data in a multicultural environment such as the Internet.
- All business user tools support globalized usage.
- Catalog data in the storefront supports globalization.
- To support globalization, view components use NLS-based properties files.

© Copyright IBM Corporation 2016

The globalized catalog design in WebSphere Commerce provides the following features:

- **Product descriptions and attributes can be culturally specific.** The merchant can create a catalog that allows product descriptions and attributes to be culturally specific. This master catalog contains all the products that the merchant offers in all the language formats that the merchant supports. Not all products need to be available in all the supported language formats and all types of formatting.
- **Multiple descriptions or attributes for the same product and language:** This gives the ability to provide multiple descriptions or attributes for the same product in the same language. It allows stores that share products to offer different descriptions for the same product in the same language.

Locale-specific information:

Depending on the locales that the store supports, different features of the same products might need to be highlighted in different locales. Images might change with locales. Prices might vary and can be expressed in different currencies.

Additional notes:

Purpose — Discuss the globalization features inherent in WebSphere Commerce. Discuss the use of property files in WebSphere Commerce.

Properties file

storetext_en_US.properties

```
# Aurora store translation text

ENCODESTATEMENT= text/html;
charset=UTF-8

# Common store wide strings
SKU = SKU
PROMOTION_CODE_TITLE = PROMOTION CODE
INFORMATION
PROMOTION_CODE = Promotion code:
#
INDEX_TITLE = {0} - Welcome!
SELECT = Select
PROMOTION_CODES_REDEEMED = You have
entered promotion codes for the
following promotions:
```

storetext_de_DE.properties

```
# Aurora store translation text

ENCODESTATEMENT= text/html;
charset=UTF-8

# Common store wide strings
SKU = Artikelnummer
PROMOTION_CODE_TITLE = RABATTCODE -
INFORMATIONEN
PROMOTION_CODE = Rabattcode:
#
INDEX_TITLE = {0} - Willkommen!
SELECT = Auswählen
PROMOTION_CODES_REDEEMED = Sie haben
Rabattcodes für die folgenden
Rabatte eingegeben:
```

© Copyright IBM Corporation 2016

```
<WCDE>\workspace\Stores\WebContent\Aurora\include\JSTLEnvironmentSetup.jspf.
<%-- Load the store bundles --%>
<fmt:setLocale value="${CommandContext.locale}" />
<fmt:setBundle basename="${jspStoreDir}storetext" var="storeText" />
This bundle is then used to display the output in the TopCategoriesDisplay.jsp.
<html>
<head>
<title>
<fmt:message key="INDEX_TITLE" bundle="${storeText}">
<fmt:param value="${storeName}" />
</fmt:message>
</title>
<meta name="description" content=<c:out
value="${catalog.description.longDescription}" />>
<meta name="keyword" content=<c:out value="${storeCatalogKeyword}" />>
<link rel="stylesheet" href=<c:out
value="${jspStoreImgDir}${vfileStylesheet}" />" type="text/css"/>
</head>
```

Additional notes:

Purpose — Discuss the properties file implementation in WebSphere Commerce. Next: Using property files

Using property files

First, define the locale and load the proper store bundle:

```
<fmt:setLocale  
value="${CommandContext.locale}" />  
<fmt:setBundle  
basename="${jspStoreDir}storetext"  
var="storeText" />
```

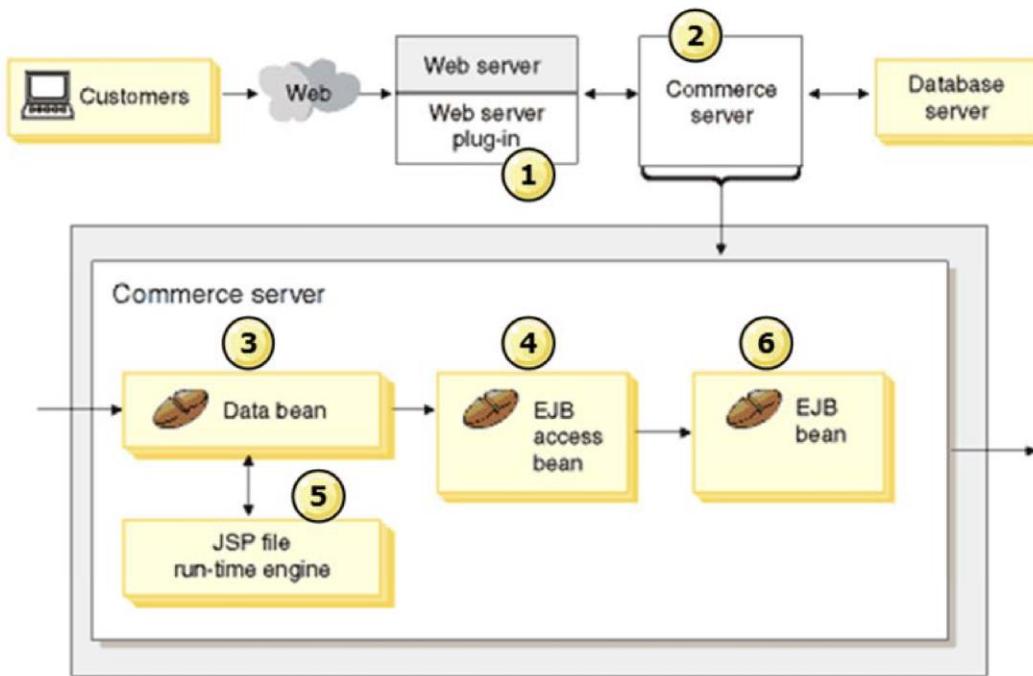
Then, when necessary, access the proper property:

```
<fmt:message key="INDEX_TITLE"  
bundle="${storeText}">  
    <fmt:param value="${storeName}" />  
</fmt:message>
```

© Copyright IBM Corporation 2016

Next: JavaServer Pages by using data

JavaServer Pages, using data



© Copyright IBM Corporation 2016

The WebSphere Commerce Server uses JSP pages in the following ways:

1. The web server plug-in parses HTTP requests, routes the requests to the WebSphere Commerce Server, and performs workload balancing functions.
2. The WebSphere Commerce Server processes the incoming request and invokes a JSP page as a response.
3. The data bean is placed in a JSP page.
4. The enterprise access bean converts an enterprise bean into a Java bean for use in a JSP page.
5. The JSP page runtime engine builds and returns an HTML page.
6. The enterprise bean provides database access.

Overview of the WebSphere Commerce Tag library JSTL

© Copyright IBM Corporation 2016

This section describes an overview of the WebSphere CommerceTag library JSTL.

WebSphere Commerce Tag library JSTL (1)

- Collection of JSP tags that provide standard functionality
- Supports
 - Conditions
 - Iteration
 - locale-sensitive formatting
 - expression language (EL) to control retrieval and display of data
- Two versions of JSTL
 - EL (expression language) based
 - RT (request-time expression) based
- EL based is used in WCS
- JSTL 1.0 consists of four functional areas
 - Core
 - XML processing
 - I18N (internationalization) capable formatting
 - Relational database access (SQL)

© Copyright IBM Corporation 2016

JSTL is collection of JSP tags that provide standard functionality most commonly sought by JSP page authors. JSTL Supports conditions, iteration, locale-sensitive formatting and expression language (EL) to control retrieval and display of data. Two versions of JSTL available are EL (expression language) based and RT (request-time expression) based. WebSphere commerce uses EL based. JSTL 1.0 consists of four functional areas, that are Core, XML processing, I18N (internationalization) capable formatting and Relational database access (SQL).

WebSphere Commerce Tag library JSTL (2)

- WebSphere Commerce Foundation Tag Library
 - A collection of custom tags that are provided to you to support the authoring of WebSphere Commerce JSP files.
 - WebSphere Commerce starter stores use the JavaServer Pages Standard Tag Library (JSTL) to perform logic, instead of Java code.
 - Business logic is moved into data beans and REST services.
 - The combination of these steps allows for less Java code in a JSP page.

© Copyright IBM Corporation 2016

https://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.component-services.doc/refs/rwvtaglib.htm

Websphere Commerce Data Beans

© Copyright IBM Corporation 2016

This section describes an overview of the WebSphere Commerce Data Beans.

WebSphere Commerce data beans

- Access information from the tables in the database
- Creating store pages using beans provide flexible way for frequent display changes of
 - catalog groups
 - product lists
 - product prices
- WCS data beans are grouped in to :
 - Access control
 - WebSphere Commerce Enterprise Approval
 - Catalog
 - Common
 - WebSphere Commerce Enterprise Contract
 - Coupon promotion, coupon redemption, coupon wallet
 - Fulfillment
 - Inventory
 - Message extension
 - Order

© Copyright IBM Corporation 2016

WebSphere Commerce data beans is used to access information from the tables in the database. Creating store pages using beans provide flexible way for frequent display changes of catalog groups , product lists or product prices. WebSphere Commerce data beans are grouped into the following areas : Access control, WebSphere Commerce Enterprise Approval, Catalog, Common, WebSphere Commerce Enterprise Contract, coupon promotion, coupon redemption, coupon wallet, Fulfillment, Inventory, Message extension, Order.

WebSphere Commerce Data beans (1 of 2)

- Data beans are Java bean objects that allow for easy data access in a JSP.
- In some cases, access beans wrap the entity EJB objects to provide a simple interface to the database.
- Three types:
 - **SmartDataBean:**
 - Extends access beans, fetches information from the database through that access bean, commonly through *lazy fetch*.
 - **CommandDataBean:**
 - Uses a command to populate data, gather all data at once.
 - **InputDataBean:**
 - Retrieves data from URL parameters or attributes.

© Copyright IBM Corporation 2016

A data bean is a Java bean that is used to provide dynamic data in JSP pages. There are two types of data beans: smart data beans and command data beans.

A smart data bean uses a **lazy fetch** method to retrieve its own data. This type of data bean can provide better performance in situations where not all data from the access bean is required, since it retrieves data only as required. Smart data beans that require access to the database must extend from the access bean for the corresponding entity bean and implement the **com.ibm.commerce.SmartDataBean** interface. For example, the **ProductDataBean** data bean extends the **ProductAccessBean** access bean, which corresponds to the **Product entity bean**. Some smart data beans do not require database access. For example, the **PropertyResource** smart data bean retrieves data from a Resource Bundle, rather than the database. When database access is not required, the smart data bean must extend the **SmartDataBeanImpl** class.

A command data bean relies on a command to retrieve its data and is a more lightweight data bean. The command retrieves all attributes for the data bean at once, regardless of whether the JSP page requires them. As a result, for JSP pages that use only a selection of attributes from the data bean, a command data bean might be expensive in terms of performance time.

While access control can be enforced on a data bean level when using the smart data bean, it is not true for command data bean. Use a command data bean only if using a smart data bean is impractical.

Command data beans can also extend from their corresponding access beans and implement the **com.ibm.commerce.CommandDataBean** interface.

A data bean that implements the **InputDataBean** interface retrieves data from the URL parameters or attributes that the view sets.

Attributes that are defined in this interface can be used as primary key fields to fetch additional data. When a JSP page is invoked, the generated JSP servlet code populates all the attributes that match the URL parameters. It then activates the data bean by passing the data bean to the data bean manager. The data bean manager then invokes the **setRequestProperties()** method of the data bean

(as defined by the **com.ibm.commerce.InputDataBean** interface) to pass all the attributes that the view sets.

Additional notes:

Purpose — Describe the WebSphere Commerce data beans.

Details — Note the data bean cross-references under the Reference section of the information center.

Additional information — *Cross-reference URLs:*

- *Data beans — EJB beans — tables*

http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rgx_ejb.htm

- *Commands — beans — tables*

http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rgx_ref.htm

Next: WebSphere Commerce data beans (2 of 2)

WebSphere Commerce Data beans (2 of 2)

- For a SmartDataBean, all changes to entity beans are reflected in the access beans and data beans immediately upon regenerating them in the Rational tool.
- Data beans are added to a page using the `wcbase:usebean` tag.
 - This tag is an extension of the `jsp:usebean` tag, and includes logic to “activate” the bean, causing it to populate itself with data.
- There is an extensive reference of Data beans, Access beans, and EJB beans in WebSphere Commerce Information Center.

© Copyright IBM Corporation 2016

A data bean normally extends an access bean. The access bean, which the Rational Application Developer can generate, provides a simple way to access information from an entity bean. When modifications are made to an entity bean (for example, adding a field, a new business method, or a new finder), the update is reflected in the access bean as soon as the access bean is generated again. Since the data bean extends the access bean, it automatically inherits the new attributes. As a result of this relationship, no coding is required to enable the data bean to use new attributes from the entity bean.

If you need to add new attributes to a data bean that are not derived from an entity bean, you can extend the existing data bean using Java inheritance. For example, if you want to add a field to the `OrderDataBean`, define `MyOrderDataBean` as follows:

```
public class MyOrderDataBean extends
    OrderDataBean { public String
    myNewField () {
        // implement the new field here
    }
}
```

Data bean example

```
<wbase:useBean id="catalog"
  classname="com.ibm.commerce.catalog.beans.CatalogDataBean"
  scope="page" />
```

Then, later in the page:

```
<c:forEach var="topCategory"
  items="${catalog.topCategories}" varStatus="counter">
  <c:set
    var="someCategoryIDs"
    value="${someCategoryIDs}, ${topCategory.categoryId}" />
</c:forEach>
```

© Copyright IBM Corporation 2016

CatalogDataBean is one of hundreds of data beans that are available for use. Each data bean is related to a WebSphere Commerce subsystem, process, or function. Some others include: ProductCompareDataBean, OrderDataBean, PaymentMethodInputDataBean, and GiftRegistryDataBean (if IBM Gift Center for WebSphere Commerce is installed).

Next: Writing a new data bean

Writing a new Data bean

- Custom data beans are stored in the WebSphereCommerceServerExtensionsLogic project.
- Create a class, inheriting from the proper type, based on the Data bean you need to create.
 - Smart Data beans subclass the appropriate Access bean and implement SmartDataBean.
 - Command Data beans implement the CommandDataBean interface.
 - Input Data beans implement the InputDataBean interface.

© Copyright IBM Corporation 2016

Checkpoint

1. True or false: There is one Struts action for every URL request parameter.
2. List the three types of data beans that might be used on a storefront JSP and their matching interfaces.

© Copyright IBM Corporation 2016

Checkpoint solutions

1. False. There is only one BaseAction.
2. InputDataBean, CommandDataBean, and SmartDataBean (types and interfaces).

© Copyright IBM Corporation 2016

Web 2.0

© Copyright IBM Corporation 2016

Web 2.0 store architecture

After completing this topic, you should be able to:

- Understand the value of Web 2.0 on storefronts.
- Identify how Web 2.0 is used in the WebSphere Commerce presentation layer.

© Copyright IBM Corporation 2016

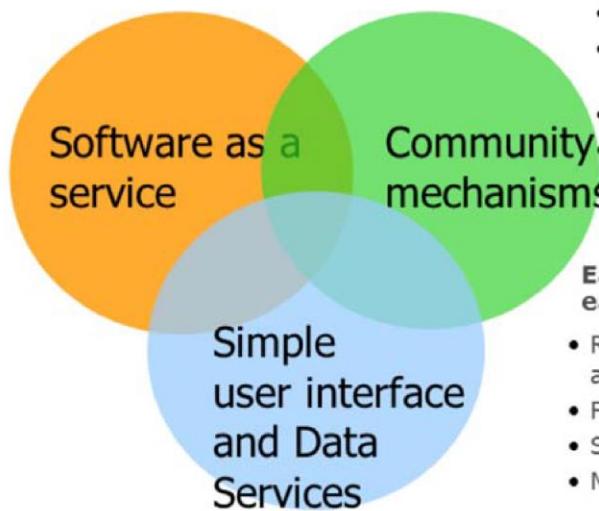
Next: What is Web 2.0?

What is Web 2.0?

- What does Web 2.0 mean for a web application?
 - Community intelligence instead of individual authority.
 - Rich and responsive but simple user interfaces.
 - Software as a service instead of a product.

Service, not software

- User-driven adoption
- Value on demand
- Low cost of entry
- Public infrastructure
- Tight feedback loop between providers and consumers



Users add value

- Recommendations
- Social networking features
- Tagging
- User comments
- Community rights management

Easy to use, easy to remix

- Rich Internet applications (AJAX)
- Feeds (Atom, RSS)
- Simple extensions
- Mashups (REST APIs)

© Copyright IBM Corporation 2016

Additional notes:

Purpose —

Details — REST stands for "Representational State Transfer". It is a style of client/server software architecture, that is geared with Web 2.0 in mind, which exchange representations of resources, where a resource is any type of meaningful artifact, and its representation is a document that captures its essence or state.

Next: Developing the Web 2.0 channel

Developing the Web 2.0 channel

- Web 2.0 channel offers new, exciting functionality to enhance user experience.
- Enabled by using widgets from Dojo Foundation toolkit, which leverages AJAX technology.
 - Widget “hot spots” are identified on the page.
 - Web 2.0 widgets (Dojo “Dijits”) offer interfaces.
 - WebSphere Commerce offers custom widgets to function with its frameworks.



© Copyright IBM Corporation 2016

Dojo Toolkit is an open source modular JavaScript library, or specifically, a JavaScript toolkit, which was designed to ease the rapid development of cross-platform applications and websites. WebSphere Commerce V8 uses the Dojo Foundation Toolkit V1.8.7 and higher Dojo widgets (commonly called "Dijits") that are made up of JavaScript, HTML markup, and CSS style declarations that provide many interactive features.

AJAX stands for Asynchronous Java and XML.

Next: Web 2.0 store in WebSphere Commerce

Web 2.0 store in WebSphere Commerce

- Starter Stores as reference applications:
 - Aurora B2C
 - Aurora B2B
 - As of V8, both Aurora B2B & B2C are only supported within extended sites
- Reference applications provide:
 - Starter stores and Catalog Store archives
 - User interface widget files
 - Enhanced user interfaces
 - Modularize the store pages (Aurora)
- Reference application that is implemented by using AJAX functionality.

© Copyright IBM Corporation 2016

Dojo

© Copyright IBM Corporation 2016

This section describes an overview of Dojo.

Dojo introduction

After completing this topic, you should be able to:

- Explain the features in the Dojo framework.
- Create user-centric JSPs by using Dijits.
- Leverage Dojo's event and XHR (XML HTTP requests) systems in an interactive Storefront.

© Copyright IBM Corporation 2016

Next: Dojo Foundation Toolkit

Dojo Foundation Toolkit

- Open source collection of JavaScript libraries
 - Drag and drop behavior
 - Standard widgets for buttons, menus, dialogs, and so on.
 - Event handling
 - AJAX request processing
- JavaScript data structures
- Aurora store uses Dojo toolkit:
 - WCS 8 uses dojo 1.8
- For more information and downloads:
<http://www.dojotoolkit.org/>

© Copyright IBM Corporation 2016

Dojo Mobile, leverages CSS 3 for scrolling, swiping animations, and transitions.

Next: Dijits

Dijits

- Dijits (Dojo widgets) are reusable, modular user interface elements.
 - A Dijit's full representation is substituted into the page, creating its own tree of DOM nodes and its own JavaScript object.
 - Dijits come with a template HTML and a template CSS file, which are used at creation time. These files are made available by your application to be changed as necessary to match your layout.
- There are more than 100 Dijits
 - Button, chart, accordion, clock, datepicker, dialog, form, menu bar, slider, taskbar, and many others.
 - Incorporating existing Dijits into an existing site's interface is a simple process.
 - It is possible to write additional Dijits, if necessary.

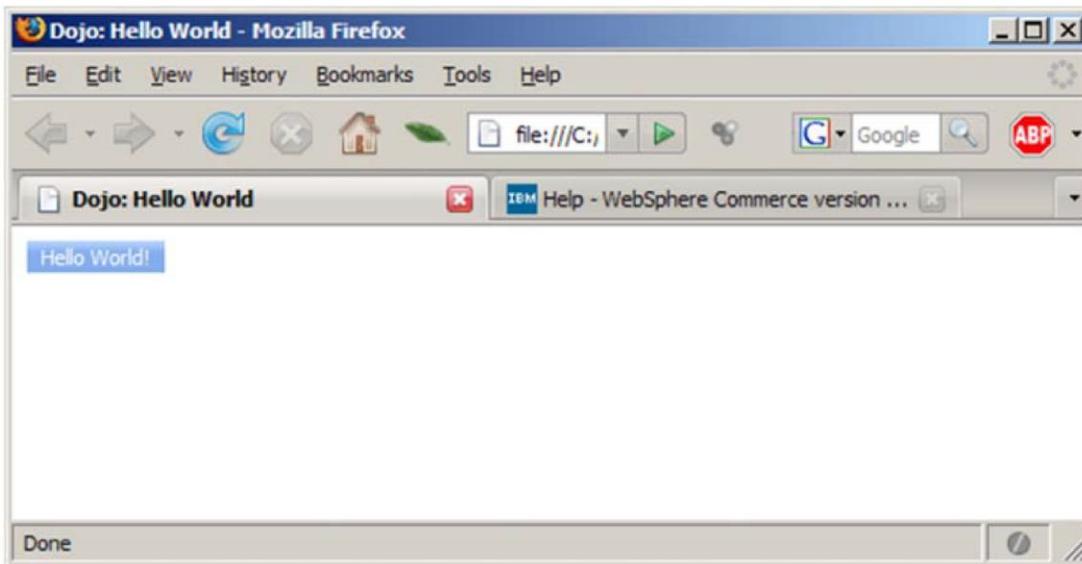
© Copyright IBM Corporation 2016

Dijits are really "macros." The Dijit's full representation on the page that the user's browser parses replaces the simple tag that places one on a page.

Next: Dijit example

Dijit example

```
<button dojoType="button" id="helloButton">Hello World!</button>
```



© Copyright IBM Corporation 2016

In the browser, the following code replaces this button element:

```
<div class="dojoButton" dojoattachevent="onMouseOver; onMouseOut;
onMouseDown; onMouseUp; onClick:buttonClick; onKey:onKey; onFocus;" style="position: relative; height: 17px; width: 73.3333px;" dojoattachpoint="buttonNode" tabindex="0" disabled="false">
    <div class="dojoButtonContents" align="center" style="position: absolute; z-index: 2; -moz-user-select: none; left: 5.66667px;" dojoattachpoint="containerNode">Hello World!
</div>

    
    
</div>
```

Additional notes:

Details — This code presupposes that the required Dojo package is included in the JSP:

```
<script type="text/javascript" src="/wcsstore/dojo18/dojo/dojo.js"/>
```

Additional information — As an instructor, you can demonstrate this example to students.

1. Create a new dynamic web project. (Right-click in the IDE and select **New > Other > Web > Dynamic Web Project**. Click **Next**.)
2. Give the project an arbitrary name, such as **DojoTest**, accept the remaining defaults, and then select **Finish**.
3. Right-click the new web project, and then select **New > Web Page**. Select **JSP** as the template, give the page another arbitrary name, and select **Finish**.
4. In the JSP, add the code to include the Dojo toolkit:

```
<script type="text/javascript" src="/wcsstore/dojo18/dojo/dojo.js"/>
```
5. Now add the button, as shown on the slide.

```
<button dojoType="button" id="helloButton">Hello World!</button>
```
6. Save the JSP.
7. Right-click the JSP and select **Run As > Run on Server**. Accept the defaults and select **Finish**.

Dijit parsing

- Dijits are parsed on page load, performing the “substitution” and any required DOM manipulation.
- Dojo must parse any new Dijits that are created as a result of new HTML from an AJAX request.

- Example:

```
dojo.require("dijit._Widget");
var testObjects =
    dojox.xml.parser.parse(document.getElementById("Cart"));
var cartWidget = create(testObjects);
```

© Copyright IBM Corporation 2016

Additional notes:

The **Document Object Model (DOM)** is a cross-platform and language- independent convention for representing and interacting with objects in HTML, XHTML, or XML documents. Aspects of the DOM (such as its "Elements") might be addressed and manipulated within the syntax of the programming language in use. The public interface of a DOM is specified in its API.

Some students might wonder why the path name is listed as "dojox." DojoX is a subfamily of the Dojo toolkit. There are four such subfamilies: djConfig, DojoX, Dijit, and the core Dojo. From <http://www.dojotoolkit.org/>

"DojoX is a collection of subprojects that is provided by Dojo committers and subject to the generous licensing and policies of the Dojo CLA. Each subproject in DojoX has its own top-level directory and a README file with status information and project status and a stability rating (experimental, beta, stable). Projects might depend on other top-level Dojo projects, like Dojo or Dijit. Unlike Dojo and Dijit, code is not subject to i18n and a11y restrictions and might vary in quality (experimental code is encouraged in DojoX, but currently prohibited in Dojo and Dijit). DojoX projects may mature to a stable state and stay in DojoX, or on occasion after proving themselves might migrate to Dojo Core or Dijit. Dojo and Dijit projects are constrained both by development resources as well as design goals, so DojoX is a natural place to provide enhanced behavior or extend Dojo Core or Dijit primitives. DojoX can also be an incubator for entirely new projects."

Next: Dojo XHR

Dojo XHR

- Abstracts AJAX-style requests
 - XHR: XMLHTTP request objects
- Basic building block for constructing responsive AJAX interactions.
- For example,

```
var xhrArgs = {
    url: "OrderItemAdd",
    load: function(type, data, evt) {
        /*do something with the data */
    },
    handleAs: "text/json"
    error: function(error) { /* handle error */ }
}
var response = dojo.xhrGet(xhrArgs);
```

© Copyright IBM Corporation 2016

Processing XMLHttpRequest is especially important in a Web 2.0 scenario, since the requests are handled by using this servlet, as opposed to the traditional XMLHttpRequest.

Several Dojo functions allow you to participate with XMLHttpRequest objects (XHR), including:

- dojo.xhrGet: use HTTP GET method to make an XHR call
- dojo.xhrPost: use HTTP POST method to make an XHR call
- dojo.xhrDelete: use HTTP DELETE method to make an XHR call
- dojo.rawXhrPut: use HTTP PUT method to make an XHR call
- dojo.formToJson: convert a DOM form to JSON
- dojo.formToObject: convert a DOM form to a JavaScript object
- dojo.formToQuery: convert a DOM form to a query string

Next: Dojo event system

Dojo event system

- Dojo abstracts the JavaScript events system, making it much more broad in scope.
 - Using objects and functions, a Developer can register to “listen” to any function call and connect functions that will be run before, after, or around that call:
 - dojo.connect()
 - dojo.addOnLoad()
 - dojo.subscribe()
 - dojo.publish()
 - It is possible to connect multiple events; those events are called in the order they are registered.
 - For example, calling dojo.addOnLoad multiple times will result in EACH function being executed after the onLoad function.

```
dojo.connect(dojo.byId("id"), "onclick", listenerObj, "handleOnClick");
dojo.addOnLoad(initFunction);
```

© Copyright IBM Corporation 2016

WebSphere Commerce user interface widgets

- Scrollable pane (wc.widget.ScrollablePane)
 - Scrolls through product thumbnails and product details.
- Tooltip (wc.widget.Tooltip)
 - Displays hints.
- Commerce dialog (wc.widget.WCDialog)
 - Opens the shopping cart contents when hovering.
- RefreshArea (wc.widget.RefreshArea)
 - Identifies a page section that is dynamically updated with new content.
 - Requires an AJAX refresh controller

© Copyright IBM Corporation 2016

wc.widget.ScrollablePane

The ScrollablePane widget displays a scrolling thumbnail picker that displays product images and details. It provides a scrolling effect for the items or images that are within the content of the widget. Each item within the scrollable pane should be in a ContentPane widget. The scrollable pane supports both automatic and manual scrolling of the items that are displayed in the widget.

wc.widget.Tooltip

The Tooltip widget is an extension of dijit.Tooltip and is used to display tooltips that match the overall style of the Madisons starter store. It inherits all the functionality from dijit.Tooltip, but does not have the default dijit.Tooltip styles.

wc.widget.WCDialog

The WCDialog widget is an extension of dijit.Dialog and is used to display the mini shopping cart on the header. The mini shopping cart dialog opens when the mouse hovers over the dialog area, and closes when the mouse leaves the dialog area.

wc.widget.RefreshArea

The RefreshArea widget is used to wrap a Document Object Model

(DOM) node that is refreshed by replacing the innerHTML property with new content loaded from the server. That is, a refresh area is a section of a page that is dynamically updated with new content. A refresh area widget is associated with a registered refresh controller that handles listening for events that require this widget to be refreshed.

Next: AJAX introduction

AJAX

© Copyright IBM Corporation 2016

This section describes an overview of AJAX.

AJAX introduction

After completing this topic, you should be able to:

- Explain the Asynchronous Java and XML (AJAX) framework.
- Compare the benefits of AJAX in a Web 2.0 Storefront.
- Describe the WebSphere Commerce presentation layer architecture by using AJAX.
- Build AJAX into a WebSphere Commerce storefront by using WebSphere Commerce tools and Dojo.

© Copyright IBM Corporation 2016

Next: Asynchronous Java and XML (AJAX)

Asynchronous Java and XML (AJAX)

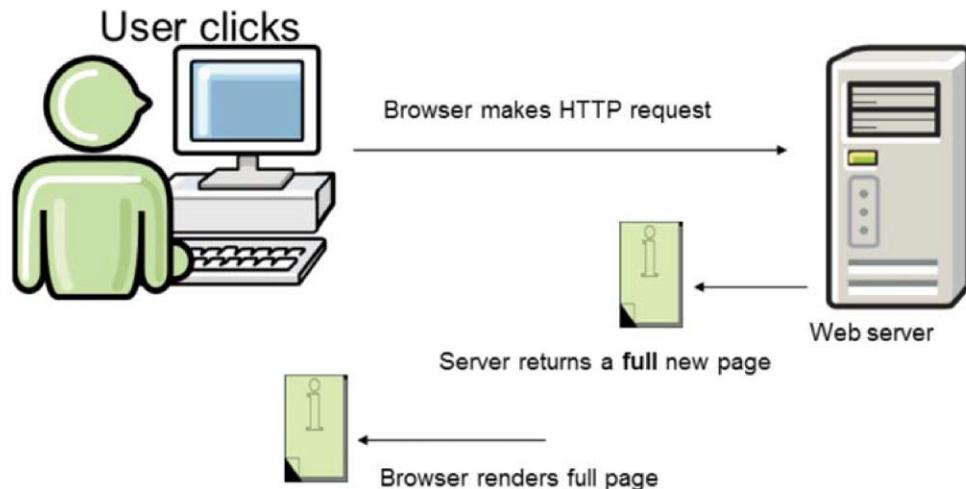
- Several Web development technologies are used to create interactive web applications:
 - XHTML and CSS for standardized presentation.
 - Document Object Model (DOM) for dynamic interaction.
 - XML and XSL for data handling.
 - XMLHttpRequest for asynchronous data requests.
 - JavaScript connects it all.
- AJAX enables rich web application interfaces.
 - Uses JavaScript to make requests and merge results into an existing page.

© Copyright IBM Corporation 2016

Using AJAX, web applications send and receive data from the server asynchronously in the background without interfering with the display or behavior of the page. This process, in turn, makes the client-side page more dynamic and interactive. Despite the name, however, requests do not have to be asynchronous: this name is to stress the differences between the "traditional" web model and web model inAJAX.

Next: Traditional Web application interaction

Traditional web application interaction

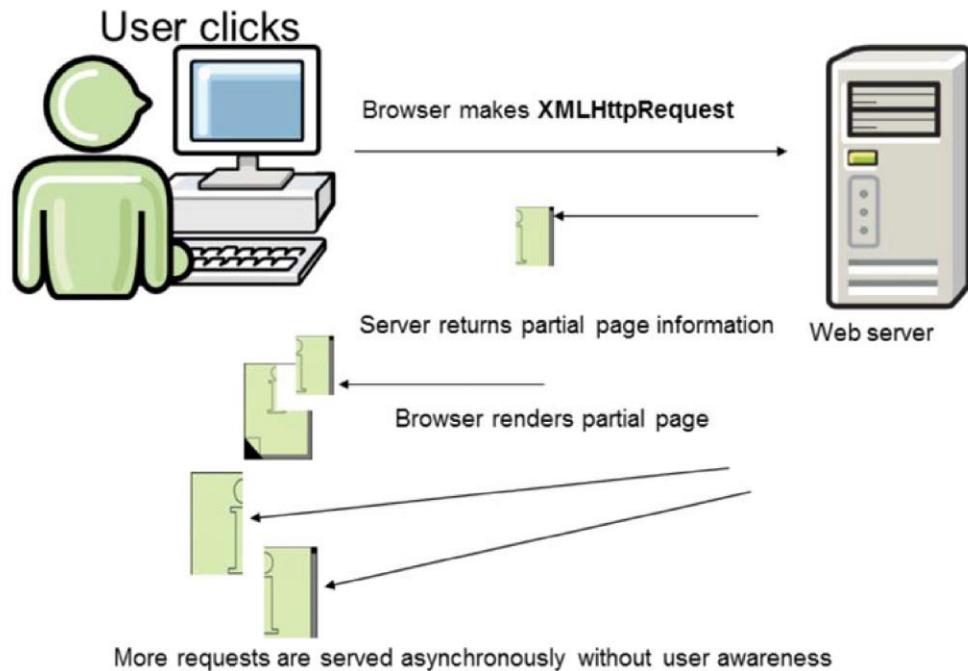


© Copyright IBM Corporation 2016

In a traditional web model, user actions from the client trigger HTTP requests to the server. The server processes the request, and then returns a response. Such is the case with the Web 1.0 channel: a request passes through the presentation layer (which includes all the Struts mapping) to the server-side business logic facade, which processes the request in the business logic and persistence layers, and then returns a response: a JSP to render as HTML on the client.

Next: AJAX web application interaction

AJAX web application interaction



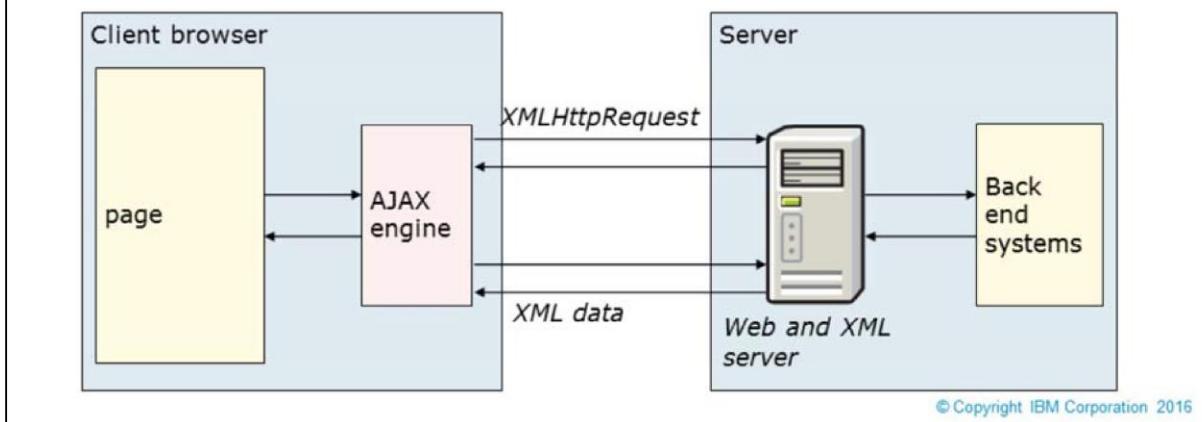
© Copyright IBM Corporation 2016

In a Web 2.0 (or AJAX) model, however, this "back-and-forth" to which web users are accustomed to, is more interactive. An AJAX engine is introduced as an intermediary on the client. This engine is written in JavaScript and gets loaded at the start of the session, and manages the page rendering and server communication. It can communicate with the server asynchronously, and control page elements so that only pieces of the page are updated instead of whole pages. The result is several smaller communiqués instead of the customary "all-or-nothing, back-and-forth" one discovers in a traditional web environment.

Next: AJAX model and benefits

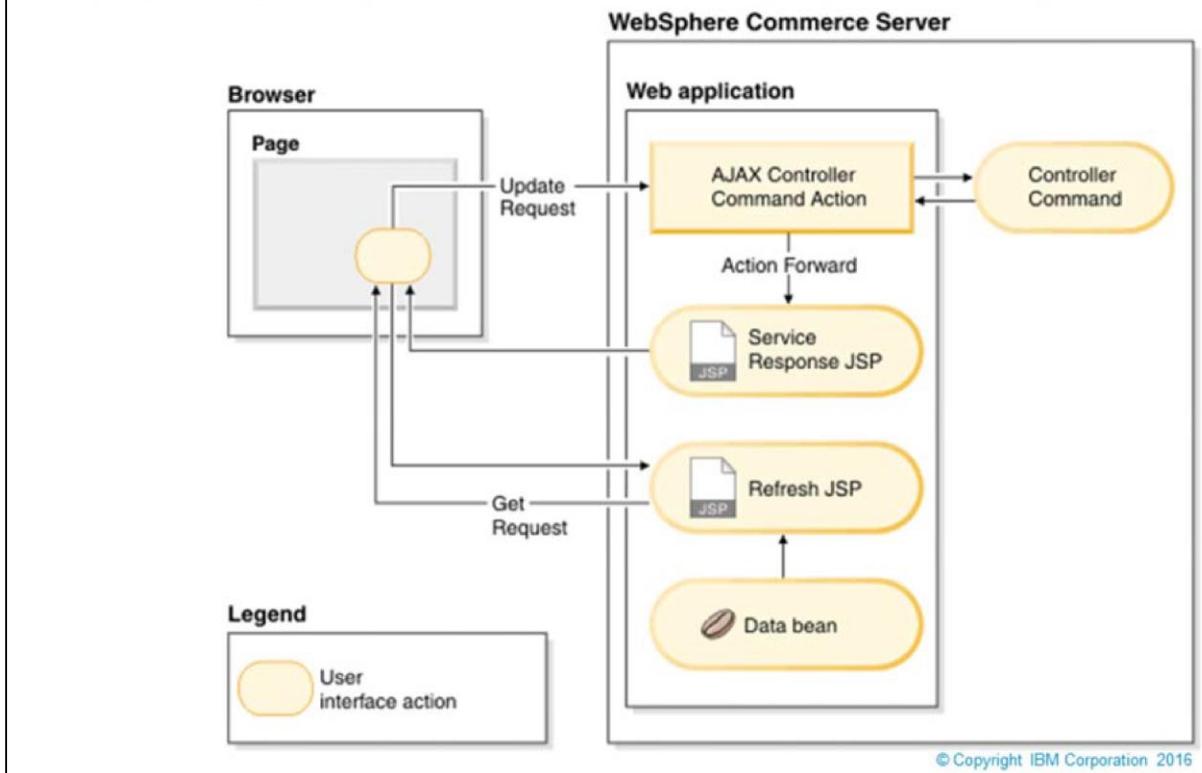
AJAX model and its benefits

- Benefits:
 - Requests are made for data or sections of pages, not for entire pages.
 - Application can appear to be more interactive.
 - Server requests are shorter, smaller.
 - Server requests performed asynchronously in background.
- Disadvantages
 - Overused, AJAX features get “chatty”.
 - Additional complexity in development and debugging.
 - Dynamic pages cannot be bookmarked or retrieved from browser history.
 - Heavy use of JavaScript difficult for crawlers to index pages.



Next: AJAX and WebSphere Commerce controller commands

AJAX and WebSphere Commerce controller commands



Calling a controller command:

- Controller command must be "AJAX enabled" in Struts configuration:
 - Actions must call `com.ibm.commerce.struts.AjaxAction` instead of `BaseAction`.
 - Runtime automatically forwards the request to a special view.
 - `AjaxActionResponse.jsp`
 - `AjaxActionErrorResponse.jsp`
 - The "view" converts response properties into a JavaScript object notification (JSON), an event for the client to consume.

Before the client is able to call a WebSphere Commerce controller command or service by using AJAX, the WebSphere Commerce server must define that the controller command or service can be called by using AJAX (rather than the traditional programming model where the request is made and WebSphere Commerce runtime implies a new redirect).

To define a controller command or service to be available for AJAX type requests, simply define a new struts-action entry in the struts-config XML file that identifies the controller command or service as an AJAX type of action. For example, to create a new struts-action for the `InterestItemAdd`

controller command that can be called by using AJAX, it must be defined in the struts configuration XML file.

Additional notes:

Similarly, REST services must be AJAX enabled by using `AjaxRESTAction`, instead of `RESTAction`.

Next: Declaring AJAX-enabled controller commands

Declaring AJAX-enabled controller commands

- struts-config-ext.xml:

```
<action  
    parameter="com.mycompany.commands.InterestItemAddCmd"  
    path="/AjaxInterestItemAdd"  
    type="com.ibm.commerce.struts.AjaxAction">  
  
    <set-property property="authenticate" value="0:0"/>  
    <set-property property="https" value="0:1"/>  
</action>
```

© Copyright IBM Corporation 2016

There are two relevant items in this example. The first is the parameter. The parameter, typically, is the path and name of the controller command interface. Nothing indicates that this controller command is any different from a controller command that was used with the Web 1.0 channel. The difference is in the generic action, AjaxAction. This action processes the request differently than BaseAction would, but the execution of the business logic is nonetheless identical.

Notice the path (/AjaxInterestItemAdd). The Ajax- prefix here is not necessary, since it functions only as a key to the URL request, but it is recommended that AJAX-enabled controller commands are prefixed in this way in order to make debugging easier.

After this response is received, the Dojo-based tags will handle any refreshes necessary, as you will learn.

Handling responses

- Response that is passed to common JSPs, which generate a JSON (JavaScript Object Notification)
- A client browser that uses the JavaScript must consume JSON:

```
wc.service.declare({  
    id: "AjaxInterestItemAdd",  
    actionId: " AjaxInterestItemAdd",  
    url: " AjaxInterestItemAdd",  
    formId: "",  
    successHandler:  
        function(serviceResponse) { alert("success"); },  
    failureHandler:  
        function(serviceResponse) {  
            if (serviceResponse.errorMessage) {  
                alert(serviceResponse.errorMessage);  
            }  
        }  
    } );
```

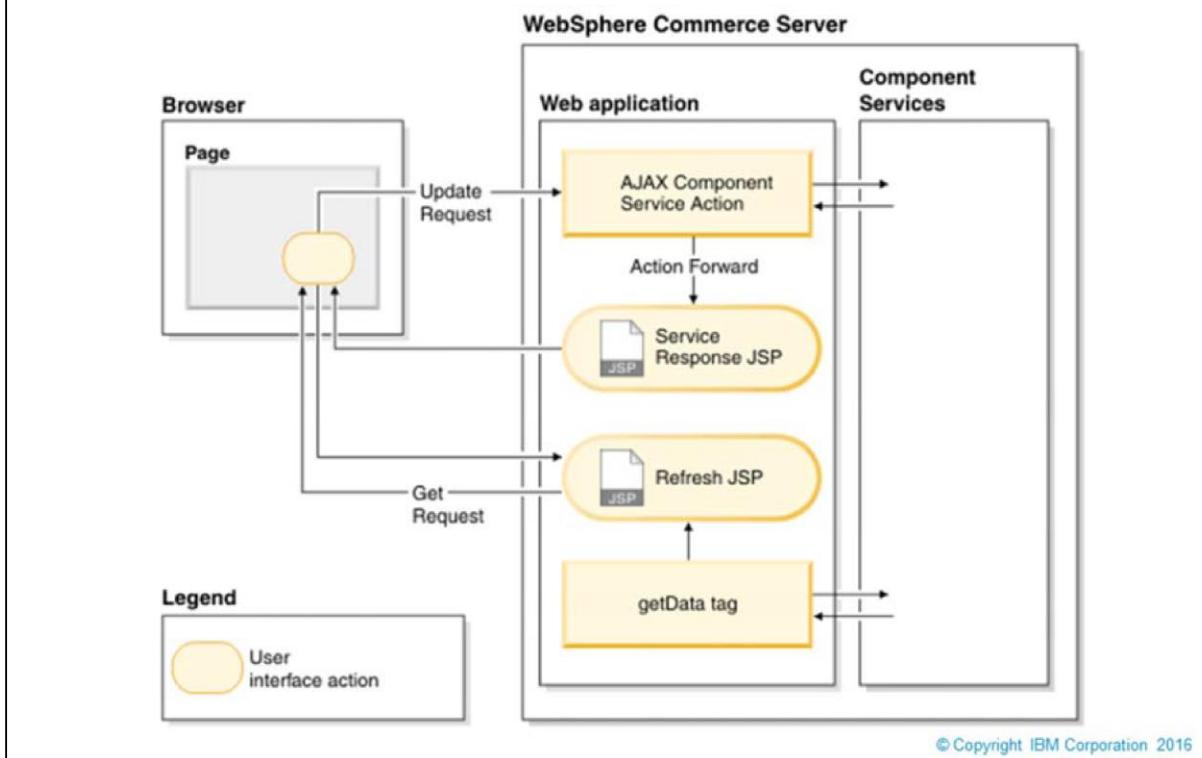
© Copyright IBM Corporation 2016

It is important to remember that the service must be declared in the JSP beforehand. Compare this service declaration to the Struts Action definition on the preceding page. Notice that the actionId is identical.

Remember, that when using the AjaxAction, only one of two possible JSPs are returned: AjaxActionResponse.jsp or AjaxActionErrorResponse.jsp. Recall that these two JSPs generate the JSON that is consumed (or handled) by this service declaration.

Next: AJAX and WebSphere Commerce BOD processing

AJAX and WebSphere Commerce BOD processing



- Calling process BOD commands:

- Almost identical to controller command processing.

- Different Struts action:

- Actions must call AjaxComponentServiceAction.

- Passed parameter includes the service name, and BOD noun and verb instead of the controller command path.

- For example, order.addOrderItem

- Return values are the same two JSPs.

`AjaxActionResponse.jsp AjaxActionErrorResponse.jsp`

- The full path is actually `com.ibm.commerce.struts.AjaxComponentServiceAction`

Next: Declaring AJAX-enabled BOD commands

Declaring AJAX-enabled BOD commands

- struts-config-ext.xml:

```
<action  
    parameter="order.addOrderItem"  
    path="/AjaxOrderChangeServiceItemAdd"  
    type="com.ibm.commerce.struts.AjaxComponentServiceAction">  
  
    <set-property property="authenticate" value="0:0"/>  
    <set-property property="https" value="0:1"/>  
</action>  
• What's changed?  
    • Parameter name calls the service (order) and BOD (addOrderItem).  
    • Action class (AjaxComponentServiceAction), which includes logic to:  
        - Transform BOD into logical object (Java SDO).  
        - Call the Data Service Layer.
```

© Copyright IBM Corporation 2016

There are two items of relevance in this example. The first is the parameter. Instead of using the fully qualified name of the controller command interface, you are calling the service and BOD. The second difference is the action class. Recall that the BOD processing of commands uses standard business objects (BODs) which need to be transformed from their XML representation into standard Java objects, called Service Data Objects (SDOs). This modified Struts action includes the logic to transform BODs into SDOs, and to call the data service facade in the Data Service Layer to communicate with the persistence layer.

After this response is received, the Dojo-based tags will handle any refreshes necessary.

Next: Using AJAX for asynchronous calls

Declaring AJAX-enabled REST services

- struts-config-ext.xml:

```
<action parameter="wishlist.update"
path="/AjaxRestWishListUpdate"
type="com.ibm.commerce.struts.AjaxRESTAction">
    <set-property property="authenticate"
value="10101:0,10151:0"/>
    <set-property property="https"
value="10101:1,10151:1"/>
    <set-property property="csrfProtected"
value="10101:1,10151:1"/>
</action>
```

© Copyright IBM Corporation 2016

The parameter identifies the REST service. The first item (before the ".") is the resource name, while the second item (after the ".") is the resource method. The REST service path and method are configured in the component specific rest-template-config.xml under Stores\WebContent\WEB-INF\config\com.ibm.commerce.<component_name>, for example com.ibm.commerce.order.

Using AJAX for asynchronous calls

The screenshot shows a shopping website with a navigation bar at the top. A callout box labeled "Refresh area: shopping cart" points to the bottom right corner of the page, where a small "refresh" icon is located. Another callout box labeled "Refresh area: marketing spot" points to a promotional banner for "COLORS OF SUMMER" featuring a woman in a blue dress.

- Server calls are made asynchronously, often unknown to user for several purposes:
 - Keep shopping cart up-to-date with display total.
 - Update marketing spots to target consumer based on items that are viewed or shopping cart contents.
 - Change product viewers to target consumers or offer specials.
 - Alert consumer to online status of customer service representatives.
 - Provide live chat features.
- Identify areas that need to be refreshed.
 - Uses `wc.widget.RefreshArea`

© Copyright IBM Corporation 2016

Recall that one of the major advantages of a Web 2.0-enabled presentation layer solution is that only pieces of the page are updated instead of sending whole pages back and forth. As service responses are received in the client, and the JSON is processed, this might trigger other server calls or refreshes, which are often transparent to the user. These calls can perform a great number of useful, interactive functions:

- Keeping the shopping cart total up-to-date with the most recent items added
- Updating e-marketing and content display spots in order to target consumers based on products or categories that are viewed, or contents of their shopping cart
- Changing the contents of scrollable pane widgets in order to target consumers based on history

In some new generation shopping sites, companies offer "live shopping experiences," so that consumers can chat live with a customer service representative (CSR, or sometimes a "personal shopping expert"). Web

2.0 is versatile enough that a company can provide chat features on their site without adding new software. Some asynchronous calls might include:

- Refreshing the chat area regularly with dialog between user and CSR
- Providing the user with the status of the CSR (online, available, offline, occupied, and so on)

Additional notes:

Some of the more popular social networking sites offer live chat that is based on Web 2.0 technology without the installation of additional client-side software. The key to maximizing the use of AJAX asynchronous calls is to declare `refreshAreas` and `refreshControllers`, which is discussed in the following topic.

Next: WebSphere Commerce AJAX libraries

WebSphere Commerce AJAX libraries

- Refresh controller (`wc.render.declareRefreshController`)
 - Every refresh area requires the controller to control widget refresh, listen to changes in the context, and decide whether to update refresh areas.
 - `wc.render.getRefreshControllerById`
- Service declaration (`wc.service.declare`)
 - Declares an AJAX service to perform some server process.
 - `wc.service.getServiceById`
- Service invocation (`wc.service.invoke`)
 - Invokes a declared service
- Context declaration (`wc.renderdeclareContext`)
 - Defines a new set of client-side information and initializes it with properties, which can help decide whether a refreshArea needs updating.
 - `wc.render.getContextById`
 - `wc.render.updateContext`

© Copyright IBM Corporation 2016

wc.render.declareRefreshController(initProperties)

The `wc.render.declareRefreshController` function declares a new refresh controller and initializes it with the specified initialization properties. The initialization properties are in addition to the new refresh controller's properties. A refresh controller controls refresh area widgets, and listens to changes in the render context and model and decides whether the registered refresh areas should be updated.

wc.render.getRefreshControllerById(id)

The `wc.render.getRefreshControllerById` function returns the refresh controller declared by the specified identifier. If the refresh controller is not declared, the function returns *undefined*.

wc.service.declare(initProperties)

The `wc.service.declare` function declares a new AJAX service with the specified ID. In WebSphere Commerce, a service is a server URL that performs a server object create, update, delete, or other server processing. When needed, a WebSphere Commerce server URL is called by using AJAX in the JavaScript code. When the service completes successfully, a JSON object that contains all the response properties of the URLrequest is returned to successHandler (the success response) or failureHandler (the failure response) defined JavaScript function. In addition, a model changed event is sent to any subscribed listeners if the service completes successfully. In this case, the modelChanged and modelChanged/*actionId* Dojo events are published.

wc.service.getServiceById(id)

The `wc.service.getServiceById` function gets the declared service by using the specified identifier. If the service is not yet declared, the function returns undefined. This function is helpful when finding a service and updating its parameters before invoking it.

wc.service.invoke(serviceId, parameters)

The wc.service.invoke function finds the registered service with the specified service ID and invokes the service via AJAX by using the specified parameters.

wc.renderdeclareContext(id, properties, updateContextURL)

The wc.renderdeclareContext function declares a new render context and initializes it with the specified render context properties. The update context URL reports render context changes to the server. A render context is a set of client-side context information that tracks information about a page. The context information helps decide whether changes to refresh areas are needed. Changes to the render context are communicated to the server by using an AJAX-style request if an updateContextURL is provided. Once the render context updates successfully, an event is published, and any listening refresh controllers can decide to refresh their content to reflect the updated render context.

wc.render.getContextById(id)

The wc.render.getContextById function returns the render context declared by the specified identifier. If the render context is not declared, the function returns undefined.

wc.render.updateContext(id, updates)

The wc.render.updateContext function retrieves the render context with the specified ID and applies the updates that are found in the specified updates object.

Next: Using AJAX refresh areas on a page

Using AJAX refresh areas on a page

1. Identify AJAX services to invoke:
 - Functions for which you want asynchronous page changes to occur.
 - Use `wc.service.declare`
2. Define refresh controllers to manage updates:
 - Controllers that trigger the updates.
 - Use `wc.renderdeclareRefreshController`
3. Identify areas that need to be updated:
 - The areas that are updated.
 - Use `wc.widget.RefreshArea` to identify.

© Copyright IBM Corporation 2016

Additional notes:

This approach is more of a bottom-up approach, but you must be aware of the top-level `refreshAreas`.

Next: `wc.service.declare`

wc.service.declare

- Declares functions for which you want asynchronous page changes to occur.
- Invoked by using wc.service.invoke()
 - Used in page code. For example,
`onclick='wc.service.invoke("myNewService", {name1:value1,name2:value2})'`
- Declare example:

```
<wc.service.declare  
    id="AjaxDeleteOrderItem"  
    actionId="order_updated"  
    url="AjaxOrderChangeServiceItemDelete"/>
```

© Copyright IBM Corporation 2016

Next: [wc.render.declareRefreshController](#)

wc.renderdeclareRefreshController

- Defines controllers for refresh areas
- Includes:
 - **id**: controller name, which the refresh areas use to define binding between area and controller.
 - **url**: view to be executed to get new contents for refresh area.
 - **renderContextId**: a context of name and value pairs.
- For example,

```
<wc.renderdeclareRefreshController  
    id="currentOrderTotalsAreaController"  
    url="${AjaxCurrentOrderInformationViewURL}"  
    renderContextId="currentOrderTotals_Context" >  
    <jsp:attribute name="modelChangedScript">  
        if (message.actionId in order_updated) { widget.refresh(); }  
    </jsp:attribute>  
</wc.renderdeclareRefreshController>
```

© Copyright IBM Corporation 2016

Next: wc.widget.RefreshArea

wc.widget.RefreshArea

- Define areas that update based on AJAX requests.
- Includes:
 - **controllerId**: refresh controller responsible for refreshing this area.
- For example,

```
<div dojoType="wc.widget.RefreshArea"  
      widgetId="OrderTotalsRefreshArea"  
      controllerId="currentOrderTotalsAreaController"/>
```

© Copyright IBM Corporation 2016

Next: Pulling it together, an example

Development approach

After completing this topic, you should be able to:

- Explain the process for customizing a Web 2.0 enabled Commerce store.
- Determine the proper components to modify and produce wanted behavior.
- Perform basic problem determination and troubleshooting in a Web 2.0 enabled store.

© Copyright IBM Corporation 2016

Next: Troubleshooting and problem determination

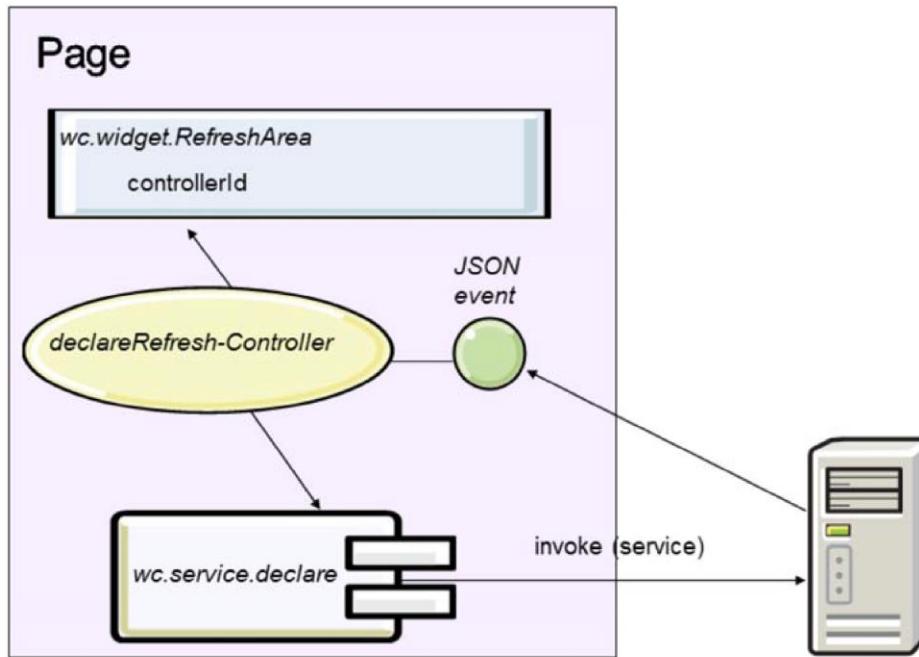
Pulling it together, an example (1 of 2)

- On page load, the quick cart and mini shopping cart register listeners for “AddToCart” event.
 - Using refreshArea and declareRefreshController
- The “AjaxAddOrderItem” service is defined, with an actionId of “AddToCart”.
 - Using declareService
- Dropping an item in the quick cart results in a call to “AjaxOrderChangeServiceItemAdd”, using AJAX.
 - “AjaxAddOrderItem” service is invoked.
- On successful response, an “AddToCart” event is created.
- Any listeners now issue AJAX requests to update their data.
 - Refresh controllers refresh their quick cart and mini shopping cart by using the URL defined in the refresh controller.

© Copyright IBM Corporation 2016

Next: Continuation of the example

Pulling it together, an example (2 of 2)



© Copyright IBM Corporation 2016

1. Declare a service to be invoked.

This service is then invoked, based on user input, making a request to update the model with new data. When a response is received, a JSON event is generated.

2. Declare the refresh controller and any refresh areas that depend upon the controller.

These areas contain the conditions for triggering an update.

3. Based on the event, refresh.

The event indicates that the refresh area is now stale. Therefore, the refreshController refreshes the data by using a request to the service to fetch the new state.

Additional notes:

Purpose — This slide shows how the RefreshArea, RefreshController, and service interact with each other.

Details — Use the diagram to tie together how the components work with each other.

Next: Development approach

Troubleshooting and problem determination

- Turn on debug output
 - In JSTLEnvironmentSetup.jspf, set the dojo debug flag to true:
`<c:set var="dojoDebugFlag" value="true"/>`
– This causes pages to "log" information by opening the Dojo debug console at the bottom of the page.
- Use one of the available JavaScript debuggers.
 - Firebug, a plug-in for Firefox
 - Microsoft Script Editor (part of the MS Office package)
 - Must be enabled in IE after installation
- Use 'debugger;' in your JavaScript to set a breakpoint.

© Copyright IBM Corporation 2016

Next: Checkpoint

Checkpoint

1. What are some of the advantages and disadvantages of a Web 2.0 store solution?
2. What elements are required on a Web 2.0 JSP in order to introduce a new refresh area?

© Copyright IBM Corporation 2016

Write down your answers here:

1.

2.

Next: Unit summary

Checkpoint solutions

1. Advantages and disadvantages

- Advantages:
 - Requests are made for data or sections of pages, not for entire pages.
 - The application can appear to be more interactive.
- Disadvantages:
 - Overused, AJAX features get “chatty”.
 - Additional complexity in development and debugging.

2. Refresh area, refresh controller, service (declare and invoke)

© Copyright IBM Corporation 2016

Overview of the WebSphere Commerce Spring framework

© Copyright IBM Corporation 2016

This section describes an overview of the WebSphere Commerce Spring framework.

Overview of the Spring framework(1)

- Comprehensive programming and configuration model for modern Java based enterprise applications
- Lightweight, open source, modular framework
- Spring framework is used for building and configuring Management Center, manages URL requests to retrieve and process object data
- Spring Features that are used are:
 - Inversion of Control (IoC), dependency injection and the Spring bean feature
- Spring framework layers and modules that are used to build Management Center
 - The Core Container layer
 - Modules: spring-core, spring-beans, spring-context, and spring-expression
 - The Spring web layer
 - Modules: spring-web and spring-webmvc
 - Central dispatcher servlet is used to handle URL requests using Spring MVC framework

© Copyright IBM Corporation 2016

WebSphere Commerce uses the Spring Framework because it is a lightweight, open source programming and configuration model for developing Java based enterprise applications.

This framework is used in WebSphere Commerce for building and configuring Management Center.

The framework manages the URL requests from Management Center to retrieve and process object data.

Developers can therefore focus more on application-level programming while the framework provides built-in best practices and design patterns.

The Spring framework is a modular framework for building enterprise applications.

Specifically, Management Center uses the Core Container layer and the Spring Web Layer.

From the Core Container layer, we use the spring-core, spring-beans, spring-context, and spring-expression modules.

The spring-core and spring-beans modules are the fundamental parts of the Spring framework.

Together, these modules provide the Inversion of Control (IoC), dependency injection, and Spring bean features of the framework, which are used in WebSphere Commerce.

The spring-context module is used for accessing objects like Java EE features (EJBs) and adds support for internalization. The spring-expression module provides the expression language that the framework uses to query and process a bean object.

From the Spring web layer, WebSphere Commerce uses the spring-web, and spring-webmvc modules.

The spring-web module is used by Management Center to initialize the Spring Inversion of Control container.

The spring-webmvc module, which is also known as the Spring Web- Servlet module, provides the Spring Model-View-Controller (MVC) framework that is used by Management Center.

The Spring MVC framework is a request-driven framework that uses a central dispatcher servlet to

handle the URL requests from Management Center.

Specifically, the Spring DispatcherServlet dispatches requests to the controller classes that are identified within bean definition controller configurations for retrieving and processing data.

Overview of the Spring framework (2)

- Spring bean definitions define the Management Center Objects that are managed by the Spring framework
 - Controller class
 - Unique identifier for the bean element
 - Properties, dependencies, and configurations/parameters
- The Spring MVC framework contains the following parts:
 - Model- is a map interface object that represents data that is transformed into the XML formatted view expected by Management Center.
 - View- represents code for displaying the XML-formatted responses to users
 - Controller – the controller class processes the data to generate the model.

© Copyright IBM Corporation 2016

Spring bean definitions also known as controller configurations, define the Management Center objects are defined in a Springframework configuration file.

The bean definitions are managed by the Spring framework IoC container.

The framework handles the instantiation and garbage collections of the bean objects.

The attributes, properties, and dependencies in each configuration must define the following information for the Spring bean object:

The controller class to use to retrieve or process the bean object or service URL request from Management Center.

A unique identifier for the controller configuration bean element which facilitates mapping and searching of the Management Center object.

The properties that are to be used to define how the associated controller class is to process the URL request and use the bean object.

The dependencies that reference other beans or resource bundles and any other configuration settings or parameters that are needed for the class to handle the bean object and URL request.

Any other configuration settings or parameters to include within the bean object.

The Spring MVC framework is composed of the following parts:

Model, represents the actual Management Center object data (this could be business or database code).

The Model object is a map interface, which the framework transforms into the XML formatted View that is expected by Management Center. The framework uses the appropriate serialization JSP fragments identified by the Controller to transform the Model to create the View.

View, represents code for displaying XML-formatted response of the Model that displays to Management Center users. The view is built using serialization JSP fragments defined by the Controller.

Controller, is the controller class that processes the data to generate the Model. When the Controller receives a URL request from Management Center, it uses the corresponding controller configuration to generate the Model and select the JSP file to use to transform the Model into the View.

Overview of the Spring framework (3)

- Improves the underlying architecture of Management Center and helps developers to focus more on application-level business logic
- Customizing the Management Center Spring framework definitions
 - A `spring-extension.xml` file is provided to override existing mappings or to define mappings for custom objects and views
 - Do not modify any `spring-ibm-component.xml` files
 - All Spring configuration files are included within the LOBTools project in IBM WebSphere Commerce Developer environments and in the LOBTools web module in other WebSphere Commerce environments
 - In IBM WebSphere Commerce Developer, configuration files exist under the directory: LOBTools > WebContent > WEB-INF

© Copyright IBM Corporation 2016

Together with all the previously mentioned components, they improve the underlying architecture of Management Center and helps developers to focus more on application-level business logic

All Management Center customization is configured through the Spring framework definitions.

A `spring-extension.xml` file is provided to override existing mappings or to define mappings for custom objects and views.

Do not modify any `spring-ibm-component.xml` files. This file is provided for each Management Center component to define the URL mappings for retrieving and processing data for the component objects.

Only use the `spring-extension.xml` file to change or define mappings.

All Spring configuration files are included within the LOBTools project in IBM WebSphere Commerce Developer toolkit.

For the exact location in the toolkit, the files exists under directory:LOBTools > WebContent > WEB-INF

Spring framework extension file

- The spring-extension.xml file takes precedence over all other configuration files.
- Includes several samples definitions:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="/Logon"
          class="com.ibm.commerce.foundation.client.lobools.spring.AuthenticationClientLibraryController">
        <property name="urlObject" value="Person"/>
        <property name="contextParameters">
            <props>
                <prop key="channelId">channelId</prop>
            </props>
        </property>
        <property name="clientLibrary" value="com.ibm.commerce.member.facade.client.MemberFacadeClient"/>
        <property name="clientLibraryMethod" value="authenticatePassword"/>
        <property name="aliasParameters">
            <props>
                <prop key="password">logonPassword</prop>
            </props>
        </property>
        <property name="generateLTPAToken" value="true"/>
        <property name="successView" value="/jsp/commerce/shell/restricted/AuthenticationSuccess.jsp"/>
        <property name="failureView" value="/jsp/commerce/shell/restricted/AuthenticationFailed.jsp"/>
    </bean>
```

© Copyright IBM Corporation 2016

Since the spring-extension.xml takes precedence over all other configuration files, it helps ensure that your configurations can override changes you need over the spring-ibm-component.xml file.

Spring-ibm-component.xml is a reserved file so any changes made in the spring-extension.xml file is preserved after applying any maintenance or upgrades to WebSphere Commerce.

Initially, the spring-extension.xml file contains sample bean definitions. In the sample code, we have an example to enable Single Sign on.

I want to point out the three main items to be aware of.

- 1) The bean tag represents a controller or view resolver configuration. It contains an ID for identification. This value is used while resolving URL requests. The class is the controller class used to process the request.
- 2) The property tag is used to pass information to the controller about the object and how to process the request.
- 3) The props tag is used to pass additional information for a configured property element. For example, you can define default parameters.

Introduction to Commerce Composer

© Copyright IBM Corporation 2016

This section describes an overview of Commerce Composer

What is the Commerce Composer?

- The Commerce Composer is a tool that business users can use to:
 - Design layouts and assign them to your category, catalog entry, and content pages, without involving IT
 - Create a URL for new pages
 - Create content pages
 - Preview the changes on the spot and on any device like desktop, tablet, mobile etc.
 - Manage SEO properties, including
 - URL keywords, page title, meta description, meta keyword, and image alt text

© Copyright IBM Corporation 2016

Next: Commerce composer UI

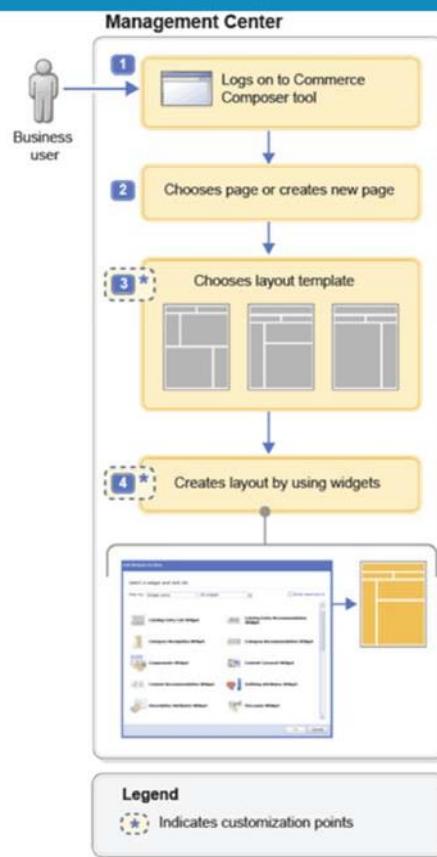
Introduction to Commerce Composer

- Tasks available are
 - Create content pages
 - To add a content page to your store, URL is created for the page directly in the Commerce Composer tool.
 - For Example : Landing pages for a brand, category, or promotion
 - Create and manage layouts
 - Design layouts and assign to category, catalog entry, and content pages, without involving IT.
 - Preview how the new pages look on different devices
 - View your page layouts at different resolutions to simulate desktop, tablet, and mobile device screens.
 - Test the layouts on actual devices by generating a shareable URL from within store preview.
 - Manage SEO content for pages
 - To optimize the page data for search engines
 - Include the URL keyword, page title, meta description, meta keyword, and image alt text
 - Copy pages and layouts
 - Saves time by copying an existing content page or layout to use as a starting point for a new page or layout.
 - Search and browse for pages and layouts
 - Search for pages and layouts by name or by advanced criteria
 - Organize content pages and layouts in folders
 - Allows by creating folders to organize the content pages and layouts.

Copyright 2016 IBM Corporation

Architecture

- Layout Templates
- Set of widgets
- Metadata information



© Copyright IBM Corporation 2016

The steps in the diagram are explained here:

- 1 The business user logs on to Management Center and opens the Commerce Composer tool.
- 2 The business user chooses a layout template to use as a starting point for the new layout. Each template is a grid that contains a specific arrangement of slots. By default, multiple layout templates are available with WebSphere Commerce for a user to select from and use to create a layout.

Customization point: You can create layout templates that suit your store design and make the templates available to business users in the Commerce Composer tool. For more information about creating layout templates, see [Creating Commerce Composer layout templates](#).

- 3 The business user adds widgets to the numbered slots in the layout template, and defines the properties and content of each widget. By default, multiple widgets are available with WebSphere Commerce. For more information about the widgets that a user can include in template slots, see [Commerce Composer widget library](#).

The values for the widget properties are associated with the specific layout and are typically stored in the [PLWIDGETNVP](#) database table. If widget properties are simple name-value pair properties, the property name and value that is associated with the layout is stored in the PLWIDGETNVP table. If you want to store more complex or language-specific data, then you must use a specific widget manager class. For more information, see [Defining a Commerce Composer widget manager class](#).

A layout is composed of three components, a layout template, a set of widgets, and metadata information. A layout template is an HTML grid outline of the page that divides the page into

the configurable slots. For more information about layout templates, see [Commerce Composer layout template architecture](#). Widgets A widget is an independent user interface module that retrieves and displays a specific type of data on a store page. Widgets are the interchangeable building blocks that a Management Center user can use to compose layouts for store pages. For more information, see [Commerce Composer widget architecture](#). Metadata information. This information is used to bind widgets to the slots that are defined in the layout template definition.

How does it look?

The screenshot shows the IBM Commerce Composer interface for managing page layouts. The title bar reads "Management Center Tools" and "Commerce Composer". The top navigation bar includes "File", "Edit", "View", "Help", "Welcome wcsadmin | Log Out", and the IBM logo.

The main area displays the "CategoryPageLayout (Read-Only)" page. On the left, a sidebar lists categories like "Content Pages", "Landing Pages", "Promotional Pages", "Store Pages", and various "Catalog Pages for External" and "Internal" sites. The central workspace is divided into two tabs: "Manage Layout" and "Design Layout", with "Design Layout" currently selected. The "Wireframe" section shows a grid of six slots (1-6) arranged in two columns and three rows. Below the wireframe is a table titled "Layout slots and widgets" showing the current configuration:

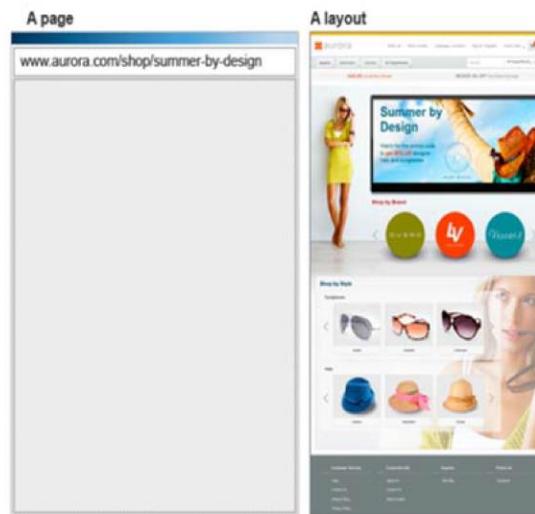
Slot	Sequence	Widget Name	Widget Type
2	0.0	HeaderLeftBannerContentWidget	E-Marketing Spot Widget
4	0.0	E-Marketing Spot Widget	E-Marketing Spot Widget

At the bottom of the workspace, there are "Save and Preview", "Save", and "Close" buttons. The footer of the interface includes the copyright notice "© Copyright IBM Corporation 2016".

Next: Managing pages

Managing Pages

- A *page* is a specific URL on your storefront with no content. Content is defined by Layout.
- Catalog page – No need to create them; Manage layout.
 - Category pages (department pages, subcategory pages)
 - Catalog entry pages:
- Content pages
- Page creation – Specify URL Keyword, Page title, Meta description and Meta keyword.



© Copyright IBM Corporation 2016

Catalog pages are created automatically when you add new catalog entries and categories to your master and sales catalogs. For example, if you add a Furniture category, your store will automatically have a page URL for the Furniture category page. Therefore, in the Commerce Composer tool, you do not create catalog pages, but you do manage their layouts.

- Product pages
- Bundle pages
- Kit pages
- Dynamic kit pages
- Predefined dynamic kit pages
- SKU pages

The term *content page* refers to any page in your site that is not a catalog page but that your company manages in theCommerce Composer tool.

- Landing pages for a brand, category, or promotion Promotional pages, such as a Deal of the Day page
- Store information pages, for example, About Us, Contact Us, Return Policy Home page
- Site map page

Next: Managing layout

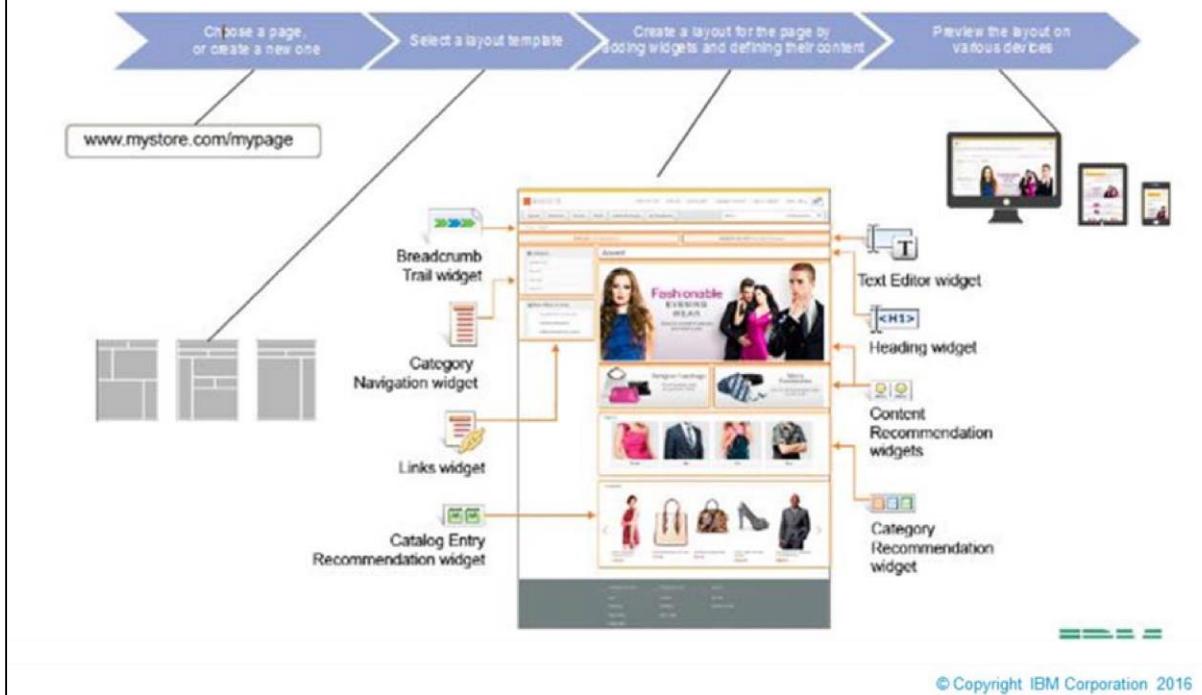
Managing Layout

- Layout - an arrangement of widgets that contain or retrieve store content.
- Layout template – a reusable wireframe that contains a specific arrangement of slots.
- Creating layout for a page –
 - Right click page in commerce composer, select new layout
 - Choose the required layout template
 - Specify the needed widgets in the design layout tab.

© Copyright IBM Corporation 2016

Next: Layout process

The process for creating layouts



Next: Commerce composer

Widgets

- A widget is a frame that displays a specific type of store content, such as ads, product recommendations, or navigational links.
- Widget content
 - Some widgets require you to define their content, either:
 - Directly in the widget.
 - By running a web activity in the widget.
 - Some widgets automatically retrieve and display content. For example:
 - The Category Navigation widget automatically retrieves a list of subcategories for the category that a shopper is viewing. You do not need to define the list of subcategories in the widget.
 - Some widgets retrieve content from external applications, such as Facebook.
- Page-dependent widgets
 - Category page-dependent
 - Search results page-dependent widgets
 - Catalog entry page-dependent widgets

© Copyright IBM Corporation 2016

Next: Widgets overview

Widgets Overview

- Content recommendation
- Catalog Entry recommendation
- Content Carousel
- E-marketing spot
- Text Editor
- Breadcrumb Trail
- Sitemap
- Category Navigation
- Facet Navigation
- Catalog Entry List
- Associated Assets
- Search Summary
- Name, Part Number and Price
- Short description
- Long description
- Full image
- Defining attribute
- Descriptive attribute
- Merchandising association
- Facebook like
- Inventory Availability
- Discounts

© Copyright IBM Corporation 2016

Content recommendation - Displays content, such as images, text, static HTML, and videos. Supports web activities.

[Catalog Entry Recommendation widget](#) - Displays the thumbnail image, name, and price of recommended catalog entries. Supports web activities.

[Content Carousel widget](#) - Displays a series of images that transition from one to the next. Supports web activities.

[Marketing Spot widget](#) - Displays content, catalog entries, and categories in common or page-specific e-Marketing Spots. Supports web activities.

[Text Editor widget](#) - Displays the HTML content or rich text that you enter directly in the widget by using a built-in text editor. [Breadcrumb Trail widget](#) - Displays a horizontal list of links that show customers their current location within the site hierarchy. [Site Map widget](#) - Displays links to category pages on a site map page.

[Category Navigation widget](#) - Displays a list of category links to help customers navigate as they browse and search your catalog.

[Facet Navigation widget](#) - Displays a list of facets, such as Brand, Price, and Color, so that customers can filter the catalog entry list on the page.

[Catalog Entry List widget](#) - Displays the list of catalog entries that are applicable to the current category or search results page.

[Search Summary widget](#) - Displays the number of search results that were returned and suggests search terms when there are no matches

[Name, Part Number, and Price widget](#) - Displays the name, part number, and price of a catalog entry. [Short Description widget](#) - Displays the short description of a catalog entry.

[Long Description widget](#) - Displays the long description of a catalog entry.

[Full Image widget](#) - Displays the full image of a catalog entry. Also displays any angle images that are defined as associated assets for the catalog entry.

[Defining Attributes widget](#) - Displays the defining attributes of a catalog entry, either in a drop-down list or as swatch images. [Descriptive Attributes widget](#) - Displays the descriptive attributes of a catalog entry in a bulleted list.

[Merchandising Associations widget](#) - Displays the catalog entries that are defined as merchandising associations (cross-sells, up-sells, replacements, and accessories) for a catalog entry.

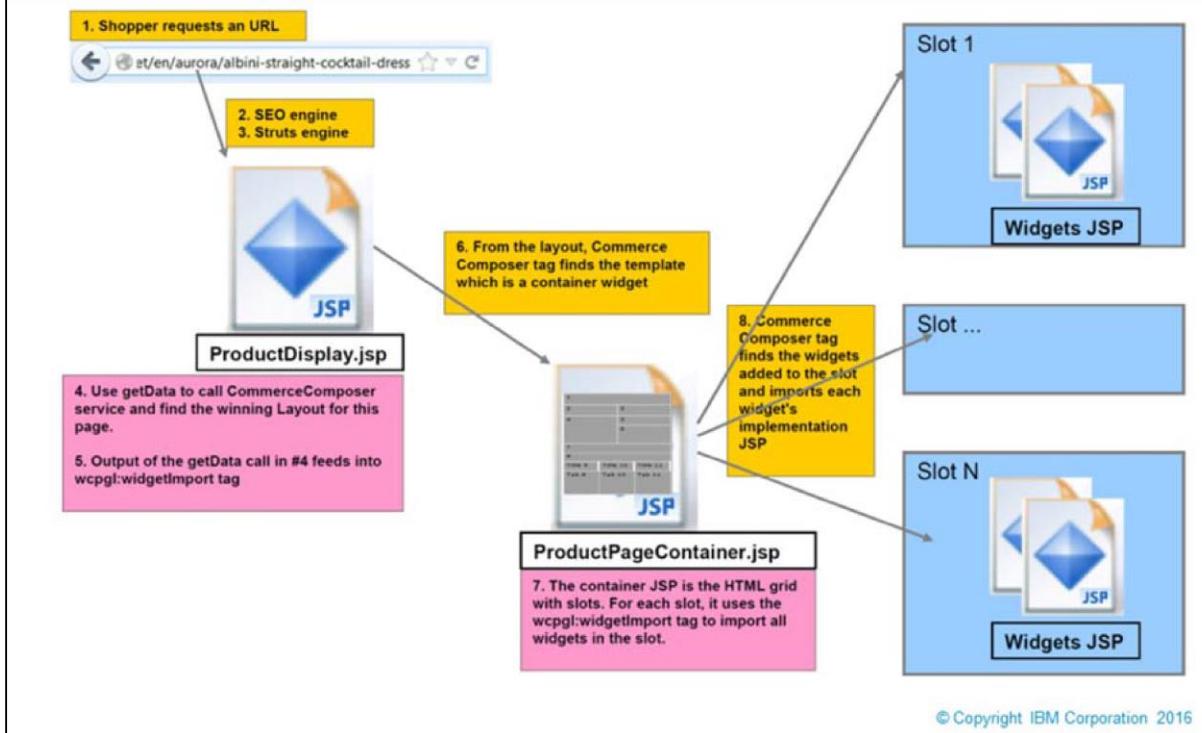
[Shopper Actions widget](#) - Displays the **Quantity** entry field, and the **Add To Cart** and **Add To Wish List** buttons for a catalog entry.

[Facebook Like widget](#) - Displays a **Like** button to allow customers to "like" a catalog entry. Also includes a **Send** button. [Inventory Availability widget](#) - Displays information about the online and in-store availability of a catalog entry.

[Discounts widget](#) - Displays the short descriptions of promotions that a catalog entry qualifies for. Each short description is a link to the long description of the promotion.

Next: Commerce composer flow

Commerce Composer Flow



Next : Creating a new template

Creating a new template

- The following are the steps a developer would take to create a new template
- Create a JSP page(widget container) by defining the layout structure using <div>'s and widgetImport tags
- Register container widget in Commerce Composer framework in PLWIDGETDEF table
- Subscribe the store to use the container widget in PLSTOREWIDGET
- Register the template with an entry in PAGELAYOUT table and set the isTemplate to true
- The template must have a root widget that is a container widget. Register the widgets that the template has in PLWIDGET and PLWIDGETREL
- Describe the container widget for Management Center to display the template thumbnail. Each slot of the container widget needs to be described with exact positions and unitless dimensions

© Copyright IBM Corporation 2016

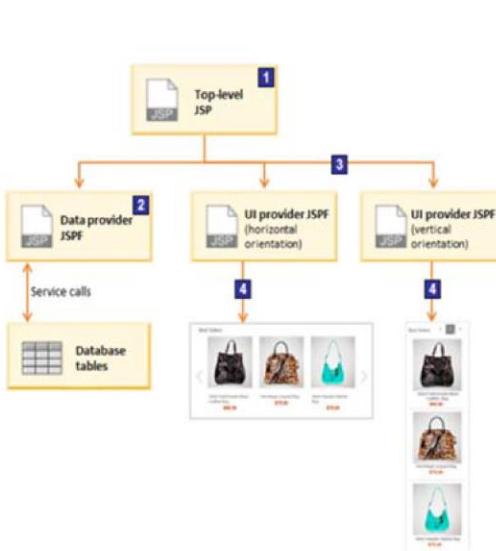
Next: Commerce composer widget architecture

Widget Architecture

A widget has two components:

Storefront components

Management Center components



Slot	Sequence	Widget Name	Widget Type
1	0.0	Catalog Entry Recommendation Widget	Catalog Entry Recommendation Widget

Widget Properties

Widget name: Catalog Entry Recommendation Widget.

Widget orientation: Horizontal.

Widget display style: Transparent background with no border.

Show a Subscribe link:

Widget Content

Select a method for populating this widget:

- Specify a list of catalog entries
- Use web activities to recommend catalog entries

Display title (United States English):

Catalog entries:

Type	Code	Name	Sequence
0 of 0 selected			

1 of 1 selected

© Copyright IBM Corporation 2016

A widget has two components: Storefront components, which include a stand-alone, compilable JSP unit that retrieves and displays data on a store page. The JSP files can be customized to display differently and to display different content based on the configurable properties that are selected by business users. If the widget JSP files follow responsive web design patterns, the widget can also be used on store pages that display for any device class (mobile, tablet, desktop). The caching and performance of the widget in the storefront must also be a consideration when customizing the storefront components for a widget.

Management Center components, which provide a properties view to business users. Users can select any configurable properties in this view to specify how the widget displays and what data the widget displays. The Commerce Composer tool, which is used to manage widgets, allows for easy saving and retrieval of widget property information. The widget properties and controls that business users configure are retrieved by the storefront components of the widget to determine the storefront display of the widget.

Store front component

- 1 On the storefront, the top-level JSP file is called and runs. The definition of the widget within the top-level JSP file identifies the data provider and UI providers for the widget.
- 2 The top-level JSP file uses the data provider JSP fragment to retrieve data from the database by using data bean, service, or RESTcalls.
- 3 The top-level JSP file is used to determine the appropriate UI provider JSP fragment to render the data that is retrieved with the data provider. The widget definition can either set a UI provider or include logic so that Management Center users can select the UI provider.
- 4 The widget is rendered on the store page by using the selected UI provider.

Management Center components

When a Management Center user adds a widget to a layout template slot, the properties view for a widget display for configuration by the Management Center user.

The properties view displays the following information: The widget name that identifies the widget in

the current layout and explains how the widget is used. For example, you can use a widget to display a page banner image, clearance products area, and more.

The widget properties are optional name-value pair properties that are stored by the Commerce Composer framework within the [PLWIDGETNVP](#) database table. When a widget renders within a layout the value for properties are passed to the store JSP files to help determine how the widget displays.

The widget content contains the optional properties for the widget, which are stored externally from the Commerce Composer framework.

Next: custom widget creation

Creating a new widget (1/2)

The following are the steps a developer would take to create a new widget

- Generating Commerce Composer widget source code
 - Use the Java Emitter Template (JET) PageComposer Resource pattern to generate source code files that you can use as a starting point for creating a custom Commerce Composer widget. Create a pattern input file that contains the information that is required to generate the source code files.
- Defining storefront assets for a Commerce Composer widget
 - Create the JSP files, user interface provider files, and data provider files for your Commerce Composer widget. These storefront assets define the appearance and content of your widget.
- Registering a Commerce Composer widget
 - To register a widget, use the Data Load utility to load the widget identifier and definition XML into the PLWIDGETDEF database table.
 - If your widget must display content in multiple languages load the relationship between the widget description and the language in PLWIDGETDESC
 - Use the Data Load utility to load a store subscription to your widget in PLSTOREWIDGET

© Copyright IBM Corporation 2016

The following are the steps a developer would take to create a new widget

- Generating Commerce Composer widget source code
- Use the Java Emitter Template (JET) PageComposer Resource pattern to generate source code files that you can use as a starting point for creating a custom Commerce Composer widget. To use this pattern, you must create a pattern input file that contains the information that is required to generate the source code files.
- Defining storefront assets for a Commerce Composer widget
- Create the JSP files, user interface provider files, and data provider files for your Commerce Composer widget. These storefront assets define the appearance and content of your widget.
- Registering a Commerce Composer widget
- Use the Data Load utility to register your widget in the Commerce Composer framework and have your store subscribe to your widget.
- Adding Management Center support for a Commerce Composer widget
- Update the Management Center framework to support displaying your custom widget in the Commerce Composer tool. By adding support for your widget, Management Center users can access and use your custom widget in the Commerce Composer tool. Users can then specify properties for the widget, associate content with the widget, and use the widget in page layouts.
- Updating a widget to create a responsive widget
- You can create responsive widget content so that they are optimized for web browsers on desktop, tablet, and mobile devices.

Next: Commerce composer creating new widget 2/2

Creating a new widget (2/2)

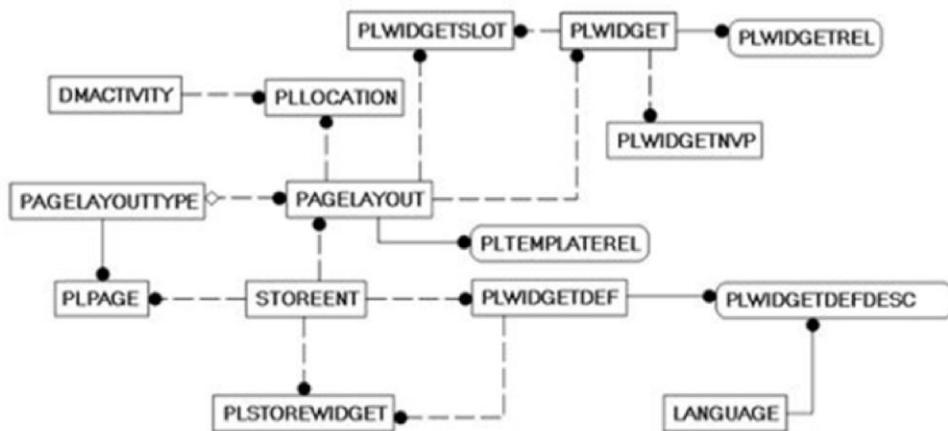
- Adding Management Center support for a Commerce Composer widget
 - Update the Management Center framework to support displaying your custom widget in the Commerce Composer tool. By adding support for your widget, Management Center users can access and use your custom widget in the Commerce Composer tool.
- Updating a widget to create a responsive widget
 - You can create responsive widget content so that they are optimized for web browsers on desktop, tablet, and mobile devices.

© Copyright IBM Corporation 2016

Next: Commerce composer data model

Composer Data Model

The following are the database tables involved in storing data for Commerce Composer layouts, widgets, and templates



© Copyright IBM Corporation 2016

PLWIDGETDEF

Contains information about all widgets registered on the system. Contains widget type (Widget/container), jsp path, XML description

PLWIDGETDEFDESC

Contains language specific descriptions for each widget in PLWIDGETDEF.

PLSTOREWIDGET

Each row specifies the relationship between a store and available widgets. Each available widget is allowed to be added to page layouts within the specified store.

PAGELAYOUT

This table contains information about various layouts present in the system – if it's layout or template, name, layout type – catentry, product, sku or bundle or category layout.

PAGELAYOUTTYPE

This table defines the group of a layout or a page. If using the Commerce Composer capability for your store, a relationship from the PAGELAYOUT table is used only for specifying a default page layout for a particular page group (or PAGELAYOUTTYPE). A relationship from the PLPAGE table represents the grouping (or PAGELAYOUTTYPE) of page. Example types for a default page layout or page include: - Category - Content - Search - Product - Kit - Bundle - DynamicKit If using the Page Layout capability for your store, this table defines the type of layout. Example values include: - HomePage - CategoryPage - ProductPage etc.

PLLOCATION

Specifies an indirect mapping between store pages and the location where they are used in the store. For every assignment of a page layout to a page, a web activity is created and referenced in this table. A 1:1 mapping exists between a page layout location and a web activity. The web activity determines which layout is used on a given page. This table provided the association between the layout and locations which you can see displayed through business tools.

PLWIDGETSLOT

Used by the business tool to display and name slots on a page layout. This table defines the configurable slots on a page layout.

PLWIDGET

This table contains the association of widgets to page layouts.

PLWIDGETNVP

This table contains information about the properties configured for a widget or container. Widgets and containers have properties when configured on a page layout.

PLTEMPLATEREL

This table stores the relationship between a page layout and the original page layout template it was based on.

PLLOCSTATIC

This table contains information about the static assignment of a layout to a user. Each PAGELAYOUT entry can have at most one entry in this table.

PLPAGE

Each entry specifies a page in the web site. A page is defined by an administrative name, type of page, URL and SEO metadata.

PLPROPERTY

This table contains information about the properties of a layout.

DMACTIVITY

A definition of the interaction with a customer to market content.

STOREENT

Each row of this table represents a StoreEntity. A StoreEntity is an abstract superclass that can represent either a Store or a StoreGroup.

LANGUAGE

Each row of this table represents a language. WebSphere Commerce supports multiple languages and is translated into ten languages by default. Using the predefined ISO codes users can add other supported languages.

Next: Commerce composer vs page composer

Page Layout vs Commerce Composer

Composer (FEP7)

- Supports creating new pages and build layouts
- Inherits page layout properties
- Developer involvement not needed while creating new pages

Page Layout tool (FEP5)

- Supports only assigning the layouts to the existing pages
- Can be migrated to composer
- Developer must design and provide layouts and enable them in CMC for to use by business users

Copyright 2016 IBM Corporation

Composer Best Practices

- Do not load layout templates that include widgets with user-defined content.
- Do not load layouts that include widgets with user-defined content
- Use widgets that always return content for display when you include widgets within tabbed layout template slots for layouts that you are loading.
- Set the cache size for the Data Load utility to be zero when you load layout assignments information.
- Clear the marketing registry when you update the layout assignment for a page.
- Invalidate the dynamic cache for your store pages after you load a new layout assignment for a page.

© Copyright IBM Corporation 2016

Layout templates

Do not load layout templates that include widgets with user-defined content.

When the Data Load utility loads layout template information, any widget that is predefined within the template cannot be edited by a user in the Commerce Composer tool. The user cannot remove the predefined widget from the template with the Commerce Composer tool. To edit the predefined widget, an administrator or developer must update the widget information in the WebSphere Commerce database with the Data Load utility or directly with SQL statements. If users do not need to define or configure content for a widget, you can predefined the inclusion of the widget within a template slot when you load the layout template information.

Layouts

Do not load layouts that include widgets with user-defined content When the Data Load utility loads layout information, any widget that is predefined within the layout cannot be edited by a user in the Commerce Composer tool. The user cannot remove the predefined widget from the layout with the Commerce Composer tool. To edit the predefined widget, an administrator or developer must update the widget information in the WebSphere Commerce database with the Data Load utility or directly with SQL statements. If users do not need to define or configure content for a widget, you can load a layout that includes this widget.

Use widgets that always return content for display when you include widgets within tabbed layout template slots for layouts that you are loading.

When a store page renders, the Commerce Composer framework checks whether tabbed slots in the layout include a widget, not whether the widget returns content. If any tabbed slot includes a widget, the tab displays on the store page, even when the widget does not display any content. When you include widgets in tabbed slots, consider including only widgets that always return content, such as default content, to ensure that the tabs on your page are not empty.

Pages

Set the cache size for the Data Load utility to be zero when you load layout assignments information.

If you remove a layout assignment and then add or update the layout assignment within the same load operation, the load process can encounter errors. If the Data Load utility retrieves unique ID data from the cache memory for objects that are already removed or changed earlier during the load process, errors can occur. When you load layout assignment information set the cache size to zero to prevent the Data Load utility from retrieving unique ID information from the cache memory. To set the value for the cache size, update the data load environment configuration file. The cache size element is defined within the ID resolver element in the file. The cache size element sets the cache memory allocation for the ID resolver. For more information about updating the environment configuration file, see [Configuring the dataload environment settings](#).

Clear the marketing registry when you update the layout assignment for a page.

When you use the Data Load utility to update the layout assignment for a page, the page might not use the newly assigned layout in store preview or on the storefront. When the Data Load utility loads a layout assignment, the utility loads information into the database. The e-Marketing Spot that the Commerce Composer uses for the layout assignment on a page is static and can be cached. If you use the Data Load utility while your server is running, the cached assignment remains and the store page can continue to use this cached layout assignment instead of your newly loaded layout assignment. The Data Load utility cannot refresh the marketing registry to clear the cached layout assignment. Whenever you load layout assignment information to update the layout assignment for a page while your server is running, manually clear the marketing registry after the load process completes. Update the registry through the Administration Console to clear the marketing caches. For more information about updating this registry, see [Updating registry components](#).

Invalidate the dynamic cache for your store pages after you load a new layout assignment for a page.

If your server is running when you load a new layout assignment for a page, the store page can still use the previous layout if the store page was cached. To ensure that the store page uses the new layout assignment, manually invalidate the DynaCache. Site administrators can invalidate the cache by restarting the server, or by running an invalidation URL within the WebSphere Commerce Administration Console browser.

Introduction to Management Center Customization

© Copyright IBM Corporation 2016

This section describes an overview of Management Center Customization

About IBM Management Center (1 of 2)

- What is it?
 - Primary tool for business users to manage business tasks.
 - Browser-based, feature rich.
 - Tools and functionality leverage WebSphere Commerce subsystems.
 - Works with other WebSphere Commerce tools.
- How can Management Center be used?
 - Catalog and merchandise management
 - Create and manage master and sales catalogs and categories.
 - Create, categorize, and bundle products and SKUs.
 - Promotions management:
 - Create and manage promotions.
 - Review and manage promotions by using the Calendar view.
 - Marketing management:
 - Design marketing campaigns that contain web and email activities.
 - Define the flow of marketing activities by using templates and builders.
 - Review marketing activities against a calendar view.

© Copyright IBM Corporation 2016

About IBM Management Center (2 of 2)

- How else can Management Center be used?
 - Asset management
 - Manage files, directories, products images, and attachments.
 - Installment rules management
 - Stores management
 - Update store profile information.
 - Specify functions available in stores.
 - Catalog filters, prices lists, and price rules management
 - Tasks management
 - Manage content within a workspace through tasks.

© Copyright IBM Corporation 2016

V8 CMC Tooling Interface

The screenshot shows the V8 CMC Tooling Interface with the "Management Center Tools" header. The "Recipes" tab is selected in the top navigation bar. The left sidebar contains links like "Search Results", "Compare View", "Active Work" (with "New Recipe Collection" expanded), "New Recipe" (selected and highlighted in blue), "Recipe by Collections", and "Unassigned Recipes". The main area displays a "New Recipe" form with tabs for "General Information", "Ingredients", and "Recipe Instruction". The "General Information" tab is active, showing fields for "Recipe Name" (with placeholder " "), "Time (min)", "Difficulty Level", and "Short Description (United)". The footer of the interface includes the copyright notice "© Copyright IBM Corporation 2016".

Business object editor modeling guidelines (1 of 2)

- The Business Object Editor (cmc/foundation/BusinessObjectEditor), class is the base class for all Management Center tools.
 - It manages all items that the user interacts with directly. For example the Management Center menu, toolbar, search widget, explorer view, and utilities view.
- Guidelines for creating a new instance of the Business Object Editor involve declaring the objects that are to be authored.
 - Include a top object definition within the declaration of the business object editor
 - If one or more top-level explorer tree nodes are required as organizational elements, they must be declared as organizational object definitions
 - All business object types that can be created, updated, or deleted must be declared as primary object definitions or as child object definitions of a primary object definition
 - If a business object exists only within the context of another object, that object type must be declared as a child object definition in a primary object definition or in another child object definition

© Copyright IBM Corporation 2016

The root class or the main base class for all tools in Management Center is the Business Object Editor class.

All objects are derived from that class. Although you never instantiate the class directly, it provides the foundation for how everything is organized for Management Center.

The objects must be declared consistently to ensure that the Business Object Editor can consume object definitions and to support different object domains.

To create a business object model that can be successfully consumed by the Business Object Editor follow these guidelines:

Include a top object definition within the declaration of the business object editor. The top object is a client-side organizational object that is not returned by a service call. The top object is not visible, but itschildren display as top-level explorer tree nodes below the Active Work node.

Think of the top object definition as an organization object definition that describes the root object for an instance of the Business Object Editor.

If one or more top-level explorer tree nodes are required as organizational elements, they must be declared as organizational object definitions.

All business object types that can be created, updated, or deleted must be declared as primary object definitions or as child object definitions of a primary object definition.

If a business object only exists within the context of another object and cannot be referenced or searched for without searching for its parent, that object type must be declared as a child object definition in a primary object definition or in another child object definition.

Business object editor modeling guidelines (2 of 2)

- Primary business objects must not have another primary business object as a direct child. An intermediate object is required to describe the nature of the relationship between primary objects
- All business objects are made up of a list of uniquely named simple properties. If an object contains a variable list of similar child properties, that property must be modeled as a separate child object type
- A business object must be uniquely identifiable by the value of the ID property
Primary objects of the same type must all have a unique ID value
- All primary objects must have a display name property
- If two business objects require different properties view declarations, they must be modeled as two separate object types

© Copyright IBM Corporation 2016

Primary business objects must not have another primary business object as a direct child. An intermediate object is required to describe the nature of the relationship between primary objects.

All business objects are made up of a list of uniquely named simple properties. If an object contains a variable list of similar child properties, that property must be modeled as a separate child object type.

A business object must be uniquely identifiable by the value of one of the properties (identified as the ID property). Primary objects of the same type must all have a unique ID value.

All primary objects must have a display name property

If two business objects require different properties view declarations, they must be modeled as two separate object types

Extended site store modeling guidelines

- Objects within an extended sites-enabled tool can be local or inherited
- Local object is an object that is owned by the currently selected store
- Inherited object is an object that is owned by a referenced asset store
- Distinguish local and inherited objects using two object definitions
- To create a common ancestor for all object types, set both the base and inherited object definitions to `createable="false"`

© Copyright IBM Corporation 2016

If the store uses extended sites stores, ensure that your tool is modeled to support extended sites, by following these guidelines:

Objects within an extended sites-enabled tool can be local or inherited. A local object is an object that is owned by the currently selected store and an inherited object is an object that is owned by a referenced assetstore.

To distinguish between local and inherited objects, the Management Center requires two object definitions - one for the local object and one for the inherited object. Since local primary business objects and inherited primary business objects are modeled as two separate object definitions in the extended sites store, you must duplicate all object definitions that are inherited from your asset store. If you are creating a base object definition to create a common ancestor for all object types, set both the base and inherited object definitions to `createable="false"` so that the user interface prevents you from creating objects of that type.

If you are modeling object definitions for extended sites objects that are managed with a Management Center tool, For a full description of the guidelines refer to the link:

http://www-01.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.management-center_customization.doc/concepts/ctf_managementcenter_customization_overview.htm

Language-sensitive objects modeling guidelines

- Language-sensitive objects are business objects that contain translatable information
- Translatable information is information that can be translated into multiple languages
- Language-sensitive object definitions must set the idProperty to languageld
- Value of the languageld property determines which language to load
- For multiple input languages, framework loads and creates instances of the language-sensitive objects, as required

© Copyright IBM Corporation 2016

Language-sensitive objects are business objects that contain translatable information. Translatable information is information that can be translated into multiple languages. For example, product names and descriptions that are offered in multiple languages are language-sensitive objects.

Model translatable information with instances of the ChildObjectDefinition class definition that has the language sensitive attribute set to true.

Language-sensitive object definitions must set the idProperty to languageld. The Management Center framework uses the value of the languageld property to determine which language to load.

If a business user selects multiple input languages from the Select Input Language dialog, the Management Center framework loads and creates instances of the language-sensitive objects, as required.

Responsive Web Design

© Copyright IBM Corporation 2016

This section describes an overview of Responsive Web design

Responsive Web Design

- Web design approach aimed at crafting pages that are optimized for a wide range of devices
- Designs one site but specifies how it appears on different devices



- Storefront shown to shoppers according to the device or such as a desktop, tablet, or web browser window size

© Copyright IBM Corporation 2016

Responsive Web Design (RWD) is a web design approach that is aimed at crafting pages that are optimized for a wide range of devices.

Rather than designing multiple sites for different-sized devices, the responsive approach designs one site but specifies how it appears on different devices.

The following diagram illustrates the Responsive Web Design approach for the Aurora starter store on several devices. The storefront is shown to shoppers according to the device or screen resolution they are using, such as a desktop, tablet, or mobile device, or web browser window size

Responsive Web Design

- The design objectives for RWD and WCS are
 - Provide rich, retail-centric RWD page templates that are accessible, localized, and touch-friendly.
 - Give business users complete control of page content and layout.
 - Provide a unique workflow to design, assemble, and test RWD pages.

© Copyright IBM Corporation 2016

The design objectives for Responsive Web Design and WebSphere Commerce are as follows:

- Provide rich, retail-centric RWD page templates that are accessible, localized, and touch-friendly.
- Give business users complete control of page content and layout.
- Provide a unique workflow to design, assemble, and test RWD pages.

Responsive Web Design framework

- Uses several technologies and design patterns to create a responsive storefront across devices
- Includes
 - Fluid layouts and grids
 - To build responsive layout templates and container widgets
 - Page components contain percentage widths
 - Dynamic adjustments are enabled when the browser window is resized
 - Defined in the WC_eardir/Stores.war/storedir/css/base.css file
 - Column stacking
 - Enabled by specifying the column span for the RWD-A breakpoint by using the acol* classes
 - Implemented in the following way:
All child elements of a row use acol12
 - Column nesting
 - Nest a row and its child elements within the child element of another row
 - Stacking order
 - Controls where a sidebar element is stacked, relative to the content element

© Copyright IBM Corporation 2016

Responsive Web Design framework (RWD) uses several technologies and design patterns to create a responsive storefront across devices.

Fluid layouts and grids is used to build responsive layout templates and container widgets. Most of the page components contain percentage widths. Dynamic adjustments are enabled for the storefront layout and page elements when the screen resolution or browser window is resized. The fluid grid system is defined in the [WC_eardir/Stores.war/storedir/css/base.css](#) file. Column stacking is enabled by specifying the column span for the RWD-A breakpoint by using the acol* classes

Column stacking is implemented in the following way: All childelements of a row use acol12.

Column nesting is possible by Nesting a row and its child elements within the child element of another row. The stacking order controls where a sidebar element is stacked, relative to the content element.

Exercise 1: Customizing StoreFront Pages

Copyright 2016 IBM Corporation

Unit summary

This course is designed to enable you to:

- Describe the behavior of the presentation layer in WebSphere Commerce.
- Understand WebSphere Commerce Struts Framework
- List the process for modifying the behavior of WebSphere Commerce Storefront Assets.
- Understand WebSphere Commerce Tag library JSTL.
- Understand and manipulate WebSphere Commerce data beans.
- Analyze the architecture of the Web 2.0 store.
- Design interactive Storefronts by using the Dojo toolkit and WebSphere Commerce extensions.
- Construct agile storefronts by using Asynchronous Java and XML (AJAX).
- Understand WebSphere Commerce Spring Framework.
- Understand the basics of Commerce Composer.
- Understand the basics of Commerce Management Center Customization.
- Understand the basics of Responsive Web design.

© Copyright IBM Corporation 2016

Additional training

For additional training needs, you have several options available for registering.

You can register	At
On site	This or any of our training facilities
Online	http://ibm.com/training/
Email	ibmswedu@us.ibm.com
By telephone	1-800-IBM Teach (1-800-426-8322)
By fax	208-692-6130

© Copyright IBM Corporation 2016

For additional training needs, you have several options available for registering like Onsite, Online , Email , by telephone or by Fax .

Unit 5

WebSphere Commerce V8 Business logic layer

© Copyright IBM Corporation 2016

This section describes an overview of the WebSphere Commerce V8 Business logic layer.

Unit objectives

This course is designed to enable you to understand:

- Name-Value Pair command processing
- Commands and the Command API
- Business Object Document processing
- Differences between NVPs and BODs
- Command behavior and Business Contexts
- Access control.

© Copyright IBM Corporation 2016

This course is designed to enable you to understand the WebSphere Commerce version 8 business logic layer with introduction to Name- Value Pair command processing, access control, Commands and the Command API, Business Object Document processing, differences between NVPs and BODs, Command behavior and Business Contexts, and access control.

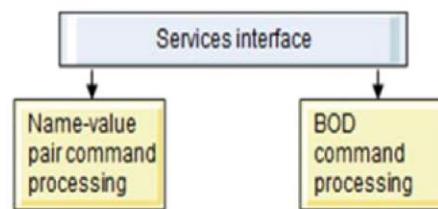
Business logic layer overview

© Copyright IBM Corporation 2016

This section describes an overview of Business Object Document processing.

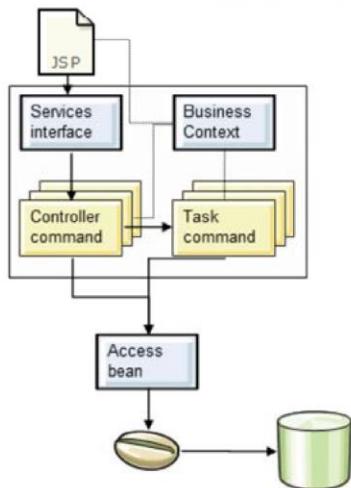
Business logic layer

- Business logic layer supports two methods of command processing:
 - Traditional name-value pair processing.
 - SOA-compliant processing of Business Object Documents (BODs).
- Both methods use the WebSphere Commerce command framework.
- Name-value pair processing:
 - Controller commands.
 - Task commands.
- BOD processing:
 - Get, Change, Process, Sync



© Copyright IBM Corporation 2016

Business logic layer processing (1 of 2)



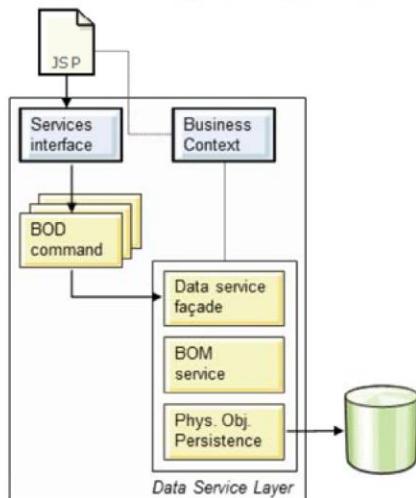
- Name-Value Pair (NVP) command processing:
 - Request for execution from the Presentation Layer.
 - Business token is passed to the Business Context service from the Data bean.
 - Service façade calls appropriate controller command (or commands) which call task command (or commands).
 - Controller and task commands call Access beans, which delegate to EJB beans.

© Copyright IBM Corporation 2016

One can think of the differences in business logic processing as direct and indirect. Name-value pairs, although they offer a level of indirection, are a more direct approach for handling business logic. They require the developer to build controller and task commands (as Java classes) which encapsulate the business logic. Either controller or task commands might then call access beans, which wrap the entity bean. Access beans provide a facade like Java for EJBs, and are therefore much easier to use than calling an EJB directly.

The Business Context Service manages contextual information, such as language, environment variables and request parameters, used by business components. The information is encapsulated within different types of Business Contexts. The abstraction of contextual information enables generic component design and personalization of content, among others. The Business Context Service is discussed in greater detail later in the course.

Business logic layer processing (2 of 2)



- **BOD command processing:**
 - Requests for execution from the Presentation Layer are in Service Data Object, or SDO, format.
 - Leverages the Business Context service.
 - Contains Pre-built BOD processing commands.
 - Commands call the Data Service Layer to access the database.

© Copyright IBM Corporation 2016

BOD processing is a more indirect, and hence flexible, method of business logic processing. The BOD processing commands are usually pre-built, but are flexible enough to allow for extension, and perform typical sorts of functions such as retrieval of data, instruction processing, data modification and system synchronization (Get, Process, Change, or Sync verbs).

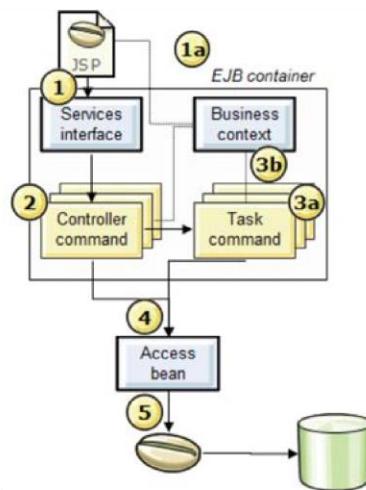
Name-value pair and BOD processing commands

- Problems with name-value pair processing:
 - Proprietary:
 - Name-value pairs are proprietary.
 - Commands and command logic is proprietary.
 - Close coupling to persistence layer:
 - Controller and task commands call the persistence layer directly through EJB beans or access beans.
 - Difficult development cycles:
 - Requires development teams to have Java experience.
 - Integration challenges:
 - Controller and task commands need to be wrapped as web services in order to be integrated.
- BOD processing is discussed later in the course.

© Copyright IBM Corporation 2016

Commands, the logic of a store

- Services façade:
 - A stateless session bean.
 - Delegates invocation of request to appropriate controller command.
- Controller command:
 - Top-level command.
 - Represents a business process.
- Task command:
 - More specific operation.
 - Might be combined with other task commands to implement behavior of a controller command.
- Access bean:
 - Java-like wrapper of an entity bean.
- Entity bean:
 - Provides database access.
- BCS (Business Context Service):
 - Separates and extracts business tokens, such as language and store, from request



© Copyright IBM Corporation 2016

1. When a request is sent from the presentation layer (that is, one of the client channels), the business logic façade intercepts it. The business logic facade is a stateless session bean that is responsible for processing the request, identifying the controller command, instantiating it, and executing it. After execution, the stateless session bean is responsible for returning the response back to the client.
 - a. At the same time, certain business tokens, like the language and store ID, are being passed along with the request and the Business Context Service handles it. The BCS might not need to perform any functionality with this data, but it needs to match the appropriate response for the request.
2. The business logic facade locates the appropriate controller command, instantiates it, and calls its execution.
3. As part of its execution, the controller command:
 - a. Might need to execute any number of task commands.
 - b. Might need to leverage business tokens (such as store ID or contract information) in order to complete its processing.
4. Both controller and task commands might call access beans, which are wrappers that are used to make working with EJB beans more manageable.
5. Access beans call their respective EJB beans, which then manipulate data on the database

Instructor notes:

Purpose — Demonstrate the structure of commands in a store.

Details — It is important to discuss each of the boxes in the diagram and to expose the student to its purpose. Highlight the extensibility of WebSphere Commerce to other modes of e-commerce, not just messaging, browser, or pervasive devices.

Additional information — The reason the business logic facade is a stateless session bean is because, to the presentation layer, it is essentially a "search." The presentation layer is requesting a "search" with parameters, which include the name of the command and any additional properties. Once the stateless session bean receives this mock request, its function is to search

the command registry (CMDREG) to match the "search parameter" with a command implementation. It then uses the command factory to instantiate the command implementation and execute it. The result of the execution is returned to the business logic facade as if it were "search results."

When the solution controller receives a request, it determines whether the request requires the invocation of a controller command or a view. In either case, the solution controller also determines the implementation class for the command and then invokes it.

Since controller commands encapsulate the logic for a business process, they frequently invoke individual task commands to perform specific units of work in the business process. Access beans are invoked when information in the database must be retrieved or updated. Either a task or controller command can invoke access beans. Requests then flow from access beans to entity beans that can read from, and write to, the WebSphere Commerce database.

Transition statement — Command framework

NVP Command framework (1 of 2)

- NVP (Name-Value Pair) Commands make up the business logic of a WebSphere Commerce store:
 - Commands are Java classes which encapsulate business logic.
 - When modifying the behavior of a store to meet requirements, it is common to create new commands, modify the implementations of existing commands, and exercise various options of existing commands.
 - Commands are invoked by the Service layer for request processing, and can return a view name used for result processing.
 - Commands can obtain contextual information using the command context.
 - Common design pattern (the command pattern):
 - Defined interface for all commands,
 - decouples client from model.

© Copyright IBM Corporation 2016

It is important to understand the WebSphere Commerce core commands to help determine how best to customize the commands in the most efficient manner for the environment.

NVP Command framework (2 of 2)

- Commands follow a specific design pattern: interface and implementation classes.
- Commands extend from abstract base classes that provide an abstraction layer for future component enhancements.
- Each store can register its own set of commands, if necessary.
- There are features in the framework to address error handling.
- Access control is also addressed in the framework.
- It is important to have a good idea of the basic flow of a request to a command.

© Copyright IBM Corporation 2016

Command beans follow a specific design pattern. Every command includes both an **interface** class (for example, **CategoryDisplayCmd**) and an **implementation** class (for example, **CategoryDisplayCmdImpl**). From the perspective of a caller, the invocation logic involves setting input properties, invoking an execute() method, and retrieving output properties.

From the perspective of the command implementer, commands follow the WebSphere command framework, which implements the standard command design pattern, allowing a level of indirection between the caller and the implementation. The key mechanisms that are enabled within this level of indirection include:

- The ability to invoke an access control policy manager that determines whether the user is allowed to invoke the command.
- The ability to execute a different command implementation for different stores **based** on the store identifier.
- The ability to execute a different view implementation, based on the device type of the requester.
- The ability to change the execution command context within the execution to execute a command for another store identifier or user ID.

Instructor notes:

Purpose — Describe the WebSphere Commerce command framework and the key points in the execution, registration, and function of the framework.

Details —

Additional information — Commands are registered and then used for different site flows. WebSphere Commerce is a flexible, extensible product. As the students go through the course, they see some fundamental design techniques that provide the flexibility to allow multilingual, multi-client, thick client, browser, PDA, and kiosk.

Transition statement — Extensible framework

Extensible framework

Command type	Example command name	Extends	Implements example interface
Controller command	MyControllerCmdImpl	com.ibm.commerce.command.ControllerCommandImpl	MyControllerCmd
Task command	MyTaskCmdImpl	com.ibm.commerce.command.TaskCommandImpl	MyTaskCmd
Data bean command	MyDataBeanCmdImpl	com.ibm.commerce.command.DataBeanCommandImpl	MyDataBean

Default commands and views:

WebSphere Commerce provides default commands and views which you can use in your store; default commands and views are listed in the Struts configuration file for the web application.

Additionally, many of the views that are used in a starter store were created specifically for it. These views are added to one of the Struts configuration files as well.

© Copyright IBM Corporation 2016

When creating new business logic for your e-commerce application, it is expected that you might need to create new controller and task commands.

New commands must implement their corresponding interface that, in turn, can extend an existing interface. To simplify command writing, WebSphere Commerce includes an abstract implementation class for each type of command. New commands must extend these classes.

Instructor notes:

Purpose — WebSphere Commerce is an application framework for use, reuse, and customization or extension.

Details — One key to note is the consideration of the existing infrastructure before doing extensions. With the significant data and object model that is provided with WebSphere Commerce, there might be a nontraditional way to use existing components. This type of customization is often simpler to migrate to future versions of WebSphere Commerce.

Additional information —

Transition statement — Next: Command flavors

NVP Command flavors

- Controller commands:
 - Top-level command that encapsulates the logic that is related to a particular business process.
 - Examples: *OrderProcessCmd* for order processing, or *LogonCmd* for allowing users to log on.
 - Contain the control statements and invoke task commands to perform individual tasks in the process.
- Task commands:
 - Implement a specific unit of application logic.
 - In general, a controller command and a set of task commands together implement the application logic.
- Data bean commands:
 - Invoked by the data bean manager, when a data bean is used, to populate the fields of a data bean.
 - Populate the fields of a data bean with data from a persistent object.
 - These commands are written to explicitly fetch certain fields, and can be useful, but should be used only when smart data beans are not practical.

© Copyright IBM Corporation 2016

Controller commands:

Controller commands encapsulate the logic that is related to a particular business process. In general, a controller command contains the control statements (for example, if, then, else) and invokes task commands to perform individual tasks in the Business Process.

Upon completion, a controller command returns a view name. Based on the view name, the store identifier, and the device type, the solution controller determines the appropriate implementation class for the view and then invokes it.

Task commands:

Task commands implement a specific unit of application logic. In general, a controller command and a set of task commands together implement the application logic for a URL request. Task commands are executed in the same container as the controller command.

Data bean commands:

The data bean manager invokes the data bean commands when aJSP page needs to instantiate a data bean. The primary function of a data bean command is to populate the fields of a data bean with data from a persistent object.

Development approach

- Why modify a command?
 - Modify a command to extend its behavior.
 - Call `super.performExecute()` in the subclass.
 - Wrap `super.performExecute()` in custom code.
- Why add a command?
 - Controller:
 - Add a controller command to replace an existing command entirely or to add a capability to the model.
 - Task:
 - Extend if possible.
 - If not, add a task command to supplement an existing controller command or introduce a new task to the model.
 - Data bean:
 - A new data bean command would be implemented to explicitly populate a new data bean.

© Copyright IBM Corporation 2016

When adding a capability (an entirely new command) to the store, one must create a new command interface and a new command implementation class.

It is a common practice to subclass an existing command's implementation class to yield a new command, and then call `super.performExecute()` to take advantage of existing behavior. At that point, the developer can add code before or after the `super.performExecute()` as required.

Extending is preferable to adding, if it can meet your requirements.

WebSphere Commerce command factory

- Uses standard object-oriented design pattern (Factory).
- Instantiates new command objects.
- Defers instantiation away from the invoking class.
- Factory class determines the correct implementation class:
 - Two ways to declare implementation class of a command:
 - **defaultCommandClassName**
Specifies the default implementation class for the interface.
 - **Command registry (CMDREG)**
Allows the declaration of different implementations for each store ID.
 - Command registry overrides the *defaultCommandClassName* specified in the interface.

© Copyright IBM Corporation 2016

To create new command objects, the caller of the command uses the *command factory*. The command factory is a bean that is used to instantiate commands. It is based on the factory design pattern, which defers instantiation of an object away from the invoking class, to the factory class that understands which implementation class to instantiate.

The factory provides a smart way to instantiate new objects. In this case, the command factory provides a way to determine the correct implementation class when creating a new command object, based on the individual store. The command interface name and the particular store identifier are passed into the new command object, upon instantiation.

There are two ways for the implementation class of a command to be specified. A default implementation class can be specified directly in the code for the command interface, using the **defaultCommandClassName** variable. For example, the following code exists in the **CategoryDisplayCmd** interface:

```
String defaultCommandClassName =  
"com.ibm.commerce.catalog.commands.CategoryDisplayCmdI mpl"
```

The second way to specify the implementation class is to use WebSphere Commerce command registry. The command registry should always be used when the implementation class varies from one store to another. More information about the command registry is discussed in the next topic.

In the case where a default implementation class is specified in the code for the interface and a different implementation class is specified in the command registry, the command registry takes precedence.

The syntax for using the command factory is as follows:

```
cmd = CommandFactory.createCommand(interfaceName, storeId)
```

where **interfaceName** is the interface name for the new command bean and **storeId** is the identifier of the store for which the command should be implemented. Typically, the store ID can be retrieved by using the **commandContext.getStoreId()** method.

Command registry

- The command registry is a database table (CMDREG) which maps command interfaces to implementations:
 - STOREENT_ID: Store entity identifier
 - Set to 0 to use the command for all stores, or to a unique store ID to indicate that the command is used for a particular store.
 - INTERFACENAME: Command interface name
 - Defines the interface. Must match the name of the inbound URL request.
 - DESCRIPTION: Description of the command
 - Arbitrary text to describe the command.
 - CLASSNAME: Command implementation class name
 - Represents the Java class that implements the command logic. Typically, this name is the same as the InterfaceName, with "Impl" appended.
 - PROPERTIES: Default name-value pairs that are set as input properties to the command
 - Format should be the same as the input query string.

© Copyright IBM Corporation 2016

WebSphere Commerce controller and task commands are registered with the command registration framework. The CMDREG database table implements the command registry.

The command registry provides a mechanism for mapping the command interface to its implementation class. Multiple implementations of an interface allow for command customization on a per store basis.

In general, when you create a new controller or task command, you should create a corresponding entry in the command registry. For example, the following SQL statement creates an entry for **MyNewCommand** that a particular store (whose store identifier is 10001) uses.

```
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION, CLASSNAME,
PROPERTIES) values
(10001,com.mycompany.commands.MyNewCommand','This is latest
command','com.mycompany.commands.MyNewCommandImpl','my
DefaulParm1=1&myDefaulParm2=2')
```

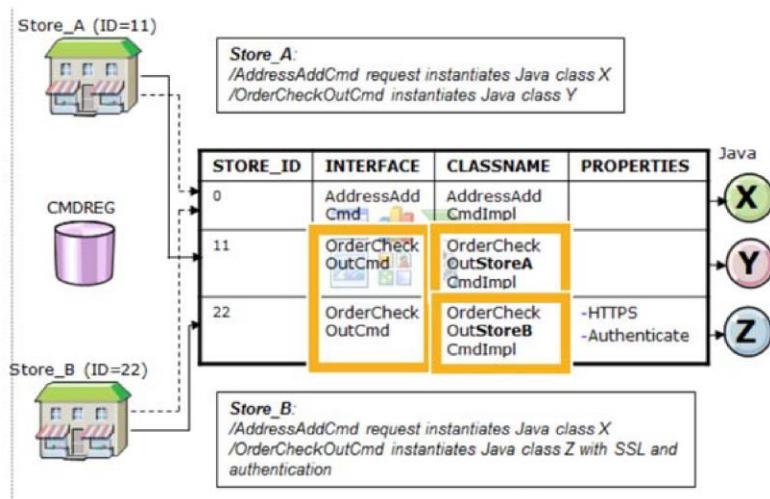
String values should be enclosed in single quotation marks when using SQL to update the Oracle, DB2, Cloudscape, or Derby databases under the WebSphere Commerce instance.

If the command you are writing always uses the same implementation class, you do not necessarily have to register the command in the command registry. In this case, you can use the **defaultCommandClassName** attribute in the interface to specify the implementation class. For example, in the code for the interface, you would include the following line:

```
String defaultCommandClassName =
"com.ibm.commerce.command.MyNewCommandImpl"
```

If you specify the implementation class in this manner, you cannot pass default properties to the implementation class and the same implementation class must be used for all stores.

Instantiating commands from command registry



© Copyright IBM Corporation 2016

Important: Package names are omitted in order to preserve space and appearance.

Consider a scenario in which your site has two stores: StoreA and StoreB. Each store has different security requirements for the OrderCheckOutCmd controller command as well as different implementations of the command. This section shows how the command registry is used to enable this customization.

The framework executes the business logic that is associated with the OrderCheckOutCmd interface. The interface is shared between the two stores, so the URL requests look almost identical; however, the store-specific configuration that is found in the command registry controls the implementation for the business logic. Even though the URL shares the business logic interface, the HTTP security consideration can differ.

To map this behavior as listed in the top table, you would do the following in the Struts configuration:

```
<action parameter="com.ibm.commerce.mycommands.OrderCheckOutCmd"
path="/OrderCheckOutCmd" type="com.ibm.commerce.struts.BaseAction">
    <set-property property="https" value="0:0,22:1" />
    <set-property property="authenticated" value="0:0,22:1" />
</action>
```

Based on entries in the URL configuration, the solution controller determines that the interface name for the OrderCheckOutCmd is com.ibm.commerce.mycommands.OrderCheckOutCmd. It also determines that StoreA requires the command to be executed without a secure socket or authentication, but StoreB requires HTTPS and authentication. The values for HTTPS and authentication are used by the solution controller for Web requests, not by the interface.

Based on entries in the command registry, the solution controller determines that for StoreA, the implementation class for the com.ibm.commerce.mycommands.OrderCheckOutCmd interface is com.ibm.commerce.mycommands.OrderCheckOutStoreACmdImpl. It also determines that for StoreB, the implementation class for the same interface is com.ibm.commerce.mycommands.

OrderCheckOutStoreBCmdImpl.

Instructor notes:

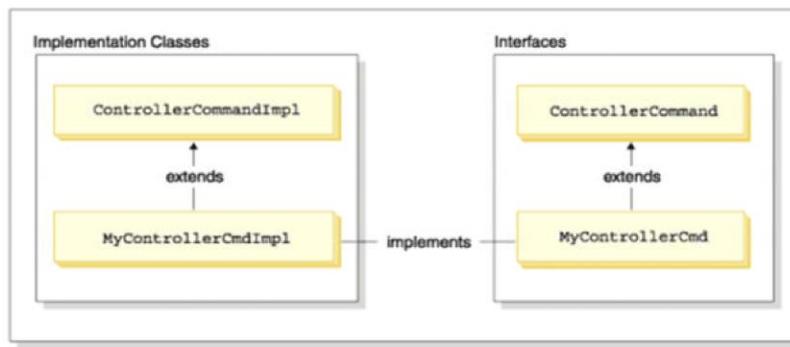
Purpose — See examples of the WebSphere Commerce command registry.
Details —

Additional information —

Transition statement — Next, controller command programming model.

Controller command programming model

- Interface extends from ControllerCommand.
- Implementation class extends from ControllerCommandImpl.
- Includes methods to implement and their usage:
 - setRequestProperties(TypedProperty): passes input properties to controller command.
 - validateParameters(): checks initial parameters and their resolution.
 - performExecute(): contains the business logic.
 - isGeneric(): returns a Boolean verifying that the command might be invoked by a generic user.



© Copyright IBM Corporation 2016

A new controller command should extend the abstract controller command class (`com.ibm.commerce.command.ControllerCommandImpl`). When writing a new controller command, you should override the following methods from the abstract class:

`isGeneric()`

In the standard WebSphere Commerce implementation, there are multiple types of users. These users include generic, guest, and registered users. Within the group of registered users, there are customers and administrators.

The generic user has a common user ID that is used across the entire system. This common user ID supports general browsing on the site in a manner that minimizes system resource usage. It is more efficient to use this common user ID for general browsing, since the web controller does not need to retrieve a user object for commands that the generic user can invoke.

The `isGeneric` method returns a Boolean value that specifies whether the generic user can invoke the command.

The `isGeneric` method of a controller command's superclass sets the value to false (meaning that the invoker must be either a registered customer or a guest customer). If generic users can invoke your new controller command, override this method to return true.

You should override this method to return true if your new command does not fetch or create resources that are associated with a user. An example of a command that the generic user can invoke is the `ProductDisplay` command. It is sensible to allow any user to be able to view products. An example of a command for which a user must be either a guest or registered user (and hence, `isGeneric` returns false) is the `OrderItemAdd` command.

When `isGeneric` returns a value of true, the web controller does not create a new user object for the current session. As such, commands that the generic user can invoke run faster, since the web controller does not need to retrieve a user object.

`isRetriable()`

The `isRetriable` method returns a Boolean value that specifies whether the command can be tried again on a transaction rollback exception. The `isRetriable` method of the new controller

command's superclass returns a value of false. You should override this method and return a value of true, if your command can be tried again on a transaction rollback exception.

An example of a command that should not be tried again in the case of a transaction exception is the `OrderProcess` command. This command invokes the third-party payment authorization process. It cannot be tried again, since that authorization cannot be reversed. An example of a command that can be tried again is the `ProductDisplay` command.

`setRequestProperties(com.ibm.commerce.datatype.TypedProperty reqParms)`

The web controller invokes the `setRequestProperties` method to pass all input properties to the controller command. The controller command must parse the input properties and set each individual property explicitly within this method. This explicit setting of properties by the controller command itself promotes the concept of type safe properties.

`validateParameters()`

The `validateParameters` method is used to do initial parameter checking and any necessary resolution of parameters. For example, it could be used to resolve `orderId=*`. This method is called before both the `getResources` and `performExecute` methods.

`getResources()`

This method is used to implement resource-level access control. It returns a vector of resource-action pairs upon which the command intends to act. If nothing is returned, no resource-level access control is performed.

`performExecute()`

The `performExecute` method contains the business logic for your command. It should invoke the `performExecute` method of the command's superclass (if possible before any custom business logic) and must return a view name.

Extending an existing controller command

- When extending a command, new logic can be called before or after calling the superclass' `performExecute()` method:

```
public class MyControllerCmdImpl extends ControllerCmdImpl {  
    public void performExecute() throws ECException {  
        // *** Insert new business logic before original command  
  
        super.performExecute();  
  
        // *** Insert new business logic after original command  
    }  
}
```

© Copyright IBM Corporation 2016

A command can throw either `ECApplicationExceptions` or `ECSysyemExceptions` that extend from the `ECException` class.

When handling your own exception handling logic, it involves:

- 1) Catching the exceptions in your command that require special processing.
- 2) Constructing either an `ECApplicationException` or `ECSysyemException` that extends from the `ECException` class, which is based on the type of exception.
- 3) For an `ECApplicationException` that requires a new message, define the message in a new properties file.

ECApplicationException This exception is thrown if the error is related to user input and will always fail. For example, when a user enters an invalid parameter, an `ECApplicationException` is thrown. When this exception is thrown, the solution controller does not retry the command, even if it is specified as a retriable command.

ECSysyemException This exception is thrown if a runtime exception or a WebSphere Commerce configuration error is detected. Examples of this type of exception include create exceptions, remote exceptions, and other EJB exceptions. When this type of exception is thrown, the solution controller retries the command if the command is retriable and the exception was caused by either a database deadlock or database rollback.

An example of how an exception can be wrapped is provided below: `throw new ECApplicationException(MyCustomCommand._CUSTOM_ERROR,CL ASSNAME, METHODNAME, new Object[] {paramval1}, null, null, null, OriginalException);`

Tracing and logging is provided via the `LoggingHelper` utility class that is provided in the `com.ibm.commerce.foundation.common.util.logging` package.

Nesting controller commands

- Once you instantiate the nested command, call its `setCommandContext()` method and pass in the current command context.
- Call `setRequestProperties(TypedProperty)` of the nested command:
 - Alternatively, call individual setter methods that are defined on the interface of the command to set the required input properties.
- After input properties are set, call `performExecute()` method of the nested command.
- All controller commands must return a view when processing completes, but the outer command can ignore the view returned by the nested command and use its own view.
- The wrapped command is executed within the transaction scope of the outer command.

© Copyright IBM Corporation 2016

Nesting controller commands can also be referred to as "wrapping."

You most often use the command factory to create instances of task commands; however, it can also be used within one controller command to create an instance of another controller command.

The syntax for instantiating task commands and controller commands is the same. That is, you specify the name of the command interface and the store ID in both scenarios.

Consider the following code snippet as an example of a nested controller command. This example shows a method in the outer command and how it can use the command factory to instantiate a second controller command.

```
YourControllerCmd ctrlCmd = null;  
public void processAndCallOtherCommand() throws ECException {  
    YourControllerCmd ctrlCmd =  
    (YourControllerCmd) CommandFactory.createCommand(com.yo  
urcompany.commands.yourControllerCmd, this.getStoreId());  
  
    ctrlCmd.setCommandContext(this.getCommandContext());  
    ctrlCmd.setRequestProperties(this.getRequestProperties());  
    ctrlCmd.execute();  
}
```

As another example, consider the case where the nested command is being executed for a different store from the first. In this case, the command context from the outer command must be preserved so that it does not get overwritten by the inner command.

```
public void processAndCallOtherCommandForOtherStore(int astoreId) throws  
ECException {
```

```

        // Make a clone to preserve the command context of the outer
        command

        CommandContext cloneCmdCtx =
        (CommandContext)this.getCommandContext().clone();

        // Now pass in a new set of request properties to the
        cloned command context

        cloneCmdCtx.setRequestProperties(reqProp);

        YourControllerCmd ctrlCmd = (YourControllerCmd)
        CommandFactory.createCommand(com.yourcompany.commands.yourControllerCmd,
        aStoreId);

        ctrlCmd.setCommandContext(cloneCmdCtx);
        ctrlCmd.setRequestProperties(reqProp); ctrlCmd.execute();
    }
}

```

Instructor notes:

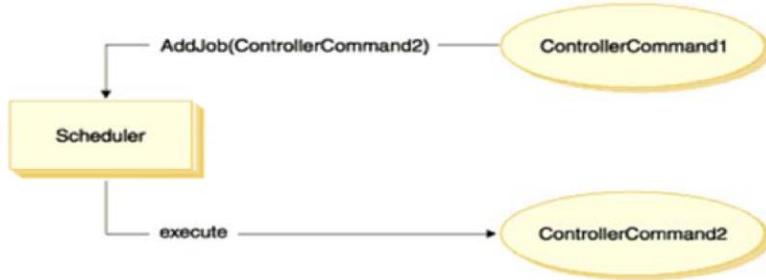
Purpose — Describe the command wrapping.

Details — If you are passing a different set of request properties to the nested command and these parameters affect the command context, you should clone the command context before instantiating the nested command. This process preserves the command context information for the outer command.

Additional information —

Transition statement — Now discuss the use of the scheduler for long- running commands.

Long-running controller commands



In this example:

ControllerCommand1 creates a job on the Scheduler (by using the AddJob command).
The job that is created on the Scheduler is ControllerCommand2.
ControllerCommand1 returns a view after adding the new job.
Scheduler controls when ControllerCommand2 gets executed

© Copyright IBM Corporation 2016

If a controller command takes a long time to execute, you can split the command into two commands.

The first command, which is executed as the result of a URL request, simply adds the second command to the scheduler, so that it runs as a background job.

The flow that is shown in the diagram on the slide is as follows:

1. ControllerCommand1 is executed as a result of a URL request.
2. ControllerCommand1 adds a job to the scheduler. The job is ControllerCommand2.
3. ControllerCommand1 returns a view, immediately after adding the job to the scheduler.
4. The scheduler executes ControllerCommand2 as a backgroundjob. In this scenario, the client typically polls the result from ControllerCommand2. ControllerCommand2 should write the job state to the database.

The WebSphere Commerce scheduler

- The WebSphere Commerce scheduler is a background process that schedules and runs jobs:
 - The scheduler can be controlled administratively or programmatically.
 - Polls the SCHACTIVE table, looking for jobs to execute.
 - Uses information in SCHCONFIG to configure the job when it is run.

© Copyright IBM Corporation 2016

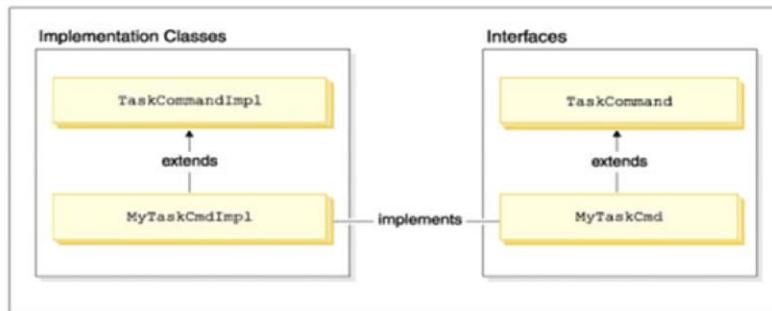
Remember: Scheduler does not prevent job overlapping by default – this can sometimes lead to problems if not accounted for. To learn more:

Preventing overlapping scheduler job executions with WebSphere Commerce job scheduler

https://www.ibm.com/developerworks/community/blogs/f75960cc-9241-4849-9199-875360adea40/entry/preventing_overlapping_scheduler_job_executions_with_websphere_commerce_job_scheduler?lang=en

Task command programming model

- Interface extends from the TaskCommand.
- Implementation class extends the TaskCommandImpl.
- The *performExecute()* method is called to execute the task command.



© Copyright IBM Corporation 2016

In Java, the new MyTaskCmdImpl task command that is pictured on the slide is defined as follows:

```

public class MyTaskCmdImpl extends com.ibm.commerce.command.TaskCommandImpl
implements MyTaskCmd
{
    ...
}
  
```

All the input and output properties for the task command must be defined in the command interface, for example **MyTaskCmd**. The caller programs to the task command interface, rather than the task command implementation class. This enables you to have multiple implementations of the task command (one for each store), without the caller being concerned about which implementation class to call.

All the methods that are defined in the interface must be implemented in the implementation class. Since the caller should set command context (a controller command), the task command does not need to set the command context. The task command can, however, obtain information about the session by using the command context.

In addition to implementing the methods defined in the task command interface, you should override the *performExecute* method from the *com.ibm.commerce.command.TaskCommandImpl* class.

The *performExecute* method contains the business logic for the particular unit of work that the task command performs. It should invoke the *performExecute* method of the task command's superclass before performing any business logic. The following code snippet shows an example *performExecute* method for a task command.

```

public void performExecute() throws ECException {
    super.performExecute();
    // Include your business logic here.
    // Set output properties so the controller command
  
```

}

```
// can retrieve the result from this task
```

The runtime framework calls the `getResources` method of the controller command to determine which protectable resources the command accesses. It might be the case that a task command is executed during the scope of a controller command and it attempts to access resources that the `getResources` method of the controller command did not return. In such case, the task command itself can implement a `getResources` method to ensure that access control is provided for protectable resources.

Instructor notes:

Purpose — Describe the WebSphere Commerce task command programming model and method usage.

Details —

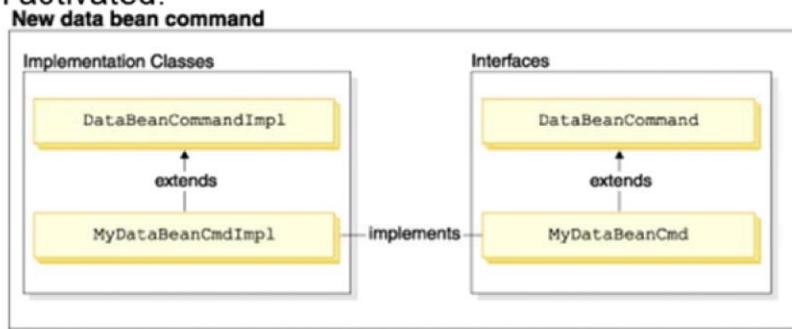
Additional information — By default, `getResources` returns null for a task command, and resource-level access control checking is not performed. Therefore, you must override this task if the task command accesses protectable resources.

Transition statement — Now discuss the third command type, data bean commands.

Data bean commands

The data bean manager calls data bean commands to populate the fields of a data bean.

- The command retrieves all fields at once, regardless of the requirements of the JSP.
- Instead of implementing the SmartDataBean interface, a data bean can implement CommandDataBean:
 - This data bean must implement the getCommandInterfaceName() method to return the command interface name used to retrieve data for this data bean.
 - The data bean manager uses this command to populate the data bean when activated.



© Copyright IBM Corporation 2016

Data bean commands are usually not used. It is a best practice to always use smart data beans when possible, or practical. However, they are useful in cases where a developer must customize the data that a bean has when populated, when the data comes from many sources, or when the data might come from some process that is applied to the data (like a sum of receipts or a count of items).

Overview of Business Object Document processing

© Copyright IBM Corporation 2016

This section describes an overview of Business Object Document processing.

Business Object Document (BOD)

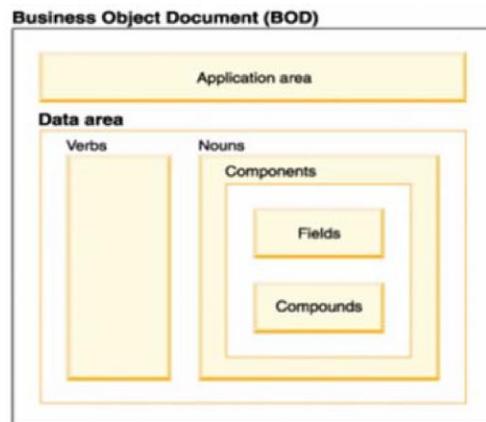
Business Object Documents (BODs) are OAGi standard messages.

Data area contains verbs and/or nouns:

- Noun: a common business object.
- Verb: an action on the Noun.
- For example, a GetCatalogEntry BOD contains a 'Get' action (Verb). A ShowCatalogEntry BOD contains complex CatalogEntry business object (Noun).

Application area contains information that the infrastructure can use to communicate the message.

- Might include information such as creation date, sender identification, authentication data, a unique BOD ID.

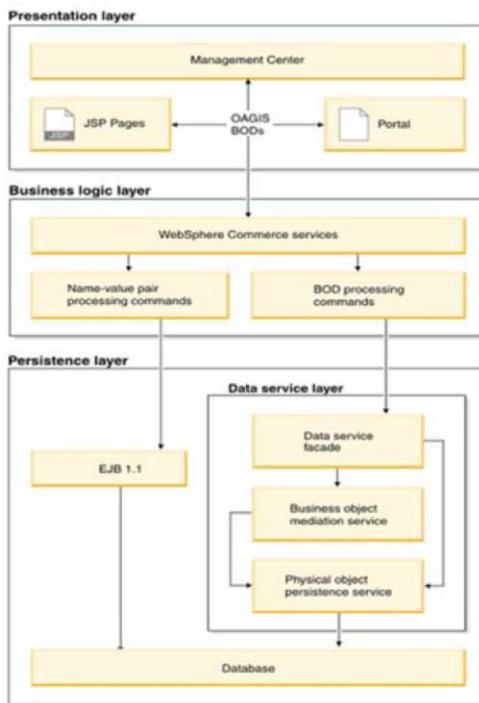


© Copyright IBM Corporation 2016

The BOD is a common horizontal message architecture for achieving interoperability between disparate systems. The BOD messages can be chained together to create different scenarios: for example, an ordering scenario might include messages for creating the order and processing the payment. BODs are the business messages that are exchanged between the presentation, business logic, and persistence layers; in other words, between components.

In order to achieve this interoperability, the message (a BOD) is constructed of a common business object (a Noun) and/or an action (a Verb). In the previous ordering scenario example, one of the BODs might be to Create (the action, a Verb) an Order (the business object, a Noun).

Relevant application layers for BOD command processing



Relevant application layers for BOD command processing are:

- Presentation layer – interacts with the business logic layer through OAGIS enabled web services.
- Business logic layer – organized into service modules, or components (marketing, order, catalog, promotion, member, infrastructure, price, contract). Leveraged by the presentation layer to return data or invoke a business process.
- Persistence layer - retrieves and stores data.

© Copyright IBM Corporation 2016

Decoupling allows reuse in other environments besides the WebSphere Commerce application, such as WebSphere Portal applications, the IBM Sales Center, and any other custom applications.

WebSphere Commerce is moving toward an approach that focuses on multichannel support instead of an approach that is primarily web-based. Instead of the name-value pair URLs that represent a typical web request, business components declare structured objects that represent services. Components use OAGi XSDs to define services. This feature adds value by enabling developers to create business services that can be used regardless of protocol.

WebSphere Commerce now provides the following logical groupings of services to standardize communication:

- marketing,
- order,
- catalog,
- promotion,
- member,
- infrastructure,
- price,
- contract.

In the current WebSphere Commerce architecture, there are three

different layers of the application: the presentation layer, the business logic layer, and the persistence layer. In WebSphere Commerce Version 6 Feature Pack 3.0.1, WebSphere Commerce introduced a new command framework for the business logic layer - the business object document (BOD) command framework. The concept of a BOD was used in previous versions of WebSphere Commerce, but recently, the entire command framework is tailored to also use BODs.

In prior versions of WebSphere Commerce, there are implementation dependencies between the presentation layer, business logic layer, and persistence layer. The new architecture uses well-defined interfaces to decouple the implementation of the presentation layer, business logic layer, and

persistence layer. From the business logic layer perspective, OAGi messages are used as the interface for making requests to retrieve business data or invoke business logic. The BOD command framework provides the capability to process these BOD requests and responses.

The interaction between the business objects and persistence layer is isolated in the business object mediator. Business object document (BOD) commands interact with the business object mediator to handle the interaction with the logical objects and how they are persisted.

Instructor notes:

Purpose — Describe the outbound web service strategy in WebSphere Commerce.

Details — Of special interest: you can use BOD processing as a "go-between" between the traditional name-value pair processing and the service-oriented BOD processing. Instead of using BOD commands in the component services framework, you can institute a solution that uses BODs to process name-value pairs. A business can use this solution to maintain their existing name-value pair commands while they migrate to BOD processing.

Additional information —

Transition statement — Next: BOD processing commands overview

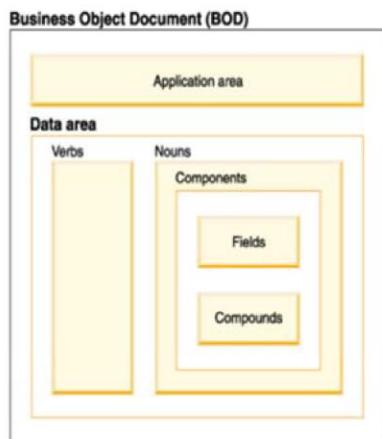
BOD processing commands overview

- BOD commands deal with Service Data Objects (SDO) instead of name-value pairs.
 - BODs and SDOs are widely accepted and integrate easier with other applications and software.
- BODs can represent a complex request that performs multiple actions instead of just one.
 - Task commands represent a single action. A controller command is a single action wrapper of several smaller tasks
- BOD commands deal with a persistence interface called the data service layer by using an object that is called the business object mediator, and are independent of the persistence technology.
 - A clearer separation between business logic and persistence logic

© Copyright IBM Corporation 2016

As part of the WebSphere Commerce move to an SOA architecture, three different layers of the application are identified. Consistent interfaces define the interactions between the layers. In previous versions of WebSphere Commerce, there are implementation dependencies between the presentation layer, business logic layer, and persistence layer. The new architecture in WebSphere Commerce version 6, Feature Pack 3.0.1 and later addresses this issue by using OAGi messages to interact with the business logic. The business logic command framework provides the capability of processing business object documents.

BOD processing commands (1 of 2)



- OAGi Business Object Documents:
 - Noun represents business object
 - Verb represents action to perform
 - Implemented in Java by Service Data Objects (SDO)
- Component facade exists to implement services:
 - Application area
 - Contains application context information (language, store, and so on)
 - Verb
 - Get or Show
 - Process or Acknowledge
 - Change or Respond
 - Sync or Confirm BOD
 - Noun
 - Logical representation of the business object.

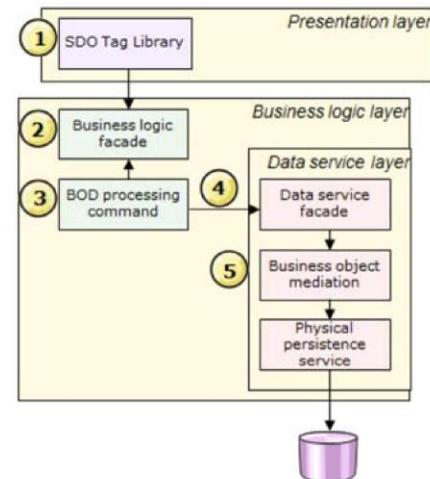
© Copyright IBM Corporation 2016

The Business Object Document (BOD) is an open standard for a common horizontal message architecture that the Open Applications Group (<http://www.openapplications.org>) developed. BODs are business messages that are exchanged between software applications or components. A BOD is a message structure that contains an application area and a data area to operate on. The application area of the BOD describes the application context to associate with the processing. The application area is associated with specific application context information to control the processing and indicate application context when returning the result. In the context of a WebSphere Commerce service, the application area is where store, language, and other session type data is found. This information is common across all component facades and uses this information during processing.

The data area contains the service operation (verb) and/or the business object of the operation (noun). The verb indicates what operation to perform. The noun represents the data in the business object that is related to the request or response.

BOD processing commands (2 of 2)

1. Inbound request is sent as a service BOD from a client channel.
2. Business logic facade is invoked for execution of the request
3. BOD command is executed on a logical object (Java-based SDO).
4. Command processing calls persistence through data service layer façade.
5. Business object mediation transforms logical object (Java) into physical object (database row).



© Copyright IBM Corporation 2016

Advantages of BOD command processing - summary

- Standardization
 - Uses standard OAGi messages, processing standard BODs.
 - Commands are more generic.
- Abstraction
 - Well-defined separation between logical and physical data.
 - Decouples components for reuse in different environments.
- Simplification
 - Less Java development, more XML, and XPath development.
 - Lightweight runtime environment for request handling.
- Integration
 - BOD processing closely resembles web services, easier to integrate.
 - Extensive set of services for working with data.

© Copyright IBM Corporation 2016

Extending a BOD service to manage UserData with the Data Service Layer

- BOD services can be extended to include custom user data in responses. The most straightforward and common example is to:
 - Extend an OOTB access profile (service response contents),
 - Define SQL queries used by the new access profile to retrieve custom data from the database.
- The new access profile is specified during BOD service invocation, e.g., from a getData tag in a JSP, or from a service client object in Java.
- Data retrieved by separate SQL statements is merged into a single object through database table relationships and object-relational metadata specified during service customization.
- More on the topic later in the class.

© Copyright IBM Corporation 2016

Example Java invocation of a custom access profile:

```
// Create the CatalogEntry client. This client is used to send requests and receive responses from  
the server.  
  
CatalogEntryFacadeClient iCatalogEntryClient = new CatalogEntryFacadeClient(businessContext,  
callbackHandler);  
  
String catEntryId = "10001";  
String[] catEntryIdArray = {catEntryId};  
  
/**  
 * The access profile used in Get requests.  
 */  
String getCustomizationAccessProfile = "Sample_GetCatalogEntryDetails";  
/**  
 * The XPath used to find catalog entries.  
 */  
String findByCatalogEntryIdXPath = "/CatalogEntry[CatalogEntryIdentifier[(UniqueID={0})]]";  
  
// Create the get expression using the access profile, XPath, and catentry to find.  
SelectionCriteriaHelper selectionCriteriaHelper = new
```

```

SelectionCriteriaHelper(MessageFormat.format(findByCatalogEntryIdXP ath, catEntryIdArray));
selectionCriteriaHelper.addAccessProfile(getCustomizationAccessProfile);
ExpressionType getExpression = selectionCriteriaHelper.getSelectionCriteriaExpression();

// Create the get verb using the get expression.
GetType get = iCatalogEntryClient.createGetVerb(getExpression);

try {
    // Perform the get request.
    ShowCatalogEntryDataAreaType showCatalogEntryDataArea
    = iCatalogEntryClient.getCatalogEntry(get);

        // Validate the show response. List
        showCatalogEntries =
showCatalogEntryDataArea.getCatalogEntry(); CatalogEntryType aCatalogEntry = null;

        Iterator showCatalogEntriesIter =
showCatalogEntries.iterator();

        while (showCatalogEntriesIter.hasNext()) { aCatalogEntry =
(CatalogEntryType)
showCatalogEntriesIter.next();
assertEquals("30", aCatalogEntry.getUserData().getUserDataField().get("warterm"));
assertEquals("LIMITED", aCatalogEntry.getUserData().getUserDataField().get("wartype"));
    }
} catch (CatalogEntryException e) {
    fail(e.getMessage());
}
}

```

Access profile definition with related SQL – in the query template file:

```

BEGIN_SYMBOL_DEFINITIONS
<!-- WebSphere Commerce tables --> COLS:CATENTRY=CATENTRY:*
<!-- Sample extension tables --> COLS:XWARRANTY=XWARRANTY:*

END_SYMBOL_DEFINITIONS

BEGIN_ASSOCIATION_SQL_STATEMENT
name=Sample_GetCatalogEntryDetails_SQL base_table=CATENTRY
<!-- Only return objects in the ENTITY_PKS list --> additional_entity_objects=true
sql= SELECT
CATENTRY.$COLS:CATENTRY$,

```

```

XWARRANTY.$COLS:XWARRANTY$ FROM
CATENTRY
LEFT OUTER JOIN XWARRANTY ON (CATENTRY.CATENTRY_ID = XWARRANTY.CATENTRY_ID)
WHERE
CATENTRY.CATENTRY_ID IN ($ENTITY_PKS$) AND CATENTRY.MARKFORDELETE = 0
END_ASSOCIATION_SQL_STATEMENT

BEGIN_PROFILE
name=Sample_GetCatalogEntryDetails extends =
IBM_Admin_Details BEGIN_ENTITY
associated_sql_statement=Sample_GetCatalogEntryDetails_SQL END_ENTITY
END_PROFILE

```

Example inbound request:

```

<?xml version="1.0" encoding="UTF-8"?>
<_cat:GetCatalogEntry xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:_cat="http://www.ibm.com/xmlns/prod/commerce/9/catalog"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:oa="http://www.openapplications.org/oagis/9" releaseID="9.0"
versionID="7.0.0.0">
<oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
<oa:CreationDateTime>2016-10-02T07:22:46.144Z</oa:CreationDateTime>
<oa:BODID>03e24020-8871-11e6-80cb-839857eecccd0</oa:BODID>
<_wcf:BusinessContext>
<_wcf:ContextData name="languageId">-1</_wcf:ContextData>
<_wcf:ContextData name="catalogId">10001</_wcf:ContextData>
<_wcf:ContextData name="storeId">10201</_wcf:ContextData>
</_wcf:BusinessContext>
</oa:ApplicationArea>
<_cat:DataArea>
<oa:Get>
<oa:Expression expressionLanguage="_wcf:XPath">{_wcf.ap=Sample_GetCatalogEntry
Details}/CatalogEntry[CatalogEntryIdentifier[(UniqueID=10001)]]</oa:Exp re ssion>
</oa:Get>
</_cat:DataArea>
</_cat:GetCatalogEntry>

```

Example response:

```

<?xml version="1.0" encoding="UTF-8"?>
<_cat>ShowCatalogEntry xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:_cat="http://www.ibm.com/xmlns/prod/commerce/9/catalog"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"

```

```

xmlns:oa="http://www.openapplications.org/oagis/9" releaseID="9.0">
<oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
  <oa:CreationDateTime>2016-10-02T07:22:50.808Z</oa:CreationDateTime>
  <oa:BODID>06a9eba0-8871-11e6-80cb-839857eecc0</oa:BODID>
</oa:ApplicationArea>
<_cat:DataArea>
  <oa:Show recordSetCompleteIndicator="true" recordSetCount="1"
recordSetReferenceId="064de8f0-8871-11e6-80cb-839857eecc0" recordSetStartNumber="0"
recordSetTotal="1">
    <oa:OriginalApplicationArea>
      <oa:CreationDateTime>2016-10-02T07:22:46.144Z</oa:CreationDateTime>
      <oa:BODID>03e24020-8871-11e6-80cb-
839857eecc0</oa:BODID>
    </oa:OriginalApplicationArea>
  </oa:Show>
  <_cat:CatalogEntry catalogEntryTypeCode="ProductBean" displaySequence="1.0">
    <_cat:CatalogEntryIdentifier>
      <_wcf:UniqueID>10001</_wcf:UniqueID>
      <_wcf:ExternalIdentifier ownerID="700000000000000101">
        <_wcf:PartNumber>AuroraWMDRS-1</_wcf:PartNumber>
        <_wcf:StoreIdentifier>
          <_wcf:UniqueID>10051</_wcf:UniqueID>
        </_wcf:StoreIdentifier>
      </_wcf:ExternalIdentifier>
    </_cat:CatalogEntryIdentifier>
    <_cat:OwningStoreDirectory>ExtendedSitesCatalogAssetStore</_cat:OwningStoreDirectory>
    <_cat:Description language="-1">
      <_cat:Name>Hermitage Fit and Flare Dress</_cat:Name>
      <_cat:Thumbnail>images/catalog/apparel/women/wcl000_dresses/200x3
10/wcl000_0028_a_red.jpg</_cat:Thumbnail>
      <_cat:FullImage>images/catalog/apparel/women/wcl000_dresses/646x1
000/wcl000_0028_a_red.jpg</_cat:FullImage>
      <_cat:ShortDescription>Jewel-toned cocktail dress with fitted bodice and gently flared
skirt</_cat:ShortDescription>
      <_cat:LongDescription>Classically tailored, this elegant satin dress features a bateau
neckline, soft pleating, and a clasped belt to define the waist.</_cat:LongDescription>
      <_cat:Attributes name="published">1</_cat:Attributes>
      <_cat:Attributes name="available">1</_cat:Attributes>
    </_cat:Description>
    <_cat:CatalogEntryAttributes>
      <_cat:Attributes displaySequence="0.0">
        <_cat:Name>DiscountCalculationCode</_cat:Name>

```

```
<_cat:AttributeDataType>String</_cat:AttributeDataType>
<_cat:StringValue>
  <_cat:Value>33042,075,724 testpromotion- 10001101</_cat:Value>
</_cat:StringValue>
</_cat:Attributes>
<_cat:Attributes displaySequence="0.0">
  <_cat:Name>DiscountCalculationCodeUniqueId</_cat:Name>
  <_cat:AttributeDataType>String</_cat:AttributeDataType>
  <_cat:StringValue>
    <_cat:Value>11151</_cat:Value>
  </_cat:StringValue>
</_cat:Attributes>
<_cat:Attributes displaySequence="0.0">
  <_cat:Name>manufacturer</_cat:Name>
  <_cat:AttributeDataType>String</_cat:AttributeDataType>
  <_cat:StringValue>
    <_cat:Value>HermitageCollection</_cat:Value>
  </_cat:StringValue>
</_cat:Attributes>
<_cat:Attributes displaySequence="0.0">
  <_cat:Name>buyable</_cat:Name>
  <_cat:AttributeDataType>String</_cat:AttributeDataType>
  <_cat:StringValue>
    <_cat:Value>1</_cat:Value>
  </_cat:StringValue>
</_cat:Attributes>
<_cat:Attributes displaySequence="0.0">
  <_cat:Name>state</_cat:Name>
  <_cat:AttributeDataType>String</_cat:AttributeDataType>
  <_cat:StringValue>
    <_cat:Value>1</_cat:Value>
  </_cat:StringValue>
</_cat:Attributes>
<_cat:Attributes displaySequence="0.0">
  <_cat:Name>subscriptionTypeld</_cat:Name>
  <_cat:AttributeDataType>String</_cat:AttributeDataType>
  <_cat:StringValue>
    <_cat:Value>NONE</_cat:Value>
  </_cat:StringValue>
</_cat:Attributes>
<_cat:Attributes displaySequence="0.0">
```

```

<_cat:Name>disallowRecurringOrder</_cat:Name>
<_cat:AttributeDataType>String</_cat:AttributeDataType>
<_cat:StringValue>
    <_cat:Value>1</_cat:Value>
</_cat:StringValue>
</_cat:Attributes>
</_cat:CatalogEntryAttributes>
<_cat>ListPrice>
    <_wcf:Price currency="USD">250.00000</_wcf:Price>
</_cat>ListPrice>
<_cat:ParentCatalogGroupIdentifier>
    <_wcf:UniqueId>10006</_wcf:UniqueId>
    <_wcf:ExternalIdentifier ownerID="7000000000000000101">
        <_wcf:GroupIdentifier>Dresses</_wcf:GroupIdentifier>
        <_wcf:StoreIdentifier>
            <_wcf:UniqueId>10051</_wcf:UniqueId>
        </_wcf:StoreIdentifier>
    </_wcf:ExternalIdentifier>
</_cat:ParentCatalogGroupIdentifier>
<_cat:NavigationRelationship displaySequence="1.0" type="child-parent">
    <_cat:CatalogGroupReference>
        <_cat:CatalogGroupIdentifier>
            <_wcf:UniqueId>10006</_wcf:UniqueId>
            <_wcf:ExternalIdentifier ownerID="7000000000000000101">
                <_wcf:GroupIdentifier>Dresses</_wcf:GroupIdentifier>
                <_wcf:StoreIdentifier>
                    <_wcf:UniqueId>10051</_wcf:UniqueId>
                </_wcf:StoreIdentifier>
            </_wcf:ExternalIdentifier>
        </_cat:CatalogGroupIdentifier>
        <_cat:CatalogIdentifier>
            <_wcf:UniqueId>10001</_wcf:UniqueId>
        </_cat:CatalogIdentifier>
    </_cat:CatalogGroupReference>
</_cat:NavigationRelationship>
<_cat:FulfillmentProperties>
    <_wcf:UserData/>
</_cat:FulfillmentProperties>
<_wcf:UserData>
    <_wcf:UserDataField name="warterm">30</_wcf:UserDataField>
    <_wcf:UserDataField name="wartype">LIMITED</_wcf:UserDataField>

```

```
</_wcf:UserData>
  </_cat:CatalogEntry>
</_cat:DataArea>
  </_cat>ShowCatalogEntry>
```

Commands versus Component Services (1 of 2)

- Name-value pair processing:
 - Based on invoking controller commands with parameters.
 - For example, UserRegistrationAdd?Firstname=John&Lastname=Smith...
 - HTTP(S) centric
- Component Services (BOD):
 - Based on OAGIS BODs with verbs and nouns.
 - A message contains a verb and possibly a noun:
 - Verb: Register
 - Noun: Person [Firstname=John, Lastname=Smith, ...]
 - Transport independent but can be mapped to HTTP(S).

© Copyright IBM Corporation 2016

Commands versus Component Services (2 of 2)

- Name-value pair processing:
 - Requires developer to build controller and task commands
 - Either controller or task commands calls access beans
 - Access beans are wrapped over entity bean
 - Access beans uses façade, for example Java for EJB
- Component Services (BOD):
 - OAGIS messages are used as the interface for making requests to retrieve business data
 - BOD commands deal with SDO instead of name-value pairs
 - BODs can represent a complex request
 - BOD commands are independent of the persistence technology

© Copyright IBM Corporation 2016

Overview of Command Behavior, Business Contexts

© Copyright IBM Corporation 2016

This section describes an overview of Command Behavior,Business Contexts.

Business context services (1 of 3)

- The business context service manages contextual information for an operation:
 - Before an operation begins, business context services (BCS) might set up the contextual information.
 - During the operation, the data is available to the components in the operation.
 - When the operation is complete, BCS is invoked to handle any post processing, such as persistence.

© Copyright IBM Corporation 2016

Business context engine

WebSphere Commerce Context Engine is the core of the WebSphere Commerce SOA run time. It supports different client types: WebApp, Portal, Rich Client.

It contains two major components:

- Business context service (BCS):
 - Stateless session bean and web service enabled.
 - Supports stateful business contexts.
 - API to client and business component.
- Business contexts:
 - Context-specific logic.
 - Interaction with Business Context service by using SPIs. (Stateful Packet Inspection, traditionally, is a form of routing that tracks connection states).
 - Stateful with context data cached across transactions.
 - APIs to business components.

BCS and selected contexts are installed into every Commerce application server instance. The business context service (BCS) formalizes your context infrastructure. By abstracting contextual information from components, it fosters more reuse.

Abstracting user contextual information allows for:

- Enablement of generic components.
- Tailored content and experience.
- Precisely targeted offers.
- Enforcement of business policies.
- Honoring appropriate prices, entitlements, and terms.

BCS session persistence

It provides for implementation of session data regardless of whether the request is from a browser or telesales client. These two tables store session information:

- CTXMGMT: Contains the master record for every session.
- CTXDATA: Contains context information. The contents are the context name and the context value in a query string format.

Business context definition

The contexts that are associated with the request are defined in the `<installedApps>/<node>/WC_<instance>/xml/config/businessContext.xml` file. The default setup is to use the production set of contexts with the following contexts:

- BaseContext: contains caller ID (user ID), run-as ID (for user), store ID, and channel ID.
- EntitlementContext: contains the contract information that is associated with the user.
- GlobalizationContext: contains the globalization and locale information that is associated with the session.
- AuditContext: used as a supported way to carry forward transport information, such as HTTP remote address.
- PreviewContext: a new feature that is required for previewing content in the future.

Command deprecation

Many methods of the command context are deprecated: getUserId(), getStoreId(), and so on. The replacement for these methods is to use the corresponding context that contains the information.

- getUserId() ?? BaseContext.getCallerId()
- getLocale() ??

Globalization.getLocale() However, some methods do not have replacements.

- These methods were never exposed to the business logic anyway. These methods include getRemoteAddress() and other transport-specific methods.
- Some of this information is preserved by using the AuditContext.

Instructor

notes:

Purpose —

Details —

Additional information —

Transition statement — Next: Business context services (2 of 3).

Business context services (2 of 3)

- Business contexts:
 - Contexts establish an execution environment that affects the output of a business component.
 - Clients do not directly invoke contexts; business components use the information present in a context to fulfill operations.
- Benefits:
 - Enablement of generic components.
 - Tailored content and experience.
 - Precisely targeted offers.
 - Enforcement of business policies.
 - Appropriate prices, entitlements, and terms for a particular user.

© Copyright IBM Corporation 2016

The business context service manages contextual information that the business components use. The information is encapsulated within different types of business contexts. This process formalizes the context infrastructure and fosters reuse between different business models.

Before the logic to process the operation is invoked, contextual information is set up and managed by the business context service. After the processing is complete, business context service is invoked to handle any post processing of the context information, such as persistence. In the business context model, a client invokes a service to operate.

Optionally, the client might receive a response. An example of a business service is a service that helps a company fulfill the needs of its customers. Multiple services can be logically grouped into business components. A component specializes in a function such as catalog management or tax calculation.

Instructor notes:

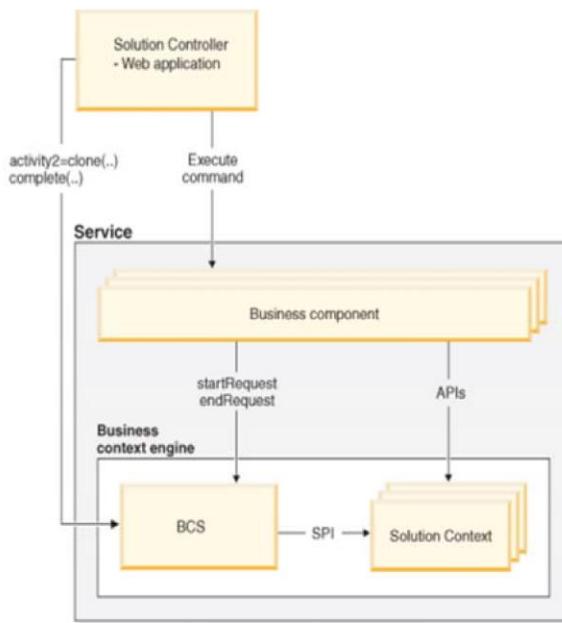
Purpose — WebSphere Commerce business services provide for the personalization, user and promotional policies, and reuse via the flyweight and controller command design patterns.

Details —

Additional information —

Transition statement — Next: Business context services (3 of 3)

Business context services (3 of 3)



© Copyright IBM Corporation 2016

Solution controller:

A solution controller is responsible for managing the activity or activities of a user. In the WebSphere Commerce Struts framework, the solution controller exists as part of the base action. In the Sales Center, the solution controller exists as part of the rich client application. The client performs a limited set of actions while the solution controller provides the additional BCS process hooks.

Activity token:

An identifier for an activity that spans multiple requests and transactions, an activity usually begins when a user logs on to the site and ends when a user logs off. The activity token is unique for each caller, run-as user, and store. The activity token groups the set of contexts effective during a particular request from the client to the various business components.

The client supplies this activity token on every request to indicate the experience that it would like from the business components. Usually, the solution controller initiates a request to the business context service to request an activity token.

Business components:

A business component groups a set of related services together, and a service provides a specialized function. It interacts with business context service to signify the beginning and the end of a request such that any preprocessing and post processing on business contexts can be done. Also, components can retrieve and update various business contexts during the execution of a request.

Business context engine

The business context engine (BCE) provides contexts that dictate the behavioral characteristics of business components over the lifetime of the session. The business context engine comprises the business context service and business contexts.

Business context service

The solution controller uses the business context service (BCS) to manage contexts that the business components require. BCS is also an interface that the components use to obtain contexts they need. Multiple unique contexts that various business components use are associated under a single identifier that is known as an activity, for a limited lifetime. The lifetime of an activity spans multiple requests and transactions. The contexts that are associated with an activity maintain their state for the lifetime of the activity. BCS also manages a cache of the business contexts that are associated with an activity within a request. This cache saves resources that would otherwise be used to initialize and destroy multiple business contexts that are used in the same request.

Types of business contexts

- **BaseContext:**
 - This context contains the basic attributes that an activity needs, such as store ID, caller ID, and the run-as ID.
- **EntitlementContext:**
 - This context holds information about entitlement criteria, such as reduced prices for gold club membership.
- **GlobalizationContext:**
 - This context helps components determine locale-specific information such as what language a message can be rendered in, or what currency can be used in the calculation of a price.
- **ContentContext:**
 - If Workspaces are enabled, this context determines the content or business objects that can be displayed or edited based on version information.

TaskContext:

If Workspaces are enabled, this context determines which task an administrator is performing.

AuditContext:

The third-party components provide this context. You might want to bridge the gap to the third-party interface instead of programming to it directly.

This context enables you to connect to a different implementation of the service in the future from a vendor without the need to rewrite your component.

PreviewContext:

This context is used for staging content.

ExperimentContext:

This context is used for storing experiment data.

© Copyright IBM Corporation 2016

Business contexts have the following characteristics:

- A context establishes an execution environment that affects the output of a business component for the equivalent input, based on the solution needs.
- The output that a component produces for an **input** is always identical for the same set of contexts.
- Clients of business processes do not directly invoke contexts. Instead, it is the business component that uses the services these contexts provide during the execution of a request.
- A context provides a set of service methods and might optionally store session data that does not change often over an activity life span.
- The life span of a context starts with the creation of an activity and ends with the completion of the activity.

Each business context implements two interfaces:

- ContextSPI:** The ContextSPI interface defines methods for a business context service to indicate events such as the beginning of an activity, the start, and end of a request.
- Context:** Each business context that defines methods for business logic extends the context interface to use to retrieve context-specific information.

WebSphere Commerce command context

- A handle to the solution controller.
- Wraps business context objects.
- Can be modified for specific command execution.
- A container of common environment objects:
 - User ID
 - User object (UserAccessBean)
 - Language identifier
 - Store ID
 - Store object (StoreAccessBean)
 - Currency
 - Trading agreements
 - Servlet request data

© Copyright IBM Corporation 2016

Overview of WebSphere Commerce Access control

© Copyright IBM Corporation 2016

This section describes an overview of WebSphere Commerce Access control.

Access control policies in WebSphere Commerce

- Policies define the access group, actions, resources, and relationships, defining a set of conditions in access control.
- Key concepts in understanding access control policies:
 - Users: members, system users
 - Users are assigned to groups that have a specific role (Registered User / Guest).
 - Resources: protected artifacts, such as JSPs or commands.
 - Resources are grouped into resource groups.
 - Actions: an operation, performed on a resource:
 - Activities that a user can perform on the resource.
 - Actions are grouped into action groups.
 - Relationships: the “link” between user and resource.
 - For example, users are Owners of the orders (resources) they create.
- Access control is implemented as a means of supporting authorization for Commerce resources.

© Copyright IBM Corporation 2016

Access control in a WebSphere Commerce application 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. 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 that is used in a store is a view. A view is invoked to display a store page to customers. The views that are 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 a user and a resource. Access control policies might require that a relationship between a user and a resource is satisfied. For example, users might only be allowed to display orders which they created.

Types of access control

- Two types of authentication:
 - WebSphere Application Server (broad)
 - WebSphere Commerce Server (fine)
- Two types of access control (authorization):
 - Command-level (broad)
 - Also known as role-based.
 - Specifies that users assigned to a particular role can execute certain commands.
 - For example, guest users can execute the commands that are contained in action group X.
 - Resource-level (fine)
 - Specifies the relationship that a user must have with a resource before an action can be performed.
 - For example, users can display only an order they created.

© Copyright IBM Corporation 2016

In a WebSphere Commerce application, there are two main levels of authentication and authorization. WebSphere Application Server performs the first level of access control. In this respect, WebSphere Commerce uses WebSphere Application Server to protect enterprise beans and servlets. The second level of access control is the fine-grained access control system of WebSphere Commerce.

The WebSphere Commerce access control framework uses access control policies to determine whether a user is permitted to act on a resource. This access control framework provides fine-grained access control. It works with, but does not replace, the access control that the WebSphere Application Server provides.

There are two types of access control, both of which are policy-based:
command-level access control and resource-level access control.

Command-level (also known as role-based) access control uses a broad type of policy. You can specify that all users of a particular role can execute certain types of commands. For example, you can specify that users with the Account Representative role can execute any command in the AccountRepresentativesCmdResourceGroup resource group.

Another example policy is to specify that all seller administrators can perform an action that is specified in the SellerAdministratorsViews group on any resource that the ViewCommandResourceGroup specifies.

A command-level access control policy always has the ExecuteCommandActionGroup as the action group for controller commands. For views, the resource group is always ViewCommandResourceGroup.

Command-level access control protects all controller commands. In addition, command-level access control must protect any view that can be called directly, or that a redirect from another command (in contrast to being launched by forwarding to the view) can launch.

Command-level access control determines whether the user is allowed to execute the particular command within the store you specify. If a policy allows the user to execute the command, a subsequent resource-level access control policy could be applied to determine whether the user can access the resource in question.

Consider the case when a seller administrator attempts to perform an administrative task. The first level of access control checking determines whether this user is allowed to execute the particular seller administration command within the current store. The user is allowed to run the command if the command belongs to the ViewCommand resource group, and the user has the seller administrator role in the current store's organization. In such cases, the command might then trigger a resource-level access control check if needed. In order to satisfy this check, a resource-level policy would have to grant access. Such a policy might state that seller administrators are only permitted to perform administrative tasks on resources that the organization for which the user is a seller administrator owns.

Instructor notes:

Purpose — This topic discusses the two types of access control in WebSphere Commerce.

Details — In command-level access control, the resource is the command itself and the action is to execute the command within the current store. The access control check determines whether the user is permitted to execute the command within the current store. By contrast, in resource-level access control, the resource is any protectable resource that the command or bean accesses and the action is the command itself.

Additional information — The XML information for the Conditions column of the MBRGRPCOND table is generated when you use the Administration Console to set up your access groups.

Transition statement — Move on to discuss the access control policies.

Access control policies

- Over 300 policies included OOB.
- Access control policies authorize access groups to perform particular actions on particular resources.
- All access control policies that are provided with WebSphere Commerce are assigned to 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. The policy group must be subscribed to by the organization that owns the resource.
- Unless authorized through one or more access control policies, users have no access to any functions of the system

© Copyright IBM Corporation 2016

Although organizations own access control policy groups, 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 an organization has child organizations, all policy groups the parent subscribes to are automatically applied to child organizations. However, if the child organization subscribes directly to a policy group, the policy groups that are subscribed to by the parent organization no longer apply to the child.

In previous versions of WebSphere Commerce, a policy that is 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. As of WebSphere Commerce V5.5, organizations can subscribe to policy groups.

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 in the command context. If there is no store in the command context, then Root Organization is used as the owner.

Access control policy utilities

- Access control records created in the database by utilities in WebSphere Commerce or directly through SQL:
 - `acugload` (load access groups).
 - `acupload` (load policy groups).
 - `acpnlsload` (load NLS descriptions).
 - `acextract` (extract policies and relationships).
- Policies are defined in a specific XML format:
 - See examples in <WCDE_HOME>\xml\policies\xml
 - Create XML files that conform to DTDs in \xml\policies\dtd
- Can load through the SAR publish process:
 - Edit `accesscontrol.xml` in \WEB-INF\stores\StoreAssetsDir\data
 - Descriptions via `accesscontrol_</language>.xml`
- View policies via the Organization Administration Console

© Copyright IBM Corporation 2016

The access control policy loading process reads the XML files that contain the access control policy information and access group definitions, and loads them into the appropriate databases. The policy and access group information that is in the XML files is loaded at installation; however, you must reload the files if they are changed.

Database user, who has read, write, and execute permissions updates the access control.

The SiteAdministratorsCanDoEverything policy is a special default policy that grants super-user access to administrators with the site administrator role. In this policy, a site administrator can act on any resource, even if those actions or resources are not defined. It is important to be aware of this fact when assigning this role to users.

BecomeUserCustomerServiceGroupExecutesBecomeUserCmdsResourceGroup policy is a special policy that allows certain administrative users to run specified commands on behalf of other users. This policy is needed, for example, when a customer requests a customer service representative to create an order on the customer's behalf. In this case, the customer service representative is able to run the command such that it appears as if the customer himself runs the command.

An important task for administrators is to keep multiple machines in sync; access control policies are no different. The acextract utility is crucial for extracting policy data for subsequent load onto another machine. There are two levels of authorization check:

- WebSphere Application Server
- WebSphere Commerce (fine-grained)

There are two types of policy-based control in WebSphere Commerce:

- Command-level (role-based, broad)
- Resource-level (resource-based, fine)

Controller commands require command-level access policies.

Instructor notes:

Purpose — The WebSphere Commerce command framework allows for fine granular access control. Policies are created in XML or SQL files that DB SQL tools or access policy tools that WebSphere Commerce provides load.

Details — Access control is based on the user's context, invoked actions, utilized resources, and the relationships within the stores and site implementation.

Additional information —

Transition statement — Now look at the implementation of access policies.

Access control, example policy

- Common change, adding a view to an action group procedure:
 - Create an action definition in the XML file that has the view name:
– <Action Name=" MyNewView CommandName=" MyNewView"> </Action>
 - Create an action group to be associated with the new role:
– <ActionGroupName=" XYZViews" OwnerID="RootOrganization"> </ActionGroup>
 - Associate the new action with the new action group:
– <ActionGroupAction Name=" MyNewView"/>

© Copyright IBM Corporation 2016

Example

We want to limit execution of a controller command to site administrators only, so we will define the following access control policy:

Place the policy file in the folder C:\IBM\WCDE80\xml\policies\xml and run command:

C:\IBM\WCDE80\bin\acupload.bat SiteAdministratorsExecuteMyCustomCommand.xml

File SiteAdministratorsExecuteCustomCommand.xml contents:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">
<Policies>
    <ResourceCategory
        Name="SiteAdministratorsExecuteMyCustomCommand_ResourceCategory"
        ResourceBeanClass="com.mycompany.MyCustomCmd">
        <ResourceAction Name="ExecuteCommand"/>
    </ResourceCategory>

    <ResourceGroup
        Name="SiteAdministratorsExecuteMyCustomCommand_ResourceGroup"
        OwnerID="RootOrganization">
```

```

        <ResourceGroupResource
Name="SiteAdministratorsExecuteMyCustomCommand_ResourceCategory"/>
</ResourceGroup>

<!--
ExecuteCommandActionGroup only needs to be referenced and not
defined,
it is provided OOTB.

<ActionGroup Name="ExecuteCommandActionGroup"
OwnerID="RootOrganization">

    <ActionGroupAction Name="ExecuteCommand"/>
</ActionGroup>
-->

<Policy

Name="SiteAdministratorsExecuteMyCustomCommand_Policy"
OwnerID="RootOrganization"
UserGroup="SiteAdministrators"

ActionGroupName="ExecuteCommandActionGroup"
ResourceGroupName="SiteAdministratorsExecuteMyCustom
Command_ResourceGroup"
PolicyType="groupableStandard">
</Policy>
</Policies>

```

Loading access control policy

- Create or copy access control policy file into the directory:
 - <WC_HOME>\xml\policies\xml
- Run the access control policy load program from:
 - <WC_HOME>\bin
 - Must be logged in as administrator.
 - Example:
 - Windows:

```
acupload.cmd database_name database_user database_user_password
policies_xml_file schema_name
```

 - > database_name Required: Name of the database in which to load the policy.
 - > database_user Required: Name of the database user who can connect to the database.
 - > database_user_password Required: The associated password for the database user.
 - > policies_xml_file Required: The input policy XML file that specifies what policy data to load into the database.
 - > schema_name Optional: The name of target database schema. This name is normally the same as database_user.
 - Check for errors in log file.
 - messages.txt
 - acupload.log

© Copyright IBM Corporation 2016

Loading an access control policy is simple. The steps include:

1. Create or copy the access control policy file that you created into the WebSphere Commerce home directory\xml\policies\xml directory.
2. Run the acupload script. This script loads the following elements:
<Action>, <ActionGroup>, <Attribute>, <ResourceCategory>,
<ResourceGroup>, <Relation>, <RelationGroup>, <Policy>,
<PolicyGroup>.
3. Check the error log file, messages.txt and acupload.log, for any errors.

Instructor notes:

Purpose — The loading process reads the XML files that contain the access control policy information and access group definitions and loads them into the appropriate databases.

Access control in controller commands

- Access control is enabled by extending the base ControllerCommand and ControllerCommandImpl classes that implement the AccCommand interface, which in turn extends Protectable.
- A simple implementation of the getResources() method follows:

```
public AccessVector getResources() throws ECException {  
    String methodName = "getResources";  
    ECTrace.entry(25L, getClass().getName(), "getResources");  
    ECTrace.exit(25L, getClass().getName(), "getResources");  
    return null;  
}
```

© Copyright IBM Corporation 2016

Implementing access control in BOD

1. Create a class that extends AbstractProtectableProxy, which implements the Protectable interface.
2. This mapping is defined in the wc-component.xml file.
3. Implement the following methods:
 - fulfills(Long member, String relationship)
 - getOwner()
4. Register the Protectable proxy class.
5. Create the access control policy XML file; start from here for the custom access profile example - default implementations already exist.
6. Load the access control policy and refresh policy registry.
 - The access profile indicates the view of the data and certain views can be limited to certain types of users.
 - Access Profile Access Control Policy (on Get requests only)
 - Resource Level Access Control Policy (on all Nouns)

© Copyright IBM Corporation 2016

Commands implement the business rules of your site.

There are two types of commands: Controller
commands

Implement business tasks such as user registration and allocation of inventory.

Task commands

implement discrete pieces of a business task such as address verification or ensuring
passwords comply to the defined password policies.

Command beans follow a Command design pattern.

Every command includes both an interface class and an implementation class

The key mechanisms enabled within this level of indirection include:

The ability to invoke an access control policy manager that determines if the user is
allowed to invoke the command.

The ability to execute a different command implementation for different stores, based upon the
store identifier.

Example access control policy which needs to be loaded for the custom assess profile example to work:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE Policies SYSTEM "..//dtd/accesscontrolpolicies.dtd">
<Policies>
    <Action Name="GetCatalogEntry.Sample_GetCatalogEntryDetails"
        CommandName="GetCatalogEntry.Sample_GetCatalogEntryDetails"/>
```

```
<!-- Action group and policy already defined OOTB, we just add our action to it. -->
<ActionGroup Name="Catalog-CatalogEntry-AllUsers- AccessProfileActionGroup"
OwnerID="RootOrganization">
    <ActionGroupAction
Name="GetCatalogEntry.Sample_GetCatalogEntryDetails"/>
</ActionGroup>

</Policies>
```

Exercise 2: Customizing the Business Logic Layer

© Copyright IBM Corporation 2016

What you learned

During this Unit, you learned:

- Name-Value Pair command processing
- Commands and the Command API
- Business Object Document processing
- Differences between NVPs and BODs
- Command behavior and Business Contexts
- Access control

© Copyright IBM Corporation 2016

This course is designed to enable you to understand the WebSphere Commerce version 8 business logic layer with introduction to Name- Value Pair command processing, access control, Commands and the Command API, Business Object Document processing, differences between NVPs and BODs, Command behavior and Business Contexts, and access control.

Additional training

For additional training needs, you have several options available for registering.

You can register	At
On site	This or any of our training facilities
Online	http://ibm.com/training/
Email	ibmswedu@us.ibm.com
By telephone	1-800-IBM Teach (1-800-426-8322)
By fax	208-692-6130

© Copyright IBM Corporation 2016

For additional training needs, you have several options available for registering like Onsite, Online , Email , by telephone or by Fax .

Unit 6

WebSphere Commerce V8 Persistence layer

© Copyright IBM Corporation 2016

This section describes an overview of the WebSphere Commerce Version 8 Persistence layer.

Unit objectives

This course is designed to enable you to:

- Understand the overview of WebSphere Commerce Data model
- Describe the implementation of EJBs in WebSphere Commerce
- Understand about extending the WebSphere Commerce data model
- Add custom SQL to existing session beans
- Create new session and entity beans to extend the WebSphere Commerce data model
- Develop and use access beans and data beans in WebSphere Commerce
- Perform the process of extending WebSphere Commerce by using EJBs

© Copyright IBM Corporation 2016

This course is designed to enable you to understand the WebSphere Commerce version 8 data model, describe the implementation of EJBs, and understanding about extending the WebSphere Commerce data model. Overview of session and entity bean, adding custom SQL to existing session beans, develop and use access beans and data beans in WebSphere Commerce and perform the process of extending WebSphere Commerce by using EJBs.

WebSphere Commerce data model

- WebSphere Commerce Data Model is designed for data integrity and persistence.
- WebSphere Commerce is a database-driven application:
 - 700+ database tables
 - 700+ EJB
- Tables exist to persist instance data, such as:
 - Organizational structure
 - Storefront operational data
 - Access control
 - Many others
- Tables:
 - Use many constraints
 - Use indexes to improve data retrieval.

© Copyright IBM Corporation 2016

To control the behavior of WebSphere Commerce, you control the data. For instance, if you want to get a price on a product or an item, the software retrieves data from the database to determine whether you have a certain contract or promotion. The price is retrieved based on the data.

Many tables and EJB beans come with the framework, so in most cases there are places to put data and ways to retrieve the data.

It is a best practice to use the existing data framework where possible.

Subsystem data models (1 of 2)

- The WebSphere Commerce Data Model is divided into many subsystems:
 - Catalog
 - Marketing
 - Promotions
 - Member
 - Order Management
 - Payment
 - Store
 - System
 - Trading
 - Content management
 - Content management system integration
 - Content Versioning
 - Folder
 - IBM Gift Center
 - WebSphere Commerce search engine optimization (SEO)
 - WebSphere Commerce search

© Copyright IBM Corporation 2016

Database data model displays the relationship among database tables in the schema. The data models are grouped into subsystems.

Database tables within the WebSphere Commerce database schema are organized into functional groups, called data models.

The WebSphere Commerce database model was designed for data integrity and optimal performance. WebSphere Commerce provides several hundred tables that store WebSphere Commerce instance data.

Understanding where to find the data model information is important. There is a reference section in knowledge center that gives good information on each of the subsystem data model that are mentioned on the slide.

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.database.doc/refs/rdb_datamodel_index.htm

In this course under data model unit most of the subsystem data model are discussed which will be a good topic to refer.

Subsystem data models (2 of 2)

- Catalog group data model
 - Relationship between database tables that contain information about catalog groups.
 - Tables : CATGROUP, CATGRPDESC
- ATP inventory data model
 - Relationship between database tables that contain information about ATP inventory.
 - Tables : ITEMFFMCTR, RECEIPT
- Content Management data model
 - The relationship between database tables that contain information about content management
 - Tables : CMMETADATA, CMWORKSPACE, CMFTSKMBREL
- WebSphere Commerce search engine optimization (SEO) data model
 - Relationship between database tables that contain information about WebSphere Commerce search engine optimization (SEO)
 - Tables : SEOPAGEDEF, SEOURLKEYWORD

© Copyright IBM Corporation 2016

This section covers few examples for data model. Catalog group data model:

Catalog group data model shows the relationship between database tables that contain information about catalog groups.

CATGROUP : This table hold the information related to a catalog group. A catalog group is similar to a generic category that can contain both other catalog groups and also catalog entries.

CATGRPDESC : This table holds the language-dependent information related to a catalog group.

ATP inventory data model:

The ATP (Available to promise) inventory data model shows the relationship between database tables that contain information about ATP inventory.

ITEMFFMCTR : Each rows contains information about reserved quantities, amount on backorder, and amount allocated to backorders for items owned by a store at a fulfillment center.

RECEIPT : Each row contains information about each receipt of an item at a FulfillmentCenter.

INVSTFFMVW : This is a view derived from the RECEIPT and ITEMFFMCTR tables which contains the existing quantity available for an item at a Store and FulfillmentCenter.

Content management data model:

The Content Management data model shows the relationship between database tables that contain information about content management.

CMMETADATA : Records the business objects that have changed and the associated workspaces, task groups, and tasks that have made that change. This table will be used for resource locking and for determining the business objects that will be committed and published to production.

CMWORKSPACE : Contains a list of valid workspaces that may be assigned to an available schema pool.

CMFTSKMBREL : Holds the task member-specific information. This relationship defines the tasks assigned to members. For example, workspaces contributor tasks, approval tasks will be stored in this table.

WebSphere Commerce search engine optimization (SEO) data model:

The WebSphere Commerce search engine optimization (SEO) shows the relationship between database tables that contain information about WebSphere Commerce search engine optimization (SEO).

SEOPAGEDEF : This table holds the SEO page definition for a particular pageName at a site/store level

SEOURLKEYWORD : This table holds the store and language specific SEO URL keyword information

Database tables

- Storing data about business objects, users, shoppers etc..
- References exist in the information center, for more information:
 - https://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/covreference.htm
 - Most customizations can be managed by using existing tables.
 - Tables have “empty” columns for simple custom extensions.
 - Most customizations can be supported by these empty columns.
 - These references are usually labeled with the word field and a number (such as field1, field2, field3).
 - These fields are accessible by using default EJB beans.
- Reference for Database tables
 - http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.database.doc/refs/rdb_database_overview.htm

© Copyright IBM Corporation 2016

Database tables:

The WebSphere Commerce database includes many database tables for storing data about business objects, users, shoppers, and more. Review the table and column descriptions for these database tables to understand the data that can be stored in the database.

From the reference location, you find a section called “WebSphere Commerce database schema.” There is a wealth of information under this subtopic.

There is a page under the section "reference" that has a link to the page: "Cross-reference of data beans, EJB beans, and tables." This section shows what tables are associated with what EJB beans.

All the tables are listed in the data models. Sometimes it is necessary to find the tables in the alphabetical list.

The customizable expansion fields are usually a preferred option when it is possible to use them. These fields, in most cases, are accessible by using the tools that come with Commerce.

For instance, you use an expansion field for a CATENTRY (product, item, and so on.). This information can be viewed through Commerce Accelerator for that CATENTRY.

This reference gives list of Database tables:

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.database.doc/refs/rdb_database_overview.htm

Each table can be clicked to get the definition and details of the table. Not all tables have expansion fields, but the most extended tables do.

The customizable fields also come in different data types. For instance, some are Strings and some are Integers.

Other references : http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.database.doc/concepts/cdb_schema_overview.htm

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rgxejb.htm

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rgxxref.htm

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rgxref.htm

Database schema changes from Version 7 to Version 8

- Tables and columns within V8 database schema can differ from V7
- Database schema changes from V7 to V8 are grouped into
 - Tables that are added since Version 7
 - Tables that are removed since Version 7
 - Deprecated tables
 - Tables that include new columns
 - Tables that include changed columns
 - Tables that include new or changed indexes
 - Table relationships that are new or changed
- Tables added in WebSphere Commerce Version 8
 - PLLOCSTATIC
 - PLPROPERTY

© Copyright IBM Corporation 2016

Database tables and columns within the WebSphere Commerce Version 8 database schema can differ from the tables and columns in the database schema for WebSphere Commerce Version 7.

The database schema changes from WebSphere Commerce Version 7 to WebSphere Commerce Version 8.0 are grouped into the following categories:

Tables that are added since Version 7
Tables that are removed since Version 7
Deprecated tables
Tables that include new columns
Tables that include changed columns
Tables that include new or changed indexes
Table relationships that are new or changed

To view the details about tables refer to knowledge center:

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.database.doc/refs/rdb_schema_changes70_80.htm

Tables added in WebSphere Commerce Version 8

PLLOCSTATIC : This table contains information about the static assignment of a layout to a user. Each PAGELAYOUT entry can have at most one entry in this table.

PLPROPERTY : This table contains information about the properties of a layout.

Implementing EJBs and extending the WebSphere Commerce data model

© Copyright IBM Corporation 2016

This section describes the implementation of EJBs in WebSphere Commerce and understand about extending the WebSphere Commerce data model.

Enterprise JavaBeans

- Component in Java EE.
- Provide a number of “built-in” services.
- Usually contains the Business Data Logic.
- Provides transaction support:
 - Pessimistic locking
 - Optimistic locking – WebSphere Commerce uses optimistic locking more often.
- Class instances are created and managed at run time by the container.
- Entity beans:
 - Used to access table data, has a model for data access.
 - The Entity bean contains a view of data that is stored in a database.
 - Example: shopping cart data.
- Session beans
 - Offer EJB services, no model for data access.
 - Example: users who have “wish lists”.
- WebSphere Commerce uses EJB 2.x modules, but most of the entity beans are at EJB 1.1 level

© Copyright IBM Corporation 2016

Enterprise JavaBeans (EJB) technology is the server-side component architecture for Java Platform, Enterprise Edition (Java EE). EJB technology enables rapid and simplified development of distributed, transactional, secure, and portable applications that are based on Java technology.

• Services examples include: *transactions, concurrency, security, and distribution*.

• Entity bean example: Shopping cart. An entity bean can be created to get a view of a shopping cart table. This EJB has a variable for each column of the table. The EJB has methods that allow access to the variables that represent the columns of the table.

• Session bean example: Users who have wish lists. The session bean can be used for getting data from multiple tables. In this example, the bean runs an SQL statement that retrieves information from multiple tables.

A transaction is a unit of work. The transaction or unit of work gives the application a greater level of data integrity. For instance, when you start a unit of work in WebSphere Commerce, it is usually a command, commands, or a command and view combination. While running a WebSphere Commerce command, you create many DBMS actions; if any unit of the transaction causes an uncaught exception, all other data is rolled back. When the rollback occurs, none of the DBMS actions are committed to the DB. (There are ways to cause a commit within a command, but it is not usually a good practice.)

Pessimistic locking: With this type of locking, when data is read with the intent of updating it, the data is locked until it is committed.

Optimistic locking: With this type of locking, multiple transactions can view the same data at the same time. The state of the data is confirmed before writing the data. (This could cause a transaction rollback.)

WebSphere Commerce uses EJB 2.x modules, but most of the entity beans are at EJB 1.1 level. IBM WebSphere Commerce Developer provides two extensions to the EJB 1.1 specification: EJB

inheritance and association. EJB inheritance allows an enterprise bean to inherit properties, methods, and method-level control descriptor attributes from another enterprise bean that resides in the same group. An association is a relationship that exists between two CMP entitybeans.

Types of EJB

- The different types of EJB and their uses in Commerce:
 - Entity bean
 - Container-Managed Persistence (CMP): Explicit data access must not be coded.
 - Bean-Managed Persistence (BMP): Bean writer must write the database access calls by using JDBC or SQLJ.
 - WebSphere Commerce uses CMP entity beans as a best practice.
 - Session bean
 - Stateless: Contains no conversational state between method calls.
 - Stateful: Manages conversational state between method calls.
 - WebSphere Commerce uses stateless session beans as a best practice

© Copyright IBM Corporation 2016

Entity beans are used in the persistence layer within WebSphere Commerce. The architecture is implemented according to the EJB component architecture. The EJB architecture defines two types of enterprise beans: entity beans and session beans.

Entity beans are further divided into container-managed persistence (CMP) beans and bean-managed persistence (BMP) beans.

Most of the WebSphere Commerce entity beans are CMP entity beans. A small number of stateless session beans are used to handle intensive database operations, such as performing a sum of all the rows in a particular column. One advantage of using CMP entity beans is that developers can utilize the EJB tools provided in WebSphere Commerce Developer. These tools allow developers to define Java objects and their database table mappings. The tools automatically generate the required persisters for the entity beans. Persisters are Java objects that persist Java fields to the database and populate Java fields with data from the database.

Bean-Managed Persistence (BMP) occurs when the entity object manages its own persistence. The enterprise bean developer must implement persistence operations (e.g., JDBC [Java Database Connectivity], JDO[Java Data Objects] , or SQLJ[SQL-Java]) directly in the enterprise bean class methods.

An entity bean with container-managed persistence relies on the container provider's tools to generate methods that perform data access on behalf of the entity bean instances.

The essential difference between an entity with bean-managed persistence and one with container-managed persistence is that in the bean-managed case, the data access components are provided as part of the entity bean, whereas in the container-managed case, the data access components are generated at deployment time by the container tools.

The conversational state of a stateful session object is defined as the session bean instance's field values, plus the transitive closure of the objects from the instance's fields that are reached by following Java object references.

Best practice: Extending the data model with EJB

- Create a table:
 - If you need additional data persisted, you can create a new table.
 - Example: expand shopping cart data or user registration information.
- Create an entity bean:
 - New table = new EJB
 - It is possible to use EJB inheritance in some cases.
 - Associations feature not recommended due to complexity.
- Create a Session bean:
 - Session beans are for read-only data.
 - In order to simplify migration in the future, you should not modify a default Session bean class.

© Copyright IBM Corporation 2016

Example: Expand Shopping Cart Data. If you need to persist more data than the Commerce shopping cart tables (ORDER, ORDERITEMS) permit, you can expand the data model by adding a table. This table should have a foreign key to the order table that is being extended. It is a good practice to ensure that the new table cascade is deleted when a record is removed.

Typically, when you create a new table, you create a new CMP entity bean. The EJB inheritance will give better performance, but it will be more difficult to upgrade in the future.

The expanded shopping cart table can be accessed for read and write by creating any of the new types of entity beans.

Best practice: Ways to extend the data model

- If the new table is logically an extension of an existing table, create a foreign key relationship between them.
- Create a new entity bean for the new table.



© Copyright IBM Corporation 2016

Overview of session and entity beans

© Copyright IBM Corporation 2016

This section describes an overview of session and entity beans.

Why extend?

- Why extend the Data Model and create an Entity bean?
 - Existing tables do not have enough expansion fields.
 - Using existing fields does not make sense.
 - Creating a new subject of data:
 - Example: A DBA can create a table to persist customer review data.
- Why create a Session bean?
 - Complex searches (for example: SQL Intersections)
 - Complex SQL (table joins)
 - Complex results-handling
 - Processing the data after the results are returned.

© Copyright IBM Corporation 2016

Example: A database administrator might create a table to persist customer review data. You could probably find a place to store this information in the WebSphere Commerce data model, but it might make more sense to create a new table. (There is no out of box subsystem for this scenario.)

Session beans: Session beans are beneficial for complex searches that might contain table joins and intersections. Session beans are also useful for searches that contain hierarchical SQL statements.

WebSphere Commerce provides two sets of enterprise beans: private and public. WebSphere Commerce runtime environment and tools use private enterprise beans. You must not use or modify these beans.

Working with new session beans

- When should you use a new session bean?
 - Large result set : When a query returns a large result set.
 - Aggregate entity : When a query retrieves data from across several, disparate tables.
 - Arbitrary SQL :When an SQL statement performs a database-intensive operation.
 - When you need access to read-only data, with no intention to commit.

© Copyright IBM Corporation 2016

Below are the situations in which it is recommended to use a session bean JDBC helper.

- A case where a query returns a large result set. This is referred to as the *large result set* case.
- A case where a query retrieves data from several tables. This is referred to as the *aggregate entity* case.
- A case where an SQL statement performs a database intensive operation. This is referred to as the *arbitrary SQL* case.

Large result set case:

There are cases where a query returns a large result set and the data retrieved are mainly for read or display purpose. In this case, it is better to use a stateless session bean and within that session bean, create a finder method that performs the same functions as a finder method in an entity bean. That is, the finder method in the stateless session bean should do the following:

Perform an SQL select statement

For each row that is fetched, instantiate an access bean

For each column retrieved, set the corresponding attributes in the access bean

Aggregate entity case:

In this case, one view combines parts of several objects and a single display page is populated with pieces of information that come from several database tables. For example, consider the concept of "My Account". This may consist of information from table of customer information (for example, the customer name, age and customer ID) and information from an address table (for example, an address made up of street and city).

Arbitrary SQL case:

In this case, there is a set of arbitrary SQL statements that perform database-intensive operations. For example, the operation to sum all the rows in a table would be considered a database intensive operation. It is possible that not all of the selected rows correspond to an entity bean in the persistent model.

An example that could result in the creation of an arbitrary SQL statement is a when a customer tries to browse through a very large set of data.

Note : Note that if the session bean is being used as a JDBC wrapper to retrieve information from the database, it becomes more difficult to implement resource-level access control. When a session bean is used in this manner, the developer of the session bean must add the appropriate " where" clauses to the " select" statement in order to prevent unauthorized users from accessing resources.

Reference: http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/csdxextendwithsessionbean.htm

Creating new session beans

- How to create a new Session bean
 - Extend BaseJDBCHelper
 - This "helper" class has many methods for JNDI and JDBC operations.
 - Configure options
 - Configure Access Control
 - Write SQL
 - Parse results
 - Use uncommitted read when performing a read in your SQL.
 - Add Access bean (This process is explained in more detail later).

© Copyright IBM Corporation 2016

WebSphere Commerce uses access beans in its programming model, and access beans do not support local interfaces. To use local interfaces, instead of using access beans, you must perform home lookup directly and cache it yourself for performance purposes. Your new session bean should extend the com.ibm.commerce.base.helpers.BaseJDBCHelper class. The superclass provides methods that allow you to obtain a JDBC connection object from the data source object used by the WebSphere Commerce Server, so that the session bean participates in the same transaction as the other entity beans.

Here JNDI refers to Java Naming and Directory Interface and JDBC refers to Java Database Connectivity.

For more details refer to knowledge center:

http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/tasks/tsdcreatesessionbean.htm

Here is an example of an SQL statement with an uncommitted read (UR):

```
(select * from users with uncommitted read )
```

A session bean can be more efficient than an entity bean. For instance, it is faster and less resource intensive to use SQL to order data that comes from the DB. This SQL can be used in a session bean.

It is more efficient to use a session bean if you want to return a row count only. Example: (select count (*) from users)

Use the wizard provided within Rational Application Developer under developer toolkit to create the necessary classes for the new session bean. When creating new session beans, create them in the WebSphereCommerceServerExtensionsData project.

For more information refer http://www.ibm.com/support/knowledgecenter/SSRTLW_7.5.5/com.ibm.tools.ejb.doc/topics/tesessb.html?view=embed

Extending a session bean with custom SQL

- Recommended for database searches only.
- ServerJDBCHelper session bean (with access bean) is useful for this task.
 - Returns a Vector of rows that contain a Vector of columns.

```
public Integer getUserCount(){  
    ServerJDBCHelperAccessBean myHelper = new ServerJDBCHelperAccessBean();  
    Integer count = new Integer();  
    try{  
        StringBuffer sql1 = new StringBuffer();  
        sql1.append("select count(*) from users with ur");  
        Vector records1 = myHelper.executeQuery(sql1.toString());  
        count = (Integer)((Vector)records1.elementAt(0)).elementAt(0);  
    } catch (Exception ex) {  
        //Code Exception here  
    }  
    return count ;  
}
```

© Copyright IBM Corporation 2016

This approach returns a vector of vectors. The first vector represents the rows that returned from the bean. The second vector represents each of the columns in the row that is returned. This is already in the software stack. Ensure that you know the data types that are returned while using the vectors. You must type cast the objects that are returned from the vector return methods.

For instance, the result set integer gets typecast to a Java Integer.

There is a session bean in the software stack that allows you to set an SQL statement, and it returns a result set as a vector of vectors.

There are methods on this bean that allow SQL updates. It is not recommended to use these methods in a production system. Entity beans are better for updating data.

This method is not necessarily the most efficient way to retrieve data, since you cannot control how result set primitives are type cast. But this approach helps speed up the development time, since a session bean does not have to be created and deployed.

Working with new entity beans

- When should you use a new entity bean?
 - When you need to read and write data to a table.
 - When you are adding data row to a table.
- Most of Commerce Entity beans are at EJB 1.1 level.
 - All new extension WebSphere Commerce Entity beans should be at level 1.1.
- EJB 2.1 level:
 - Current limitations:
 - All beans that the EJBQ references must be at level 2.1.
 - EJBQ cannot reference beans at level 1.1.
 - Access beans do not support local interfaces and container-managed relationships.
 - Although the use of EJB 2.x Entity beans in WebSphere Commerce is possible, it is not recommended.

© Copyright IBM Corporation 2016

When developing EJB beans for WebSphere Commerce, the EJB projects are at EJB specification level 2.1. When creating EJB beans in the project, you should choose the EJB specification level of 1.1.

-*Local interfaces* and *Container-Managed Relationships*, which are the two main features that are provided by the EJB 2 specification, does not benefit WebSphere Commerce users for the following reasons:

-**Local interfaces:** The performance increase gained by local interfaces is already obtained through the WebSphere Application Server feature that allows pass-by-reference. For more information on pass-by-reference, see the Object Request Broker tuning guidelines topic in the WebSphere Application Server informationcenter.

-**Container-managed relationships:** To minimize complexity in the object model, rather than using the container-managed relationships, an object relationship between enterprise beans can be established by adding explicit getter methods to the enterprise beans that return access beans.

-**WebSphere Commerce Developer provides two extensions to the EJB 1.1 specification:** EJB inheritance and association. EJB inheritance allows an enterprise bean to inherit properties, methods, and method-level control descriptor attributes from another enterprise bean that resides in the same group. An association is a relationship that exists between two CMP entity beans. EJB inheritance allows an enterprise bean to inherit properties, methods, and method-level control descriptor attributes from another enterprise bean that resides in the same group. An association is a relationship that exists between two CMP entity beans.

Create a CMP entity bean

- High-level steps to create a new entity bean:
 1. Create the bean.
 2. Set the bean's Isolation Level.
 3. Set the Security Identity of the bean.
 4. Set the Security Role of the methods in the bean.
 5. Remove and add some methods.
 6. Add FinderHelpers methods as needed.
 7. Add new ejbCreate() and ejbPostCreate() methods
 8. Add new getOwner() and fulfills() methods.
- WebSphere Commerce specific:
 - All new Entity beans should extend ECEntityBean.
 - All new Entity beans should implement protectable.
 - ECKeyManager should be used in ejbCreate() to track Primary Keys.

© Copyright IBM Corporation 2016

Online documentation for creating a new entity bean is at the following website:

https://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/tasks/tdecreateentitybean.htm

The entity bean must extend the commerce class ECEntityBean. ECKeyManager is used to maintain the key list from the KEYS table.

When you create a new row in a table, you need to create a primary key. The ECKeyManager class helps you to track the last key. The increments of the keys are configurable in the table. The code that you need to leverage the ECKeyManager is:

```
myKeyValue = com.ibm.commerce.key.ECKeyManager.singleton().getNextKey("table_name");
```

Protectable interface: *"A key factor for having a resource that the WebSphere Commerce access control policies protect, is that the resource must implement the com.ibm.commerce.security.Protectable interface. This interface is most commonly used with enterprise beans and data beans, but only those particular beans that require protection need to implement the interface."*

The following are more detailed steps for creating an entity bean:

1. Create the CMP entity bean by using the Enterprise Bean Creation wizard. For each column in the corresponding database table, add a CMP field to the bean.
2. Set the transaction isolation level for the new bean.
3. Set the security identity and role of the new bean.
4. Set the optimistic locking option of the bean.
5. Modify the entity context fields and methods.
6. Modify the ejbLoad and ejbStore methods.

7. If required, define new finders by using the EJB deployment descriptor editor.
 8. Create an ejbCreate method, if required, and promote the ejbCreate method to the home interface of the enterprise bean. This step is required if the new enterprise bean must create new entries in its corresponding database table.
 9. Initialize optimistic locking fields in ejbCreate
10. Create an ejbPostCreate method.
11. If the WebSphere Commerce access control system protects the bean, implement the required access control methods in the bean. Refer to *Understanding access control* for more details about implementing access control in enterprise beans. Optionally, you can implement access control after you create your accessbean.
12. Map the fields in the enterprise bean to the columns in the database table.
13. Modify the schema name.
14. Generate the corresponding access bean for the enterprisebean.
15. (Optional) Use the optimistic locking migration plug-in to enable optimistic locking in your new entity bean, if you did not already manually enable optimistic locking. For more information, see coding practices.
16. Generate the deployed code for the enterprise bean.

Implementing access control in EJB

1. Create an enterprise bean, extending **ECEntityBean**.
2. Remote interface must extend protectable interface:
 - If a resource is going to be grouped by an attribute other than its Java class name for applying Access Control Policies, the remote interface of the bean must also extend **Groupable**.
3. Implement the following inherited methods:
 - **getOwner()**
 - **fulfills()**
 - **getGroupingAttributeValue()**
4. Create or re-create the enterprise bean's access bean and generated code.

© Copyright IBM Corporation 2016

Override any methods that you need. You must at least override the **getOwner** method.

The **fulfills** method must be implemented if an access control policy includes this resource in its resource group, and specifies a relationship or relationship group. The **getGroupingAttributeValue** method must be implemented if there is an access control policy with an implicit resource group that includes certain instances of this resource, based on specific attribute values (for example, if an access control policy pertains only to the Orders with status = 'P' (pending)).

Note: If the only relationship needed is owner, then you do not need to override the **fulfills** method. In this case, the policy manager uses the result of the **getOwner()** method.

The default implementations of these methods are shown in the following code snippets. These implementations come from the **ECEntityBean** class.

```
public Long getOwner() throws Exception { return null; }

}

public boolean fulfills(Long member, String relationship) throws Exception {
    return false;
}

public Object getGroupingAttributeValue(String attributeName,
GroupingContext context) throws Exception {
    return null;
}
```

The following are sample implementations of these methods based on the implementations that are used in the **OrderBean** bean:

For the **getOwner** method, the logic of the provided method is:

```

com.ibm.commerce.common.objects.StoreEntityAccessBean storeEntAB = new
com.ibm.commerce.common.objects.StoreEntityAccessBean(
);
storeEntAB.setInitKey_storeEntityId(getStoreEntityId()
.toString());
return storeEntAB.getMemberIdInEJBType();

```

For the fulfills method, the logic of the provided method is:

```

if ("creator".equalsIgnoreCase(relationship)) { return
    member.equals(bean.getMemberId());
} else if ("BuyingOrganizationalEntity".equalsIgnoreCase(relationship)) {
    return (member.equals(bean.getOrganizationId()));
} else if ("sameOrganizationalEntityAsCreator".
equalsIgnoreCase(relationship)) {
    com.ibm.commerce.user.objects.UserAccessBean creator = new
com.ibm.commerce.user.objects.UserAccessBean();
    creator.setInitKey_MemberId(bean.getMemberId(
).toString());
    com.ibm.commerce.user.objects.UserAccessBean
ab = new com.ibm.commerce.user.objects.UserAccessBean();
    ab.setInitKey_MemberId(member.toString());
    if (ab.getParentMemberId().equals(creator.getParentMemberId()))
return true;
}
return false;

```

For the **getGroupingAttributeValue** method, the logic of the provided method is:

```

if (attributeName.equalsIgnoreCase("Status")) return getStatus();
return null;

```

If you create a new data bean that an access control policy must directly protect, the data bean must do the following tasks:

- Implement the com.ibm.commerce.security.Protectable interface. As such, the bean must provide an implementation of the getOwner() and fulfills(Long member, String relationship) methods.

When a data bean implements the Protectable interface, the data bean manager calls the isAllowed method to determine whether the user has the appropriate access control privileges, based on the existing access control policies. The following code snippet describes the isAllowed method:

```

isAllowed(Context, "Display", protectable_databean);
    where protectable_databean is the data bean to beprotected.

```

- If resources that the bean interacts with are grouped by an attribute other than the resource's Java class name, the bean must implement the com.ibm.commerce.grouping.Groupable interface.

- Implement the com.ibm.commerce.security.Delegator interface.

The following code snippet describes this interface:

```

Interface Delegator {Protectable getDelegate(); }

```

The distinction between which data beans should be protected directly and, which should be protected indirectly is similar to the distinction between primary and dependent resources. If the data bean object can exist on its own, it should be directly protected. If the existence of data bean depends

upon the existence of another data bean, then it should delegate to the other data bean for protection.

An example of a data bean that would be **directly protected** is the **Order data bean**. An example of a data bean that would be **indirectly protected** is the **OrderItem data bean**.

If you create a new data bean that an access control policy should indirectly protect, the data bean must do the following tasks:

- Implement the com.ibm.commerce.security.Delegator interface.** The following code snippet describes this interface:

```
Interface Delegator {Protectable getDelegate();}
```

Note: The data bean returned by getDelegate must implement the Protectable interface.

If a data bean does not implement the Delegator interface, it is populated without the protection of access control policies.

Overview of access and data beans

© Copyright IBM Corporation 2016

This section describes an overview of access and data beans.

Access beans (Extensions to the EJB specification)

- What are access beans?
 - Objects that provide more simple programming interface:
 - Simple-to-use, Java-like beans that wrap EJB beans.
 - Behave more like a Java bean.
 - Hide all Enterprise bean-specific programming interfaces, like JNDI.
 - Mask complex home and remote interfaces from the clients.
- How do access beans work?
 - At run time, the Access bean caches the Enterprise bean home object.
 - Lookups to the home object are expensive, in terms of time and resource usage; caching saves time and resources.
 - Access can be faster than direct access of EJB beans by reducing number of remote calls.

© Copyright IBM Corporation 2016

WebSphere Commerce commands interact with access beans rather than directly with entity beans. EJB access beans can greatly simplify client access to enterprise beans and alleviate the performance problems that are associated with remote calls for multiple enterprise bean attributes.

Access beans are Java bean representations of enterprise beans. In WebSphere Commerce, access beans are used in Controller commands and Task commands. Access beans shield you from the complexities of managing enterprise bean lifecycles. This means that you can program to enterprise beans as easily as you can program to Java beans. This greatly simplifies your enterprise bean client programs and helps reduce your overall development time.

• An **EJB factory** is an access bean that simplifies the creating or finding of an enterprise bean instance. It exposes the create and finder methods on the remote home interface. The factory is not available for local home interfaces.

• A **data class access bean** is an implementation of what is often referred to as a value object or data transfer object. It is a container that holds a local copy of selected entity bean attributes.

• The **Java bean wrapper** is designed to allow either a session or entity bean to be used like a standard Java bean, and it hides the enterprise bean home and component interfaces from the developer.

• A **copy helper access bean** has all of the characteristics of a Java bean wrapper, but it also maintains a local copy of attributes from a remote entity bean.

• The access bean method is much faster than direct access of EJB beans. The home is cached in the

access bean and other caching is performed. The access bean method also allows developers with noEJB knowledge to write servlets and JSPs that can access Enterprise JavaBeans and display their properties.

By modifying the enterprise bean implementation, the number of remote calls that go over the network can be greatly reduced, significantly improving performance. This modification, however, causes incompatibility with the other options for accessing EJB beans, locking into a single implementation.

Another disadvantage happens when an access bean is involved in an association made navigable. The navigation method of the access bean returns an access bean corresponding to the enterprise bean at the other side of the association. This requires the developer to generate access beans for all the enterprise beans that are related through associations, as errors occur when following associations through access bean methods.

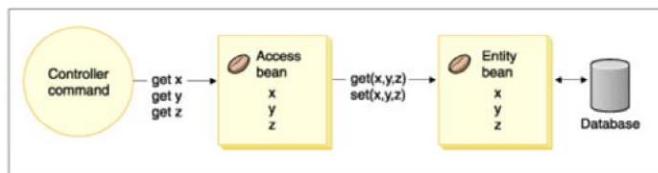
A Redbook that explains this concept:

<http://www.redbooks.ibm.com/abstracts/sg246819.html?Open>

The Java bean wrapper hides the enterprise bean home and remote interfaces from you. The copy helper has the basic characteristics of a Java bean wrapper, but it also incorporates a single copy helper object that contains a local copy of attributes from a remote entity bean.

Access beans in WebSphere Commerce

- Behave like a Java bean.
- Hide EJB interfaces (JNDI, home, remote).
- Cache the EJB home object (expensive).
- Implement a copyHelper object.
- Used primarily within commands (a few JSPs).



© Copyright IBM Corporation 2016

The above diagram displays the interaction between commands, access beans, entity beans, and the database.

Access beans behave like a Java bean which is the representation of enterprise beans. A program that uses enterprise beans must deal with the Java

Naming and Directory Interface (JNDI) as well as the home and remote interfaces of enterprise beans. To simplify the programming model, an access

bean for each enterprise bean is generated. There are three types of access beans: Data class, Copy helper, and bean wrapper. We use the Copy helper

for Entity beans and we use the Java bean wrapper for Session beans.

When you create your own enterprise beans, you can use WebSphere Commerce

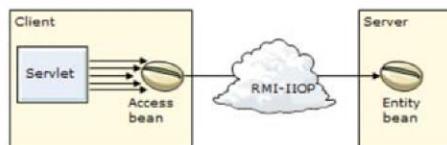
Developer to generate this access bean. Access beans are used in Controller commands and Task commands.

Note: On the access bean, the get and set methods are not automatically cached. Only the methods that are part of the copy helper object are cached.

The getters and setters on the remote interface are invoked when called.

Types of access beans

- Types of access beans:
 - EJB factory
 - Data class (for an Entity bean)
 - JavaBeans wrapper (for a Session or Entity bean)
 - CopyHelper (most common, used for an Entity bean)
- Example of CopyHelper:
 - CopyHelper keeps a cached copy of home object



© Copyright IBM Corporation 2016

The image on the slide is for a CopyHelper access bean. The data class access bean would have the access bean itself moved between the client and server. RefreshCopyHelper populates the access bean attributes with the EJB attributes. (This process retrieves the data from the database.)

CommitCopyHelper populates the EJB with the attributes of the access bean. (This **process** moves data to the database.)

Even when the EJB client and server are on the same machine, there is a network communication between the access bean and entity bean.

Access bean methods

- Methods that synchronize access beans:
 - RefreshCopyHelper (use this method before using the bean).
 - Refreshes bean with latest data from database.
 - CommitCopyHelper (use this method to commit changes).
 - Updates database with changes made to Access bean.
- Finder methods:
 - findByLastUpdate (you can create custom finder methods).
- Getter and Setter methods (methods for accessing attributes).

© Copyright IBM Corporation 2016

RefreshCopyHelper populates the access bean attributes with the EJB attributes. (This process retrieves the data from the database.)

CommitCopyHelper populates the EJB bean with the attributes of the access bean. (This process moves data to the database.)

These EJB beans use a two-phase commit. The CommitCopyHelper commits the state to the EJB bean, but the data in the database is not committed until the second phase of the commit occurs. (This is when the transaction is committed.)

The access bean implements a copyHelper object that reduces the number of calls to the enterprise bean when commands get and set enterprise bean attributes. Therefore, only a single call to the enterprise bean is required, when reading or writing multiple enterprise bean attributes.

Access beans may also provide specific finder methods for commonly needed "find" operations. These methods will return you an Enumeration of access beans that match the find criteria. If records are found, there will be one or more fully populated access beans in the Enumeration. If the result of the finder is empty, the finder method will throw

`javax.ejb.ObjectNotFoundException`.

For example, in the following code sample, if no users are found with the specified nick name, `ObjectNotFoundException` is caught and logged.
`AddressAccessBean abAddress = new AddressAccessBean().findByNickname("nickname", new Long("10001"));` Where `nickname` is the search term used for the nick name.

Create an access bean

- 1. Click File > New > Access Bean**
 - It is also possible to operate this in the deployment descriptor editor for the EJB project.
- 2. Select the bean type (Copy helper).**
- 3. Select your EJB project.**
- 4. Select the Entity bean to wrap.**
- 5. Select the Constructor method.**
 - `findByPrimaryKey(yourPackageName.yourNewBeanKey)`.
- 6. Select attributes in the Attribute Helpers section.**
- 7. Finish and save.**
- 8. Enter JNDI name under JNDI - CMP Connection Factory Binding section**
- 9. Save the changes**

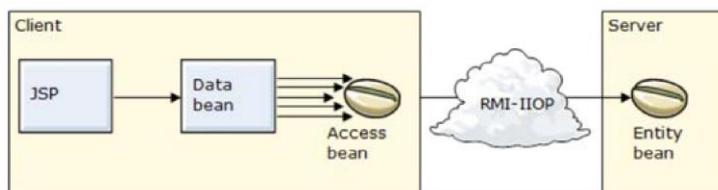
© Copyright IBM Corporation 2016

To create access bean follow below steps:

1. Click File > New > Access Bean.
2. In the Add an Access Bean window, select Copy helper and click Next.
3. From the EJB Project list, select your EJB project.
4. Select the yourNewBean bean and click Next.
5. From the Constructor method drop-down list, select `findByPrimaryKey(yourPackageName. yourNewBeanKey)` as the constructor method.
6. Select all attributes in the Attribute Helpers section.
7. Click Finish.
8. Save your work.
9. In the EJB Deployment Descriptor Editor, in the Overview tab, scroll down to the JNDI - CMP Connection Factory Binding section and in the JNDI name field enter the value that matches your database type.
10. Save your changes.
11. Proceed to generating the deployed code Reference:
http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/tasks/decreateentitybean_12.htm

Data beans (1 of 2)

- What is a Data bean?
 - Used on JSPs to provide Dynamic Content.
 - Allows easy access to data.
 - Extend Access beans (Data beans are Access beans that are used for presentation purposes).
 - For read-only data.
 - Gather data only when required (“lazy fetch”).



© Copyright IBM Corporation 2016

A data bean is a Java bean that is used in JSP pages to retrieve information from the enterprise bean. A simple data bean extends its corresponding access bean and implements the SmartDataBean interface. By extending an access bean, the data bean provides a simple indirect representation of an entity bean: it encapsulates the properties that can be retrieved from or set within the entity bean.

When the EJB is modified, the access bean and data bean inherits the changes.

Data beans can have methods that get other data. For instance, a ProductDataBean can extend an entity bean that contains information about the product. The ProductDataBean can have a method that retrieves items of that product. The method populates the data when it is called.

DataBeanManager activates the data bean. Typically, this process populates the attributes in the data bean by refreshing the underlying EJB.

- **SmartDataBean:** A data bean that implements the SmartDataBean

interface can retrieve its own data, without an associated data bean command. A smart data bean usually extends from the access bean of a corresponding entity bean. When a smart data bean is activated, the data bean manager invokes the data bean's populate method.

- **CommandDataBean:** A data bean that implements the CommandDataBean interface retrieves data from a data bean command. A data bean of this type is a lightweight object; it relies on a data bean command to populate its data.

- **InputDataBean:** A data bean that implements the InputDataBean interface retrieves data from the URLparameters or attributes that the view sets.

Data beans (2 of 2)

- Types of Data beans:
 - SmartDataBean
 - Retrieves its own data by using "lazy fetch".
 - InputDataBean
 - Retrieves data from URL parameters or view attributes.
 - CommandDataBean
 - Relies upon a command to retrieve data.
- Using a Data bean
 - Activated by DataBeanManager class.
 - *wcbase:usebean*

```
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wcbase" %>
<wcbase:useBean
    id="orderBean"
    classname="com.ibm.commerce.order.beans.OrderDataBean"
    scope="page"/>
```

© Copyright IBM Corporation 2016

A smart data bean uses a *lazy fetch* method to retrieve its own data. This type of data bean can provide better performance in situations where not all data from the access bean is required, since it retrieves data only as required. Smart data beans that require access to the database should extend from the access bean for the corresponding entity bean and implement the com.ibm.commerce.SmartDataBean interface.

For example, the ProductDataBean data bean extends the ProductAccessBean access bean, which corresponds to the Product entity bean.

Some smart data beans do not require database access. For example, the PropertyResource smart data bean retrieves data from a resource bundle, rather than the database. When database access is not required, the smart data bean should extend the SmartDataBeanImpl class.

Attributes that are defined with InputDataBean interface can be used as primary key fields to fetch additional data. When a JSP page is invoked, the generated JSP servlet code populates all the attributes that match the URL parameters, and then activates the data bean by passing the data bean to the data bean manager. The data bean manager then invokes the data bean's setRequestProperties() method (as defined by the com.ibm.commerce.InputDataBean interface) to pass all the attributes that the view set. The following code is required in order for the data bean to be activated:

```
com.ibm.commerce.beans.DataBeanManager.activate(data_be an, request, response)
```

where *data_be an* is the data bean to be activated, *request* is an HttpServletRequest object, and *response* is an HttpServletResponse object.

A command data bean relies on a command to retrieve its data and is a more lightweight data bean. The command retrieves all attributes for the data bean at once, regardless of whether the JSP page requires them. As a result, for JSP pages that use only a selection of attributes from the data bean, a command data bean might be costly in terms of performance time. While access control can be

enforced on a data bean level when using the smart data bean, this is not true for command data bean. Use only the command data bean if using a smart data bean is impractical.

Create a data bean

1. Right-click the package into which you store the data bean and select **New > Class**.
2. Name the new data bean.
3. Select public modifiers.
4. Select the appropriate superclass.
 - Data beans should extend an identical Access bean.
 - For example, ProductDataBean should extend ProductAccessBean.
5. Select the interface to implement:
 - com.ibm.commerce.beans.SmartDataBean
 - com.ibm.commerce.beans.InputDataBean
 - com.ibm.commerce.beans.CommandDataBean
6. Finish and save.

© Copyright IBM Corporation 2016

Create a data bean that extends the corresponding access bean and implements the appropriate data bean interface.

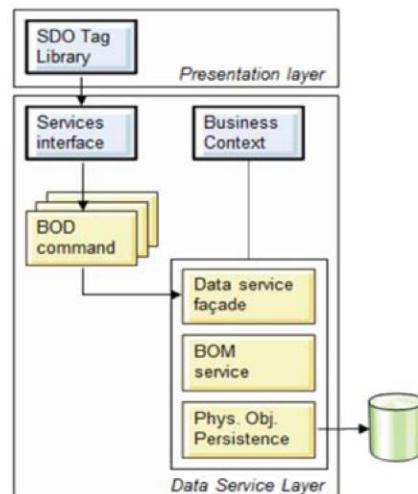
Procedure:

- 1) Right-click the package into which you will store the data bean and select New > Class.
- 2) In the New Java Class wizard, the project and package name fields are already populated. In the Name field, enter a name for your new data bean. For example, to create a data bean that extends the UserResAccessBean, enter UserResDataBean.
- 3) From the Modifiers list, select public.
- 4) To specify the superclass, click Browse, then in the pattern field, enter the name of the corresponding access bean. For example, enter UserResAccessBean and click OK.
- 5) To specify the interfaces that the data bean should implement, click Add. In the Interface window:
 - a. In the Pattern field, enter com.ibm.commerce.beans.SmartDataBean then click Add.
 - b. In the Pattern field, enter com.ibm.commerce.beans.InputDataBean then click Add. c. Click OK.
- 6) Click Finish.

Reference : https://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/tasks/tdecreatedatabase_2.htm

Persistence layer: Data service layer

- Working with EJB beans closely couples persistence layer to business logic layer.
- Using the data service layer is the recommended best practice for implementing persistence in an SOA solution.
- Data service façade single point of entry, accepts BOD commands.
- Business object mediation transforms logical object (BOD) into physical object (row).
- Physical persistence service translates XPath queries into SQL statements.
- Discussed later in the course.



© Copyright IBM Corporation 2016

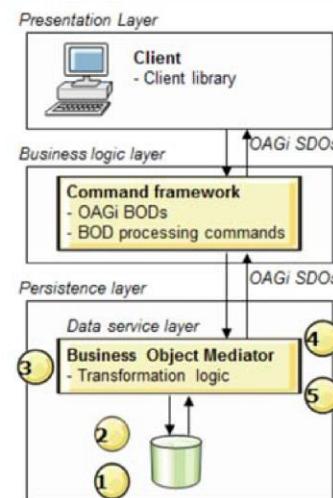
The Data Service Facade is a thin layer that provides a single entry point into DSL. It provides interfaces to work with both physical and logical data, and it delegates to the business object mediation service or to the physical persistence service. It also allows each service module to register with the data service layer, and loads the service module-specific configuration files.

For read operations, the data service layer facade receives a query from the business logic layer. The query consists of an XPath expression and an access profile, which are extracted from an OAGi GET verb. The data service layer forwards this query to the business object mediator (BOM) who, in turn, passes it to the persistence service. That service looks up the correct SQL template for the query, and uses it to generate one or more SQL statements. It then runs these statements, and maps their result sets into physical SDOs. This mapping, between the database schema (tables and columns) and the SDO classes, is defined by XML called object-relational metadata. Finally, the physical SDOs are returned to the BOM. The BOM configuration describes how to instantiate the necessary mediators. These mediators are returned to the businesslogic layer. It is the mediator who is returned, not just the SDO. The mediator contains the physical SDO data. It also contains the logic to convert the physical SDO to the logical SDO (which is the Java representation of an OAGi noun).

For change operations (create, update, delete), the data service layer facade receives the OAGi nouns as input, and passes them to the business object mediation service. This service instantiates the appropriate change mediators. In turn, they fetch the physical representation of the nouns from a service called the physical object mediation service. The BOM then returns the mediators to the business logic layer. The mediators are then called with specific actions to create, update, or delete nouns and noun parts. The logic inside the mediators translates these actions into operations on physical SDOs. For example, a create request creates new physical objects and populate them with noun values. After making all its changes, the business logic layer instructs the change noun mediator to save the updated physical SDOs. The mediator calls a physical object persistence service to update the database.

Customizing persistence through the data service layer

1. Add new tables to database.
2. Generate custom metadata that describes new tables.
3. Generate static service data objects (SDO).
 - Java representation of new tables
4. Configure logical data object to physical data object mapping.
5. Create a custom query template file. The template file:
 - a) Defines SQL template that fetches data from new tables.
 - b) Associates new access profile to new SQL template statement.
 - c) Associates XPath expression to new SQL template statement.



© Copyright IBM Corporation 2016

You can extend the existing schema to support customization. These extensions include changing the physical database schema, generating the object-relational metadata and physical SDOs, and by using query templates to retrieve the new information.

In order to customize persistence through the data service layer:

1. Update the WebSphere Commerce database with new tables and relationships.
2. Generate custom object-relational metadata that describes the new tables and relationships.
3. Generate static service data objects (SDO) that provide a Java representation of the newly added tables. The term "static SDOs" and "physical data objects" are used interchangeably.
4. Configure the logical data object to physical data object mapping.
5. Create a custom query template file to map an XPath expression and access profile to SQL. This template file:
 - a. Defines a SQL template statement that fetches data from the new tables.
 - b. Associates a new access profile to the new SQL template statement.
 - c. Associates a XPath expression to the new SQL template statement.

The WebSphere Commerce Data Service Layer wizard generates the physical SDO Java classes for your customizations, along with the required object-relational metadata and configuration for business object mediators. These steps are required if you add a table, and also for any new service modules

you develop. XML assets that are generated are stored in the service module configuration extension directories and custom physical SDO Java classes are stored inside the WebSphereCommerceServerExtensionsLogic project.

Unit summary

This unit was designed to enable you to:

- Provide a high-level description of the WebSphere Commerce data model
- Describe the implementation of EJBs in WebSphere Commerce
- Understand about extending the WebSphere Commerce data model
- Add custom SQL to existing session beans.
- Create new session and entity beans to extend the WebSphere Commerce data model.
- Develop and use access beans and data beans in WebSphere Commerce.
- Perform the process of extending WebSphere Commerce by using EJBs.

© Copyright IBM Corporation 2016

Additional training

For additional training needs, you have several options available for registering.

You can register	At
On site	This or any of our training facilities
Online	http://ibm.com/training/
Email	ibmswedu@us.ibm.com
By telephone	1-800-IBM Teach (1-800-426-8322)
By fax	208-692-6130

© Copyright IBM Corporation 2016

For additional training needs, you have several options available for registering like Onsite, Online , Email , by telephone or by Fax .

Unit 7

WebSphere Commerce V8 BOD Command Processing and Data service layer

© Copyright IBM Corporation 2016

This section describes an overview of the WebSphere Commerce Version 8 BOD Command processing and data service layer.

Unit objectives

This course is designed to enable you to:

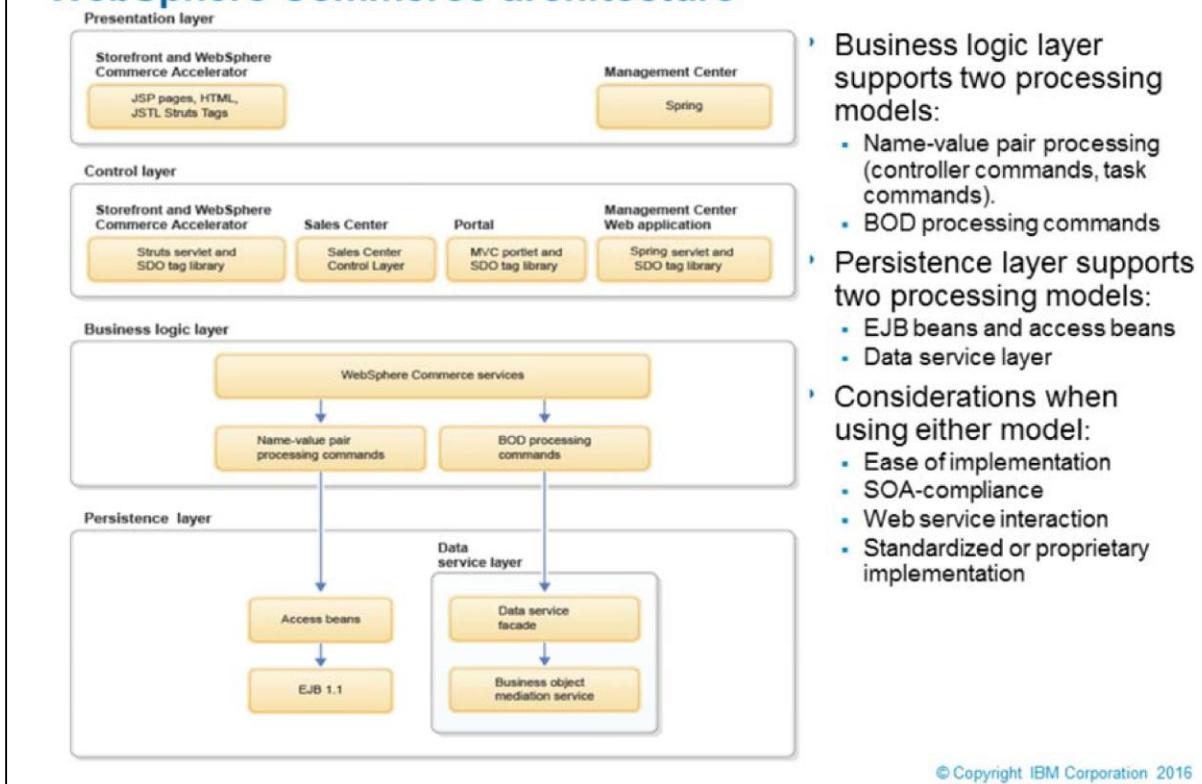
- Describe the purpose and function of BOD command processing in the business logic layer.
- Customize or extend BODs.
- Describe how service data objects (SDOs) are used within the BOD command processing framework.
- Leverage the Java Emitter Template (JET) to generate service modules and implementation code.
- Transform logical objects to physical objects by using business object mediators in the data service layer.
- Use a query template to translate XPath into SQL statements.

© Copyright IBM Corporation 2016

This course is designed to enable you to understand the purpose and function of BOD command processing in the business logic layer in WebSphere Commerce version 8, customize or extend BODs, and describe how service data objects (SDOs) are used within the BOD command processing framework.

Also enables about leveraging the Java Emitter Template (JET) to generate service modules and implementation code and covers the basic concepts and functions of data service layer, business object mediators.

WebSphere Commerce architecture



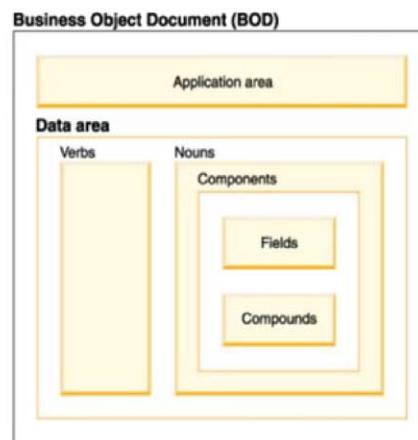
WebSphere Commerce supports two different models for processing business logic and persistence: name-value pair command processing, which uses controller commands and task commands, at the business logic layer interact with access beans and EJB beans. BOD processing commands at the business logic layer, which use BODs and their prerequisite nouns and verbs, interact with the persistence layer through the data service layer. The two models are not interchangeable within a single function: either you use name-value pairs with EJB beans, or you use BODs with DSL. The two models are interchangeable within a solution, however: both models might be used for different functions, but only one model per function.

There are several considerations to make when choosing one model over the other: BOD processing requires less Java development, is SOA-compliant, and provides easier integration with web services and external services. Name-value pair processing, however, although offering proprietary means of implementation, uses standard EJB technology, which is a widely accepted method for database access.

Business Object Document (BOD)

Business Object Documents (BODs) are OAGIS standard messages.

- Data area contains verbs and nouns:
 - Noun: a common business object.
 - Verb: an action on the Noun.
 - For example, a GetCustomer BOD contains a complex Customer business object (Noun) and a 'fetch' action (Verb).
- Application area contains information that the infrastructure can use to communicate the message.
 - Might include information such as creation date, sender identification, authentication data, a unique BOD ID.



© Copyright IBM Corporation 2016

WebSphere Commerce service interfaces are defined using the OAGIS (Open Applications Group Integration Specification) message structure.

The BOD is a common horizontal message architecture, which is used in order to achieve interoperability between disparate systems. The BOD messages can then be chained together to create different scenarios: for example, an ordering scenario might include messages for creating the order and processing the payment. BODs are the business messages that are exchanged between the presentation, business logic, and persistence layers; in other words, between components.

In order to achieve this interoperability, the message (a BOD) is constructed of a common business object (a Noun) and an action (a Verb). In the previous ordering scenario example, one of the BODs might be to Create (the action, a Verb) an Order (the business object, a Noun).

Advantages of BOD command processing

- Standardization
 - Uses standard OAGIS messages, processing standard BODs.
 - Commands are more generic.
- Abstraction
 - Well-defined separation between logical and physical data.
 - Decouples components for reuse in different environments.
- Simplification
 - Less Java development, more XML, and XPath development.
 - Lightweight runtime environment for request handling.
- Integration
 - BOD processing closely resembles web services, easier to integrate.
 - Extensive set of services for working with data.

© Copyright IBM Corporation 2016

Decoupling allows reuse in other environments besides the WebSphere Commerce application, such as WebSphere Portal applications, the IBM Sales Center, and any other custom applications.

WebSphere Commerce is moving toward an approach that focuses on multichannel support instead of an approach that is primarily web-based. Instead of the name-value pair URLs that represent a typical web request, business components declare structured objects that represent services. Components use OAGi XSDs to define services. This feature adds value by enabling developers to create business services that can be used regardless of protocol.

In the current WebSphere Commerce architecture, there are three different layers of the application: the presentation layer, the business logic layer, and the persistence layer. In WebSphere Commerce Version 6 Feature Pack 3.0.1, WebSphere Commerce introduced a new command framework for the business logic layer - the business object document (BOD) command framework. The concept of a BOD was used in previous versions of WebSphere Commerce, but recently, the entire command framework is tailored to also use BODs.

In prior versions of WebSphere Commerce, there are implementation dependencies between the presentation layer, business logic layer, and persistence layer. The new architecture uses well-defined interfaces to decouple the implementation of the presentation layer, business logic layer, and persistence layer. From the business logic layer perspective, OAGi messages are used as the interface for making requests to retrieve business data or invoke business logic. The BOD command framework provides the capability to process these BOD requests and responses.

The interaction between the business objects and persistence layer is isolated in the business object mediator. Business object document (BOD) commands interact with the business object mediator to handle the interaction with the logical objects and how they are persisted.

BOD processing commands overview

- BOD commands deal with Service Data Objects (SDO) instead of name-value pairs.
 - BODs and SDOs are widely accepted and integrate easier with other applications and software.
- BODs can represent a complex request that performs multiple actions instead of just one.
 - Task commands represent a single action. A controller command is a single action wrapper of several smaller tasks
- BOD commands deal with a persistence interface called the data service layer by using an object that is called the business object mediator, and are independent of the persistence technology.
 - A clearer separation between business logic and persistence logic.

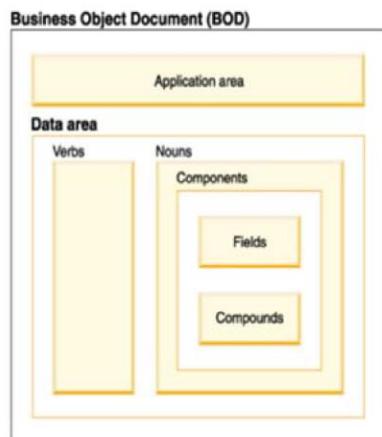
© Copyright IBM Corporation 2016

As part of the WebSphere Commerce move to an SOA architecture, three different layers of the application are identified. Consistent interfaces define the interactions between the layers. In previous versions of WebSphere Commerce, there are implementation dependencies between the presentation layer, business logic layer, and persistence layer. The new architecture in WebSphere Commerce version 6, Feature Pack 3.0.1 and later addresses this issue by using OAGi messages to interact with the business logic. The business logic command framework provides the capability of processing business object documents.

The interaction between the business objects and persistence layer is isolated in an object called the Business Object Mediator. Business object document (BOD) commands interact with the Business Object Mediator to handle the interaction with the logical objects and how they are persisted.

Services transform the OAGIS messages (BODs) to Java objects called service data objects (SDOs). The BOD commands use these objects as their interface to represent the BOD. The command then uses an object called the Business Object Mediator to accept service data objects and handle the mapping between these objects and how they are persisted.

BOD processing commands (1 of 2)



- OAGi Business Object Documents:
 - Noun represents business object
 - Verb represents action to perform
 - Implemented by Service Data Objects (SDO)
- Component facade exists to implement services:
 - Application area
 - Contains application context information (language, store, and so on)
 - Verb
 - Get or Show
 - Process or Acknowledge
 - Change or Respond
 - Sync or ConfirmBOD
 - Noun
 - Logical representation of the business object.

© Copyright IBM Corporation 2016

The Business Object Document (BOD) is an open standard for a common horizontal message architecture that the Open Applications Group (<http://www.openapplications.org>) developed. BODs are business messages that are exchanged between software applications or components.

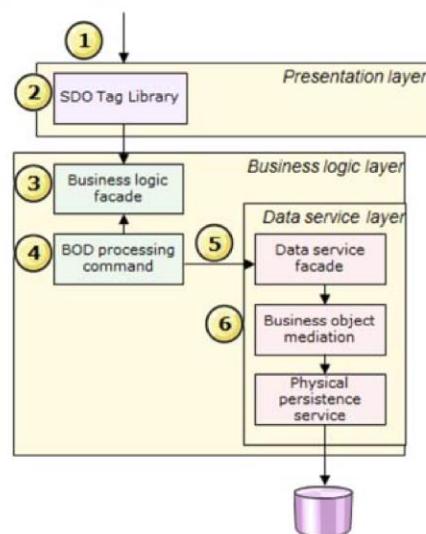
A BOD is a message structure that contains an application area and a data area to operate on. The application area of the BOD describes the application context to associate with the processing.

The application area is associated with specific application context information to control the processing and indicate application context when returning the result. In the context of a WebSphere Commerce service, the application area is where store, language, and other session type data is found. This information is common across all component facades and uses this information during processing.

The data area contains the service operation (verb) and the business object of the operation on (noun). The verb indicates what operation to perform or the response of the service operation. For requests, it contains details of the operation and for responses, it contains information that is related to the response. The noun represents the data in the business object that is related to the request or response.

BOD processing commands (2 of 2)

1. Inbound request that is received as a BOD from client channel.
2. Request data is interpreted into a SDO (Java interpretation of BOD).
3. Presentation layer calls business logic facade for execution.
4. BOD command is executed on logical object (Java-based SDO).
5. Command processing calls persistence through data service layer façade.
6. Business object mediation transforms logical object (Java) into physical object (database row).



© Copyright IBM Corporation 2016

Customization scenarios

- Three main customization scenarios to be aware of:
 - Classic WebSphere Commerce
 - Based on command framework that use NVPs as input.
 - SOI
 - Using OAGIS messages as input. Transforms input into NVPs and continue to use classic WebSphere Commerce commands.
 - BOD
 - BOD command framework that uses OAGIS messages as input. New commands still extend the WebSphere Commerce command framework. It uses the data service layer to map the BODs to the persistence layer.

© Copyright IBM Corporation 2016

NVP:

Services transform the OAGIS messages (BODs) to name-value pairs for processing by name-value pair commands. This makes for easy integration with existing WebSphere Commerce or customized commands. WebSphere Commerce commands are Java beans that contain the programming logic associated with handling a particular request. 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.

A controller command encapsulates the business logic for a business process. Individual units of work within the business process may be performed by task commands. As such, there are several ways in which a controller command can be customized, some of which involve customizing task commands.

There are two standard ways to modify existing WebSphere Commerce task commands. You can add new business logic to a task command, or replace the business logic of an existing task command

SOI:

WebSphere Commerce contains many controller commands that can be exposed as services. The Process and Change BOD command is just a command that converts the existing BOD request into a set of name-value pairs and delegates to an existing WebSphere Commerce controller command. However, the existing message mapping feature of WebSphere Commerce provides this feature and already addresses some of the additional customization aspects of extending the noun and adding more information. You need to customize only the controller command where the name-value pairs exist. You do not need to write new code to read the BOD request.

Struts config file:

```
<action parameter="order.addOrderItem" path="/AjaxOrderChangeServiceItemAdd"
type="com.ibm.commerce.struts.AjaxComponentServiceAction">
```

ChangeOrderSOIBODMapping.xml

```
<Command CommandName="com.ibm.commerce.orderitems.commands.OrderItem AddCmd"  
Condition='actionCode="Create" AND actionExpression="/Order/OrderItem">
```

Here OAGID message -> NVP = BOD:

The concept of a BOD has been used in prior versions of WebSphere Commerce, but in WebSphere Commerce Version 6 Feature Pack 3.0.1, WebSphere Commerce Version 7, and Websphere Commerce Version 8, the entire command framework is tailored to using BODs. The WebSphere Commerce BOD command framework architecture uses well defined interfaces to decouple the implementation of the presentation layer, business logic layer and persistence layer. From the business logic layer perspective, OAGIS messages are used as the interface for making requests to retrieve business data or invoke business logic. The BOD command framework provides the capability to process these BOD requests and responses.

BOD application area

- Commerce-specific session information (business context).
- Known context parameters:
 - storeId, langId, forUserId
- Common session information independent of the business component.
 - Input into the business context service.
 - Managed by specific context implementations.

© Copyright IBM Corporation 2016

The application area of the BOD describes the application context to associate with the processing. The application area will associate specific application context information to control the processing and indicate application context when returning the result. In the context of a WebSphere Commerce service, the application area is where store, language and other session type data will be found. This information is common across all component facades and will use this information during processing.

Nouns

- Logical representation of the business object:
 - Contains elements and attributes.
 - Does not contain service control parameters.
- Not reusing the OAGIS included nouns:
 - OAGIS nouns are rich.
 - Commerce functionality uses only 10% of the OAGIS-defined capacity

© Copyright IBM Corporation 2016

A logical model definition is required for services that are fronted by an XML schema. In WebSphere Commerce, the logical model definition is represented as a noun. WebSphere Commerce uses its own simplified nouns, defined types, and primitive XML schema types. WebSphere Commerce does not use nouns and base types provided by OAGIS.

Nouns define the name of each data element in the logical model, and assign that name to a data type. The data type can be a primitive XML schema type such as boolean, or a complex type. A complex type is a construct of data elements such as CurrencyType which contains price (represented by a double type) and currency (represented as a string). A noun can contain one or more noun parts.

Extending and customizing nouns

- **UserData element:**
 - A complex type that is intended as an extension point.
 - UserData contains simple name-value pairs; populate UserData with name-value pairs.
 - 1 - 1 relationship with the extended data.
 - **UserData is automatically passed to the service:**
 - The client facade (used to take data from the request and get it into the model), and a composer (a command used to get data from the model and add it to a response) handle population.
- For example, to extend member services, you would extend: MemberFacadeClient and ComposePersonFromDatabeanAllCmdImpl
- **Overlay:**
 - Extension of an existing complex type (similar to inheritance in Java).
 - Replaces the usage of the default complex type with the overlay.
 - 1 to many relationship to the extended data

© Copyright IBM Corporation 2016

There are extension points present in the existing services for adding more attributes.

The UserData element can be used as a data extension point to add new data without changing the logical model.

The overlay is an XML type that extends another XML type and contains more attributes and elements. This new type is then declared as a substitution of the type and can be used when creating a service request and returning a service response.

Schema customizations should be done by using one of following methods:

- 1) Using the UserData element (simplest) extension points that the component nouns provide.
- 2) Using schema overlays. Overlay extensions allow users to have their extensions appear within the OAGIS or WebSphere Commerce complex types. To add elements, a user must extend the OAGIS or WebSphere complex types within their own namespace.

For more details can be referenced in knowledge center :

https://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.component-services.doc/tasks/twvextendnoun.htm

There are tutorials in the information center that address these customizations.

Reference : https://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.tutorials.doc/tutorial/twvfounduserdata.htm

XML representation of an extended noun

```
<commerce-component name="ExtendedCatalog"
    company="SampleCompany"
    packagenameprefix="com.samplecompany.commerce"
    namespace="http://www.sampleco.com/xmlns/prod/commerce/9"
    nlsprefix="sampleco" type="BOD">
    <noun name="ExtendedCatalog" get="true"
        process="true" change="true" sync="false"/>
</commerce-component>
```

- XML Schema Definition (XSD) contains data type information that relates to tables and columns.

© Copyright IBM Corporation 2016

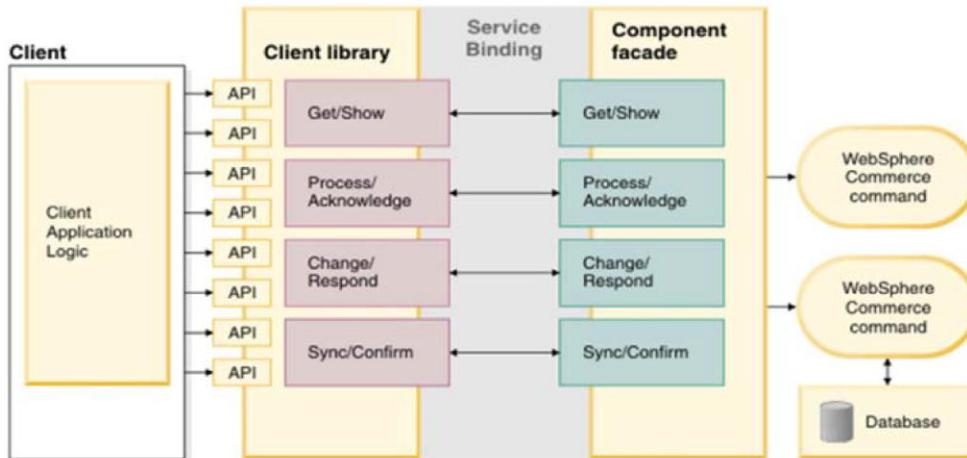
OAGi verbs

- Represent an action to be performed on a noun.
- Represent the request or response:
 - Request verbs contain expressions that represent the action to perform.
 - Response verbs indicate the result.
 - Response verbs communicate application errors.
- Verbs are high level and can support new actions.
- WebSphere Commerce uses only four request/response pairs of verbs:
 - Get and Show
 - Process and Acknowledge
 - Change and Respond
 - Sync and ConfirmBOD

© Copyright IBM Corporation 2016

A verb represents an operation that can be performed on a noun. For example, an "Order" noun has a specified operation that is performed in response to the verb "Get" (in this case, providing details to the client about the Order). Although OAGIS messaging defines a number of verbs; WebSphere Commerce uses a subset.

WebSphere Commerce Services functional architecture



© Copyright IBM Corporation 2016

The decoupling of the presentation layer from the business logic that started with version 6.0 of WebSphere Commerce includes adding a generic facade between the Struts framework and the command layer. This generic facade handles the name-value pair parameters for the WebSphere Commerce commands. In this release, instead of just having a single generic facade, WebSphere Commerce introduces well-defined facades for various business logic subsystems. These facades take structured objects (BODs) instead of name-value pair input and follow the OAGi processing model.

WebSphere Commerce supports only a subset of the OAGi verbs. This limitation is because the OAGi specification can get complex and not all of it was needed for Commerce.

The starting point of a WebSphere Commerce service is the definition of the high-level business object called the noun. Based on the noun, services are defined based on supported verbs that can act upon that noun. These combinations of verb and noun form the OAGi messages that represent a WebSphere Commerce service request and the corresponding response. The functional architecture is structured around the transmission of these OAGi messages from the client (for example, a portlet in a WebSphere Commerce Portal server) to the WebSphere Commerce Server, and back again.

The above diagram shows how a client such as a portlet can use WebSphere Commerce business logic:

- The client (for example, a portlet in a WebSphere Portal Server) uses the client library to create an OAGi message.
- The service binding routes the request.
- The service binding handles the serialization and deserialization of the OAGi message and sends the request to the component facade.
- The component facade calls the appropriate WebSphere Commerce commands.
- The component facade forms the response.
- The service binding routes the response back to the client library.

- The client library returns the response message to the client.

Example service: Member services

- Member services allow an external system to create, update, and search for members (organizations, users, and member groups) in WebSphere Commerce.
 - Two top-level nouns: Person and Organization.
 - Three verbs: Get, Process, and Change.
- The following table lists the operations that can be performed on a Person.

Service	Description
GetPerson	This service finds a person by unique ID, by distinguished name, or finds the person who makes the request.
ProcessPerson	This service registers the person with the store. Also used to create new persons.
ChangePerson	This service updates the person's information, and adds, updates, or deletes information for the person, such as addresses.

© Copyright IBM Corporation 2016

There are a number of component services available for use, focusing on common operations:

- Member
- Order
- Catalog

Get and Show verbs

- Get:
 - Request verb
 - Uses extended XPath notation to represent search clause
 - Similar to an SQL statement:
 - XPath expression represents the “where” clause
 - Access profile represents columns to select
 - Noun represents tables
 - Includes paging attributes
- Show:
 - Response verb
 - Includes the list of nouns that match the expression
 - Includes paging information

© Copyright IBM Corporation 2016

Get Request and the Show Response:

Get requests are requests for data, and use the Get verb, and return a Show response. The Get verb will indicate the search criteria used to retrieve the business objects along with paging options for paged requests. The Show response will include the business objects that match the search criteria.

An example of a Get request verb:

```
{_wcf.ap=IBM_Details}/Catalog[CatalogIdentifier[UniqueID=123]]
```

Compare this statement to an SQL statement.

```
select catalog_id, shortname, longname from catalog where catalog_id = 123
```

Control parameters can be added to the XPath expression if they are included in braces {}. Some such parameters might include the access profile, which provides directives on the amount of data to return, or the data language, which indicates the language of the data to return.

An example of a Show response verb:

```
<_cat>ShowCatalogEntry xmlns:Oagis9="http://www.openapplications.org/oagis/9"
xmlns:_cat="http://www.ibm.com/xmlns/prod/commerce/9/catalog"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Oagis9:ApplicationArea
        xsi:type="_wcf:ApplicationAreaType">
            <Oagis9:CreationDateTime>2007-01-
25T17:01:18.297Z</Oagis9:CreationDateTime>
                <Oagis9:BODID>954527f0-acbf- 11db-986f-
827045b7cf58</Oagis9:BODID>
                    </Oagis9:ApplicationArea>
                    <_cat:DataArea>
                        <Oagis9>Show/>
```

```
<_cat:CatalogEntry
catalogEntryTypeCode="ProductBean">

    <_cat:CatalogEntryIdentifier>
        <_wcf:UniqueID>51100000117</_wcf:UniqueID>
        <_wcf:ExternalIdentifier ownerID="700000000000000000000002">
            <_wcf:PartNumber>TAWI-03</_wcf:PartNumber>
        </_wcf:ExternalIdentifier>
    </_cat:CatalogEntryIdentifier>
    <_cat:Description
language="-1">
        <_cat:Name>"Corrolus" Wineglasses</_cat:Name>
        <_cat:Thumbnail>images/catalog/TAWI_03_sm.jpg
    </_cat:Thumbnail>
        <_cat:FullImage>images/catalog/TAWI_03.jpg</_cat:FullImage>
        <_cat:ShortDescription>"Corrolus"
wineglasses.</_cat:ShortDescription>
        <_cat:LongDescription>"Corrolus"
wineglasses.</_cat:LongDescription>
```

Process and Acknowledge verbs

- Process:
 - Used when submitting a new business object, canceling an existing business object, or invoking processing logic against an existing business object.
- Acknowledge:
 - Response includes shell of the processed business object.
 - Shell contains (at minimum) the unique identifier for the new or changed business object.
- Action code represents the operation of the process:
 - The business components define actions:
 - Process verb might contain Create, Delete, Register, or Submit actions.
 - For example, ProcessPerson supports a "Register" action, while ProcessOrder supports "Submit" or "Release" actions.

© Copyright IBM Corporation 2016

Process request and the acknowledge response:

Process is used when the client is submitting a new business object, canceling an existing business object, or invoking processing logic against an existing business object. The Acknowledge response will include the shell of the business object that was processed, containing at a minimum the unique identifier for the new or changed business object.

On Process requests, the action 'Create' is used for submitting a new business object such as create a new catalog entry. The action 'Delete' is used for cancelling an existing business object, like delete a catalog entry. Actions can also be user-defined. The action must indicate the business operation that needs to take place on the included Noun. There is no predefined list of actions that can be associated with the Process verb, because the action is very specific on the supported business process associated with the business object.

An example of a Process verb:

```
<_mbr:DataArea>

<oa:Process>
  <oa:ActionCriteria>
    <oa:ActionExpression actionCode="Register"
expressionLanguage="XPath"/>/Person[1]</oa:ActionExpression>
  </oa:ActionCriteria>
</oa:Process>
<_mbr:Person>
  <_mbr:PersonIdentifier/>
  <_mbr:ParentOrganizationIdentifier/>
  <_mbr:Credential>
    <_mbr:LogonID>test121622</_mbr:LogonID>
    <_mbr:Password>web1admin</_mbr:Password>
```

```
<_mbr:SecurityHint/>
</_mbr:Credential>
<_mbr>ContactInfo>
  <_wcf>ContactInfoIdentifier>
    <_wcf:ExternalIdentifier>
      <_wcf:PersonIdentifier/>
    </_wcf:ExternalIdentifier>
  </_wcf>ContactInfoIdentifier>
  <_wcf>ContactName/>
  <_wcf:Address>
    <_wcf:City>Toronto</_wcf:City>
  </_wcf:Address>
  <_wcf:Telephone1/>
  <_wcf:Telephone2/>

  <_wcf:BestCallingTime>Evening</_wcf:BestCallingTime>
  <_wcf:EmailAddress1>
    <_wcf:Value>abc@123.com</_wcf:Value>
  </_wcf:EmailAddress1>
  <_wcf:EmailAddress2>
    <_wcf:Value>abc@456.com</_wcf:Value>
  </_wcf:EmailAddress2>
  <_wcf:Fax1/>
  <_wcf:Fax2/>
</_mbr>ContactInfo>
</_mbr:Person>
</_mbr>DataArea>
```

Limited response information is needed for anAcknowledge verb.

Change and Respond verbs

- Change:
 - Used to notify the master repository that they should change a business object that they own.
 - Message contains verb and noun to change.
 - Noun contains the information to uniquely identify itself and the attributes to change.
- Respond:
 - Response includes shell of the business object that was processed.
 - Shell contains (at minimum) the unique ID for business object.
- ActionExpression indicates action to perform and an XPath pointer to the part of the noun to change.
 - XPath can use the syntax to include control parameters:
 - {_ord.param1='value'}/Person[1]/ContactInfo
- There is a finite set of recommended change actions:
 - Add, Change, or Delete

© Copyright IBM Corporation 2016

Change Request / Respond Response:

Change is used to notify the master repository that they should change a business object that they own. The Change message will contain the verb and the noun. The noun will contain the information to uniquely identify itself and the attributes to change. For example, when adding a contact to a customer's contact list (address book), the noun specified will contain the contact information to add along with the information to uniquely identify the Person business object.

An example of a Change request verb:

```
<_mbr:DataArea>
    <oa:Change>
        <oa:ActionCriteria>
            <oa:ActionExpression
                actionCode="Update" expressionLanguage="XPath"/>/Person[1]/ContactInfo</oa:
                ActionExpression>
        </oa:ActionCriteria>
    </oa:Change>
    <_mbr:Person>
        <_mbr:PersonIdentifier />
        <_mbr:ContactInfo>
            <_wcf:Address>
                <_wcf:City>Toronto</_wcf:City>
                </_wcf:Address>
                <_wcf:Telephone1>555-555-
                    5555</_wcf:Telephone1>
```

ngTime>

```
<_wcf:BestCallingTime>Evening</_wcf:BestCalli  
          <_wcf:EmailAddress1>  
              <_wcf:Value>abc@123.com</_wcf:Value>  
                  </_wcf:EmailAddress1>  
          </_mbr>ContactInfo>  
      </_mbr:Person>  
</_mbr>DataArea>
```

Limited information is required in the Respond response verb.

Sync and ConfirmBOD verbs

- Sync:
 - Used to notify interested parties the current state of the business objects that the system manages.
 - Only the system that contains the master data record should be sending Sync requests.
 - Used to synchronize the business object or objects across systems.
 - Initiated by the system that holds the master data record for the business object.
- ConfirmBOD:
 - Generic result to indicate the result of processing the Sync request.

© Copyright IBM Corporation 2016

Sync request and the ConfirmBOD response:

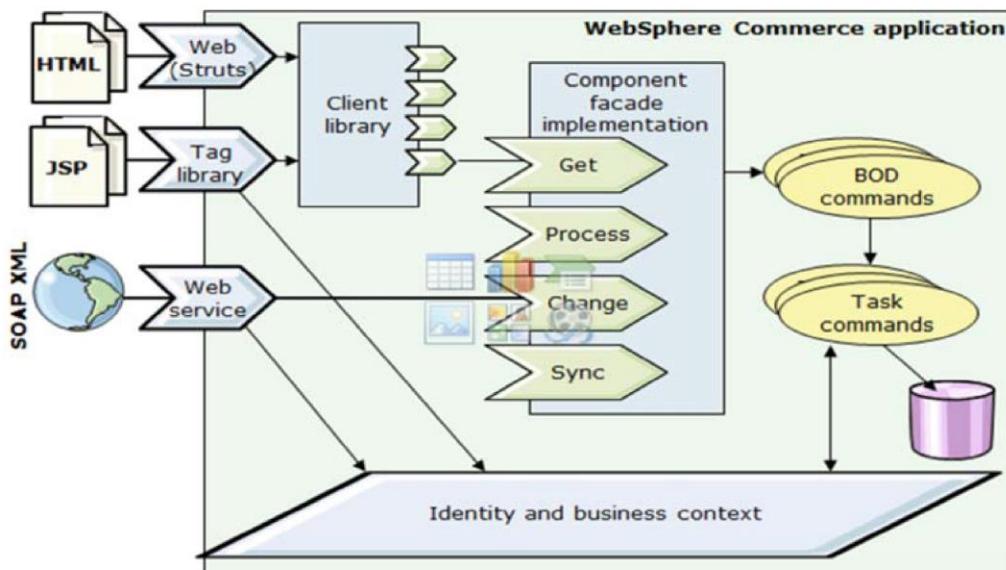
Sync is used to notify interested parties the current state of the business objects that the system manages. Only the system that contains the master data record should be sending Sync requests. Sync is used to synchronize the business object or objects across systems and is initiated by the system that holds the master data record for the business object.

A common example of when sync is used is asynchronous updates by JMS. Sync is not limited to this usage but typically when backend systems are pushing updates, these updates are sent asynchronously and reliably. For example, the master data record for member will use Sync BOD to broadcast updates. The Sync BOD request will be received from the master data record when the order status changes.

The following example shows a Sync message for the Person noun:

```
<oa:Sync>
<oa:ActionCriteria>
<oa:ActionExpression actionCode="Create" expressionLanguage="XPath">
/Person[1]
</oa:ActionExpression>
</oa:ActionCriteria>
</oa:Sync>
```

Transforming requests into BOD commands



© Copyright IBM Corporation 2016

The service binding resides between the client library and the services. It provides the transport mechanism to pass data by using Service Data Objects (SDOs), between the client and the service. This transport mechanism can be web services or local Java binding.

Service Data Objects

SDOs are an implementation of the value object pattern. SDOs are used to hold data, and have methods to get and set this data. Additionally, they are able to serialize and deserialize themselves based on the transport mechanism, and keep a history of the changes they underwent. In WebSphere Commerce, the benefit of using SDOs is that they start on the client side, in their Java form, serialize themselves to XML for transportation through web services, and then deserialize themselves back into a Java object on the WebSphere Commerce server. You do not need to transform input and output data for every stage of the processing lifecycle of a request because the SDOs can change forms based on where they are being used.

Transport types

There are two types of binding between the client library and the component facade implementation.

Local enterprise bean

The local enterprise bean connects the client to the component facade implementation in the local JVM. When the client and component facade implementation are deployed within the same application, the client should communicate with the component facade implementation through a local EJB call. The client uses the component facade's localenterprise bean to invoke the method that matches the intendedservice.

Web services

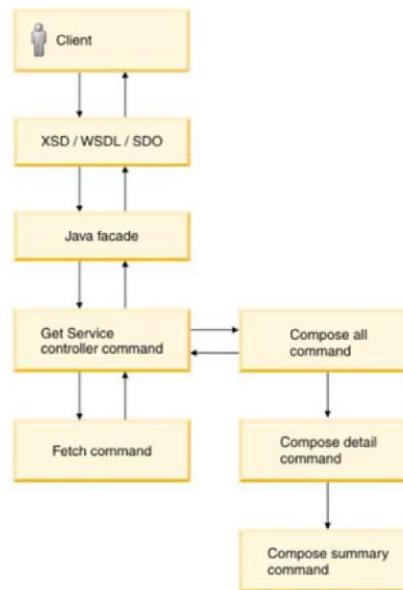
The Web service connects the client to the component facade implementation remotely. When the client and component facade implementation are deployed in separate applications, the client sends the service request by using web services as the transport layer.

Because the client and component facade implementation are distinct applications, the web service

security is used for authorization.

Get design pattern by using existing data beans

- Common design pattern for retrieving and displaying information from web services.
- Get service is divided into two tasks:
 - Fetch the data.
 - Compose the data into the logical model.
- One fetch interface, multiple implementations:
 - Fetch implementation to callout to return Data beans that are based on an expression.
- Compose commands to populate the logical model with information based on access profile.
 - Compose implementation per supported access profile.
- Programming pattern to reduce cost of customization to add a search expression.



© Copyright IBM Corporation 2016

The design pattern for Get services is the basic design pattern to be used for retrieving and displaying information from Web services.

Fetch the data:

The Fetch command returns a list of data beans that matches the expression. Extensions of this Fetch command are associated with a particular XPath expression. They must implement the search expression only to return the appropriate list of data beans that matches the expression.

Compose the data:

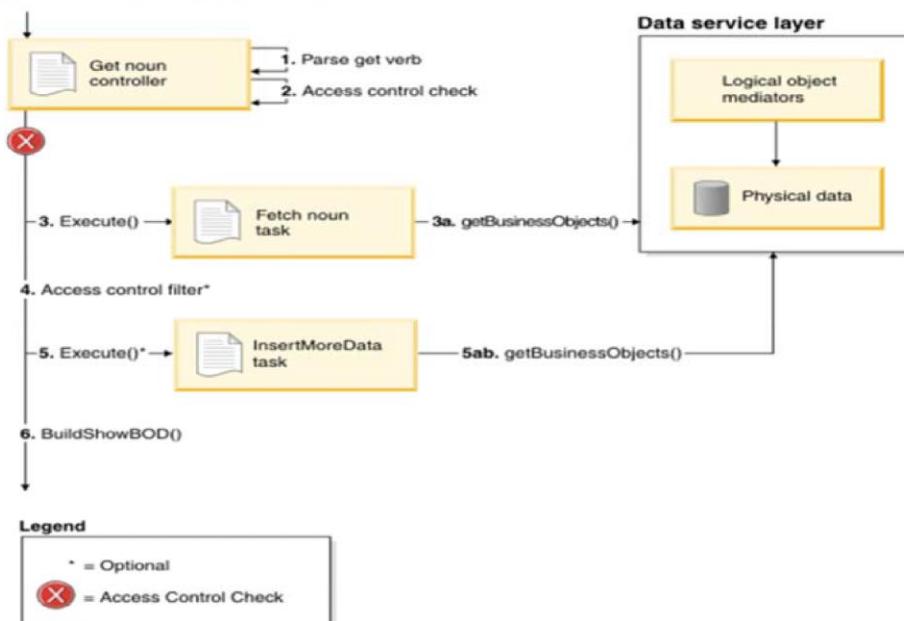
The command framework can use the XPath expression and Fetch task command to resolve the Get request to a particular implementation by using the existing WebSphere Commerce command registry (CMDREG) data. Instead of having one implementation for a Fetch business task, the command framework uses the XPath as the selector to resolve the implementation. If a specific implementation is not defined for the given XPath, then a default Fetch is used.

Compose:

The Get command takes the list for each data bean, it calls a Compose task to transform the data bean into the appropriate logical model(noun).

It is recommended that an access profile be used to scope the response data. Using an access profile makes it easier to extend the service at a later time, using different profiles to allow different access or data being returned. For example, to return a specific view of the data (such as a search view of a catalog entry), or to return the data in all languages for authoring purposes.

Get processing pattern



© Copyright IBM Corporation 2016

The business object document get processing pattern describes the design pattern used to search and retrieve data.

Processing the Get request involves a Get controller, which delegates to task commands:

- A fetch command that pulls the data for the requested business objects from the persistence layer and transforms it into the OAGi logical structure.
- An optional InsertMoreData task command that is used to augment the objects that the fetch retrieves, for example, to add computed values or aggregate with content from an alternate data source.
- The GetNoun command is a controller command that parses the expression from the get request, perform common validation such as access control, and then delegate to the fetch command to retrieve the list of nouns that match the expression. The fetch command returns the list of nouns that match the search expression. The GetNoun command uses the XPath selector approach to choose the appropriate fetch command implementation to retrieve the data.

Preceding diagram describes the detailed flow of the Get processing pattern :

1. The get noun controller command executes and parses the getverb into a search expression.
2. An access control check is performed to ensure the current user is allowed to use the access profile specified.
3. The fetch command runs the searchexpression.
 - a. The fetch command delegates to the business object mediator torun the search expression and returns a list of nouns matching the expression.
4. There is an optional step of access control filtering of the data. The list of nouns returned by the fetch command is filtered to ensure the current user only views those nouns that they are allowed to see. To override this, you can override the filterNouns() method with an empty implementation.

5. The InsertMoreData command runs to retrieve additional data (which could be from alternate data sources) pertaining to the noun.
 - a. The InsertMoreData command delegates to the business object mediator to execute the search expression and returns a list of nouns matching the _____ expression.
6. The nouns and the show verb is packaged in the ShowNoun response and returned.

For more details refer to knowledge center here:

https://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/csdsoaget.htm

Customization points for the Get pattern

- Scenario 1: New XPath expressions
 - New expression represents SQL statement.
 - Update DSL template to register XPath to SQL.
 - New expression where business logic computes a result.
 - Create a fetch command to handle data retrieval.
 - Update command configuration to associate XPath key with fetch command.
- Scenario 2: Manipulate access profile
 - Access profile returns more data from database.
 - Update DSL template to return more data for a particular access profile.
 - Update business object mediator to handle association of new data in logical model.
 - Access profile returns more data from an external system.
 - Create implementation to access external system, retrieve data, and add more data in logical model.
 - Update command configuration to associate access profile with new implementation.
 - Return a logical model that has a different amount of data populated.
 - Update access control to register who can execute profile.
 - Update DSL configuration to register data that represents new profile.
 - (optional) Create and register a new implementation

© Copyright IBM Corporation 2016

The implementation class to use when fetching data is **FetchNounCmdImpl**.

The implementation class to use when manipulating the access profile is **InsertMoreNounDataCmdImpl**.

Process or Change requests

- Generic message mapping command use the message mapping feature to reuse existing controller commands.
 - New message mapping configuration defined called "component-services".
- Message mapping command invokes controller commands and returns response maps.
- Acknowledge or Respond verbs create response messages.



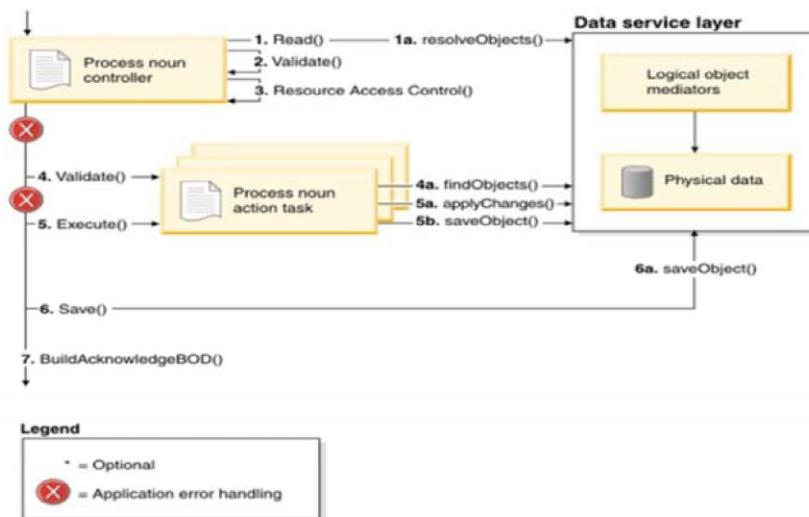
© Copyright IBM Corporation 2016

WebSphere Commerce contains many controller commands that can be exposed as services. The Process and Change BOD command is just a command that converts the existing BOD request into a set of name-value pairs and delegates to an existing WebSphere Commerce controller command. However, the existing message mapping feature of WebSphere Commerce provides this feature and already addresses some of the additional customization aspects of extending the noun and adding more information. You need to customize only the controller command where the name-value pairs exist. You do not need to write new code to read the BOD request.

For this purpose, the service request is mapped to a generic message-mapping service command called com.ibm.commerce.foundation.server.command.soi.MessageMappingCommandImpl. This generic command uses the message-mapping configuration that is defined under the name "component-services" to convert the SDO into a set of name-value pairs and a WebSphere Commerce command implementation. Then, it implements the command and responds to the request.

In the message mapping configuration, this generic command looks for two control attributes to determine the code to call to generate the response. The attributes are responseCommand and errorCommand. The generic command uses the specified implementation to generate the response to the request. This implementation is the code hook to generate the success or error response. These responsegenerators must follow the MessageMappingResponseCmd interface.

Process processing pattern



© Copyright IBM Corporation 2016

The following describes the detailed flow of the BOD Process processing pattern:

1. The Process noun controller command breaks down the BOD and calls the read() to resolve the root object of the nouns to change.
2. The validate() method is called to perform any common validation required.
3. An access control check is performed to ensure that the current user is permitted to change the specified noun.
4. The Process noun action task command or commands (there can be more than one) are instantiated, and for each command, the validate method is executed to report potential errors that might occur during processing.
 - a. The Process noun action command reads any information that is required and validate whether the input is valid for the operation.
5. The Process noun action task command or commands (there can be more than one) are executed to perform the change.
 - a. The Process noun action task command applies the changes.
 - b. The Process noun action task command saves any changes that are made to data objects retrieved in the current instance of the command.
6. The save is called on the object that is retrieved in step 1.
7. The response is created and returned.

Generating BOD processing packages

- WebSphere Commerce leverages Java Emitter Template (JET) processing.
- JET:
 - Eclipse-enabled template engine for generating applications.
 - Installed with Rational Application Developer.
- WebSphere Commerce uses JET plug-in to:
 - Create WebSphere Commerce service modules from XML files.
 - Generate BOD client and server implementation code.
 - Reduce the amount of setup and configuration to perform.
- WebSphere Commerce JET template:
 - com.ibm.commerce.toolkit.internal.pattern.componentprojects

© Copyright IBM Corporation 2016

The process of creating service modules for BODs and BOD processing can be a time-consuming endeavor. Besides the BOD itself, which is no doubt the easiest task, a developer needs to also build the client and server-side processing so that the new BOD can communicate with and between the client presentation layer and the server business logic and persistence layers.

Fortunately, WebSphere Commerce uses the JET transformation plug-in for creating these service modules from a simple XML file. By describing the service module in a specialized XML syntax, the service modules can be generated. This allows a developer to start directly with the service module implementation without having to spend hours with the setup and configuration of a service module.

The location of these XML files, that is known as application definitions, is changed slightly. In V6, the application description files were stored within a specific ComponentProjects project. In V7, your application definition files should be stored in the WebSphereCommerceServerExtensionsLogic project. Create a directory by name ServiceModuleDefinition to store the files.

BOD artifacts that the JET transformation generates

- *ServiceModule-Client*
 - Contains client-specific code that is used to build requests and handle responses.
- *ServiceModule-DataObjects*
 - Contains XSD that defines the noun and generated SDOs.
- *ServiceModule-Server*
 - Contains the implementation of the verbs.
- *ServiceModule-UnitTests*
 - Contains ready-built unit tests to verify the component.
- *ServiceModuleServicesHTTPInterface*
 - Enables the Server module for web services over HTTP.
- *ServiceModuleServicesJMSInterface*
 - Enables the Server module for web services over JMS.

© Copyright IBM Corporation 2016

Service data objects (SDOs)

- Service data objects provide a framework for data application development.
- SDO unifies representation of data from multiple sources by providing a single API for data access.
 - No need to know multiple technology-specific APIs.
- SDO framework is designed to support a disconnected programming model.
 - No need to be connected to a data source to manipulate data.
- SDO is integrated with XML.

© Copyright IBM Corporation 2016

It is not uncommon for an enterprise application developer to know several data access technologies, such as: JDBC, XML, JMS, Web services, and Enterprise Information Systems. Unfortunately, this requirement needs developers to become experts in many different data access technologies.

The goal of SDO is to provide a programming model that unifies data representation across heterogeneous data sources, and simplify application development for developers and tools providers.

SDO provides a common API that can be used regardless of the back-end data store that is being accessed. This common way of representing data also makes SDO an ideal choice for data abstraction in a service-oriented architecture.

In addition, support for some common programming patterns is built into the SDO architecture. SDO supports a disconnected programming model. Typically, a client might be disconnected from a particular data access service (DAS) while working with a set of business data.

However, when the client completes processing, and needs to apply changes to a back-end data store by way of a DAS, a change summary provides the appropriate level of data concurrency control. This change summary information is built into the SDO programming model.

Another important design point to note is that SDO integrates well with XML. As a result, SDO naturally fits in with distributed service-oriented applications.

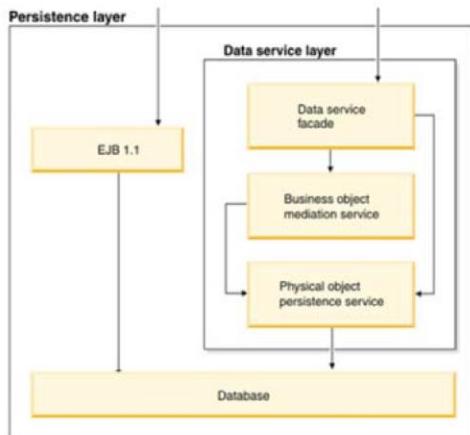
Finally, SDO is designed to support both dynamic and static data access APIs. The dynamic APIs are provided with the SDO object model and provide an interface that allows developers to access data even when the schema of the data is not known until run time. In contrast, the static data APIs are used when the data schema is known at development time, and the developer prefers to work with strongly typed data access APIs.

Overview of Data service layer

© Copyright IBM Corporation 2016

This section describes an overview of Data service layer.

Persistence Layer



- Database can be accessed in two ways:
 - **Directly** – through EJB beans and access beans
 - **Indirectly** – through the data service layer
- Advantages to both methods:
 - Directly
 - Uses a common, standard technology (EJB architecture)
 - Introduces access beans as simplified interaction with EJB beans
 - Indirectly
 - Greater flexibility SOA solution
 - Requires manipulation of XML files instead of more complex EJB beans and Java classes
 - Preferred method for Web 2.0 stores, web services

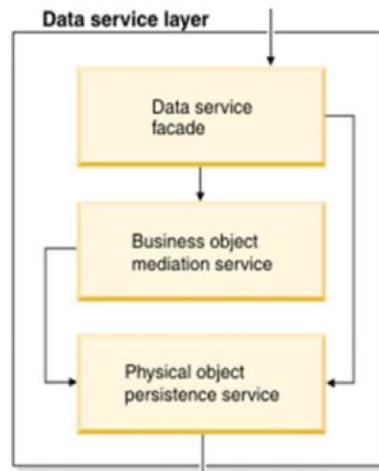
© Copyright IBM Corporation 2016

The interaction between the business objects and persistence layer is isolated in an object called the Business Object Mediator. Business object document (BOD) commands interact with the Business Object Mediator to handle the interaction with the logical objects and how they are persisted.

The persistence layer maps these structured objects to the persistence implementation to perform the data retrieval or updates. The persistence layer accepts structured data objects (SDOs) which are transformed (mediated) into objects called physical SDOs. Physical SDOs are stored in the data service layer. All persistence-specific assets, such as SQL queries, are isolated within the data service layer. To simplify maintenance, the query template file stores SQL.

Data Service layer (1 of 3)

- Abstraction for data access.
- Independent of the physical schema.
- Purpose:
 - Functions independent of the framework.
 - Transforms data that is retrieved from database into Java objects (implemented as SDO).
 - Offers bidirectional transformation between physical SDOs (schema) and logical SDO (classes.)
 - Allows user to perform CRUD operations on physical or logical SDOs.
- Services of the DSL:
 - **Data service facade:** Interface for accessing data.
 - **Business object mediation:** Initializes classes (mediators) that transform physical to logical data.
 - **Physical object persistence:** Allows mediators access to physical data, translating XPath to SQL.



© Copyright IBM Corporation 2016

The data service layer (DSL) provides an abstraction layer for data access that is independent of the physical schema.

The purpose of the data service layer is to provide a consistent interface (called the data service facade) for accessing data, independent of the object-relational mapping framework (such as EJB, DAS, or JPA). In its turn, the abstracted mapping framework is used to transform the data **that is** retrieved from the database into a collection of Java objects. These objects are implemented as physical service data objects (SDOs).

The data service layer performs bidirectional transformations between physical SDOs and logical SDOs. You can use the data service layer to perform Create, Retrieve, Update, and Delete (CRUD) operations on the logical SDOs. Alternately, you can also use it to do CRUD operations directly on the physical data, bypassing the logical schema altogether.

DSL consists of three pieces: the business object mediation service, the physical persistence service, and the data service facade.

The business object mediation service initializes mediators. These classes transform between the logical and physical representations of the domain model. This allows the business logic layer to deal only with the logical representation of the data.

The mediators access the physical data through the physical persistence service. This service translates XPath queries to SQL queries.

Data Service layer (2 of 3)

- Layer of abstraction for data access:
 - Abstract object-relational mapping frameworks (which transform database tables to Java).
 - Transform between physical SDOs and logical SDOs (OAGi nouns).
- Create, read, update, and delete (CRUD) operations on both physical and logical data.
- XPath:
 - Convert extended XPath query (logical schema) to SQL (physical schema).
 - Generate SQL statements from query template (business context aware).
 - Access profile.

© Copyright IBM Corporation 2016

The purpose of DSL, the data service layer, is to provide a neutral interface for accessing data, independent of the object-relational mapping framework (such as EJB, DAS, Hibernate, or JPA). In its turn, the abstracted mapping framework is used to transform the data that is retrieved from the database into a collection of Java objects. Internally, these objects are implemented as physical SDOs.

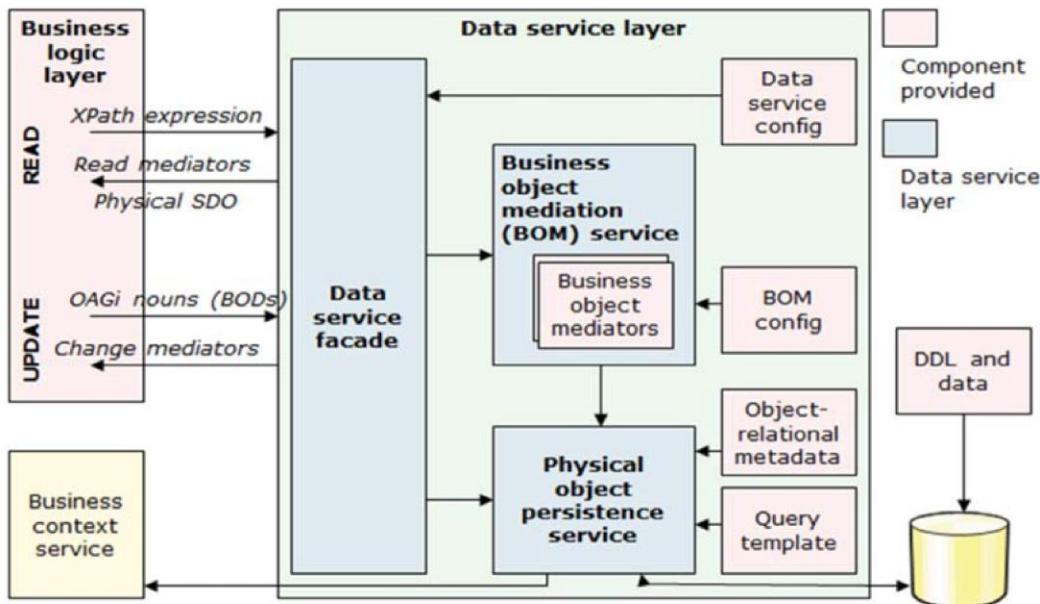
These objects are Java representations of the database schema (tables and columns). They are different from the logical SDOs that were discussed earlier. Logical SDOs represent the OAGi nouns, which are the external view of the business objects, as exposed by the components. DSL performs bidirectional transformations between physical SDOs and logical SDOs. This means you can use it to do create, retrieve, update, and delete operations on the logical SDOs. To be more accurate, the component-specific code, which you plug into the DSL does the job.

You can also use it to do create, retrieve, update, and delete operations directly on the physical data, bypassing the logical schema altogether.

XPath is used as a query language on the logical schema. DSL maps XPath queries to templates of SQL statements. These templates are used to generate actual SQL statements that access the database.

The data service layer also allows the templates to contain business context variables, which get substituted when the SQL statements are generated. Thus, the XPath queries sent in by the client can result in different actual SQL queries, depending on the property values of the business context. The statements are stored in a separate query template file, which isolates the runtime logic from the SQL query code.

Data Service layer (3 of 3)



© Copyright IBM Corporation 2016

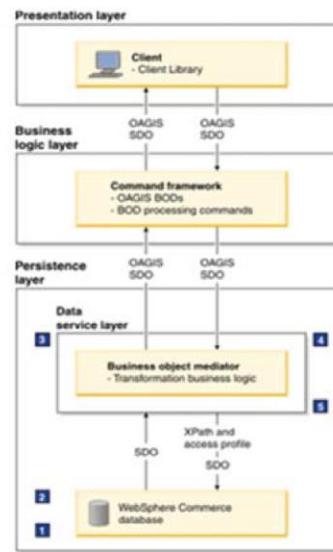
Consider a customization scenario in which you add new information to the logical model (such as warranty information for products). You want the new data to be stored in the database and to provide it to the client on demand. In this case, you can use your noun's `UserData` element to make this information available to the client. This does not require any changes to the logical model.

You store the new data in a new table that has a foreign key relationship to the table that represents your noun. For example, if you are adding warranty information for a product, you create a warranty table that references the `CATENTRY` table.

You can do this task without writing any Java code. You need to change only the configuration and add SQL statements that fetch the new information.

Customizing persistence through the Data service layer

1. Add new tables to database.
2. Generate custom metadata that describes new tables.
3. Generate static service data objects (SDO).
 - Java representation of new tables.
4. Configure logical data object to physical data object mapping.
5. Create a custom query template file.
The template file:
 1. Defines SQL template that fetches data from new tables.
 2. Associates new access profile to new SQL template statement.
 3. Associates XPath expression to new SQL template statement



© Copyright IBM Corporation 2016

You can extend the existing schema to support customization. These extensions include changing the physical database schema, generating the object-relational metadata and physical SDOs, and using query templates to retrieve the new information.

In order to customize persistence through the Data Service Layer:

1. Update the WebSphere Commerce database with new tables and relationships.
2. Generate custom object-relational metadata that describes the new tables and relationships.
3. Generate static service data objects (SDO) that provide a Java representation of the newly added tables. The term "static SDOs" and "physical data objects" are used interchangeably.
4. Configure the logical data object to physical data object mapping.
5. Create a custom query template file to map an XPath expression and access profile to SQL. This template file:
 - a. Defines a SQL template statement that fetches data from the new tables.
 - b. Associates a new access profile to the new SQL template statement.
 - c. Associates a XPath expression to the new SQL template statement.

The WebSphere Commerce data service layer wizard generates the physical SDO Java classes for your customizations, along with the required object-relational metadata and configuration for business object mediators. These steps are required if you add a table, and also for any new service modules you develop. XML assets that are generated are stored in the service module configuration extension directories and custom physical SDO Java classes are stored inside the WebSphereCommerceServerExtensionsLogic project.

Service modules, BODs, and the DSL

- Service modules contain the assets that are required to enable BOD processing in the data service layer.
- Each service module contains the following assets:
 - Static physical SDOs that provide a Java view of the schema.
 - Object-relational metadata that maps the physical SDO to the actual table.
 - Query templates that map XPath notation to SQL template queries.
 - Logical SDO to physical SDO mapping.
 - Business object mediators that build logical SDOs from physical SDOs:
 - Read mediators build logical SDOs from physical SDOs.
 - Change mediators build physical SDOs from logical SDOs.

© Copyright IBM Corporation 2016

Query templates

- Purpose:
 - Map XPath extended query to SQL statements
 - Isolate SQL statements from Java code
- Structure (five sections):
 - Column symbol definitions
 - XPath to SQL statements (defines name, template)
 - Association SQL statements (defines name, template)
 - Profile (references name, graph composer)
 - SQL statement (defines name, template)
- Development:
 - Design Pattern Toolkit or Java Emitter Template (JET)
 - Physical Data Object wizard

© Copyright IBM Corporation 2016

The query template file provides a mechanism to easily map from the query on your logical model, that is, your XPath query, to one or more SQL statements. These statements retrieve the physical data from the database. The file also isolates the SQL statements from the runtime Java code. This makes the code easier to maintain. It is also useful to database administrators when they want to locate and analyze queries. Changes to the SQL queries do not require Java recompilation.

Design Pattern Toolkit and the component projects design pattern:

The Design Pattern Toolkit (DPTK) is an Eclipse-enabled template engine for generating applications that are based on customizable, model-driven architecture transformations. WebSphere Commerce uses the DPTK plug-in for creating WebSphere Commerce service modules from a simple XML file. In WebSphere Commerce V7, particularly in RationalApplication Developer V7.5, the Java Emitter Template (JET) is the preferred method for generating templates. JET is a more standard, wider accepted generator than DPTK. By describing the service module in a specialized XML syntax, the service modules can be generated. So, you can start directly with the service module implementation without having to spend hours with the setup and configuration of a service module.

Data service layer physical data object wizard:

The WebSphere Commerce physical service data objects wizard generates the physical SDO Java classes for your customizations. For creating physical to logic business object mediator mapping, it is generated by extending the BOD by using UserData. These steps are required if the database schema is changed (new table, new relationships between existing tables, new columns, changed column types) and for any new service modules you develop. XMLassets that are generated are stored in the component configuration extension directories and custom physical SDOs are stored inside the WebSphereCommerceServerExtensionsLogic project.

Sample Query template

```
BEGIN_XPATH_TO_SQL_STATEMENT
name=/CatalogEntry[CatalogEntryIdentifier[ExternalIdentifier
  [(PartNumber=)]]]
base_table=CATENTRY
sql=
  SELECT CATENTRY.$COLS:CATENTRY_ID$3
  FROM CATENTRY
  JOIN STORECENT ON (CATENTRY.CATENTRY_ID =
    STORECENT.CATENTRY_ID AND STORECENT.STOREENT_ID =
    $CTX:STORE_ID$)4
  WHERE CATENTRY.PARTNUMBER IN (?PartNumber?) AND
    CATENTRY.MARKFORDELETE = 05
END_XPATH_TO_SQL_STATEMENT
```

© Copyright IBM Corporation 2016

1. The XPath statement. In this example, the XPath statement looks for a catalog entry based on a part number.
2. The base table upon which to search. The base table must be defined.
3. The matching SQL statement. The \$cols.catentry_id\$ symbol was defined earlier in this same query template file as the primary key of the CATENTRY table.
4. Another symbol, \$ctx:store_id\$, describes the store ID taken from the context.
5. The part number in the SQL statement is a match for the same from the XPath statement.

This query template examines the CATENTRY table based on a part number. It selects a valid product, based on its primary key, from the CATENTRY table for the current store (taken from the context).

Problem determination for BOD and DSL (1 of 2)

- DSL
 - The query name, SQL statement, and the template file where the query is defined is displayed in the trace.
 - SELECT queries:
com.ibm.commerce.foundation.server.services.dataaccess.*=all
 - UPDATE and INSERT: com.ibm.ws.sdo.mediator.jdbc*=all
- Component services
 - com.ibm.commerce.foundation.server.command.*=all
- BOD-related tracing (for example, transformations)
 - com.ibm.commerce.foundation.server.command.*=all
- DSL:
 - com.ibm.commerce.foundation.server.services.dataaccess.*=all
- Client library:
 - com.ibm.commerce.foundation.client.*=all

© Copyright IBM Corporation 2016

Problem determination for BOD and DSL (2 of 2)

- To trace query parameters:
 - Update wc-attribute-masking.xml and uncomment:
 - <wc:MaskedAttribute name="extendedDataValue" compareType="starts" />
 - Update wc-server.xml and remove:
 - <Parameter display="false" name="extendedDataValue"/>
 - Restart the server after the changes.

© Copyright IBM Corporation 2016

Error handling in the BOD command framework

- Service modules throw either applications errors or system errors.
- Application exceptions (which are usually first thrown):
 - Business logic problems such as input validation, and authorization issues.
 - The main BOD command throws the exception.
 - BOD processor performs the appropriate logic to account for the exception and call the instance of the command. Passing the exception. Response BOD represents the exception that occurred.
 - Any client validation that the XSD or simple rules on how to construct the request is server validation.
 - Server validation validates the BOD input and return a list of application errors in the ChangeStatus area of the response verb. (SOI can return only one application error.).
- System Application exceptions:
 - Unchecked exceptions.
 - Extend the abstract system exception class for common logging.
- handleException() method of your BOD handles compensation for rollback.

© Copyright IBM Corporation 2016

There are two types of exceptions that service modules can throw, application errors and system errors.

application errors:

This is a service module typed exception that represents a business logic problem, such as input validation, authorization issues, or other business

reasons. When an application exceptions is encountered, the main business object document command will throw this exception. When this exception

occurs, the Business Object Document processor will perform the appropriate runtime logic to account for the exception and then call the instance of the

command, passing the exception that occurred. The command will notified of the exception and should change the response Business Object Document

to represent the exception that occurred. This means the basic exception data must be populated but the command can add more information such as areas in the noun that caused problems. system errors:

The second type of exception is a system application error, which is an unchecked exception which does not need to be explicitly caught by clients. These

system exceptions indicate an internal problem which the framework would catch and handle and not the service module. If you want to throw system

exceptions, extend the abstract system exception class for common logging and other quality of service reasons.

In the SOI implementation, using name-value pair commands, only one action is supported.

The BOD command framework implementation support multiple actions, and so the framework supports returning multiple errors.

The handleException() method of your BOD command is called whenever an exception occurs which may require some compensation logic. The purpose of the compensation logic is to undo any operation as a result of the current operation being undone.

Best practices for BOD command framework (1 of 2)

- Configuration data for BOD service modules exists as either base or custom configuration.
- Custom configuration for new functionality or customize existing functions. Takes precedence over base configuration.
 - For example, complex data structures for NVP through a configuration file and configuration parser
- To modify and reload configuration of a BOD service module, without have to restart the server, use a .reloadconfig file.
- READ SQL queries under workspaces in DSL are likely bottleneck for performance.
- Use DB2 JCC type 4 connection type.
 - Although you can still use type 2, the syntax is converted internally to type 4 syntax and a type 4 connection is created. This conversion impacts conversion.

© Copyright IBM Corporation 2016

Best practices for BOD command framework (2 of 2)

- Business object thresholds
 - Applying limits on business operations reduces the risk of system attacks where unbound conditions might result in system failures.
 - Each service component can have one or more business thresholds that are defined in a component configuration file. Threshold checks by using the ConfigurationFactory API.
- Single step or two-step query?
 - Should use single-step query if possible.
 - Use two-step query if it cannot be done in a single query, you need to join many tables, and for pagination.
- DSL limitations
 - Should not be using the EJB and data service layer persistence models in the same transaction (mixed mode not supported).
 - Extension to BOD service modules must exclusively use DSL.
 - Complete list is found in information center.

© Copyright IBM Corporation 2016

You should use single-step queries if possible. However, in some cases it is not possible to fetch all the data in a single query or such a query needs to join many tables and might not perform. In this case, a two-step query should be used.

Another reason to use a two-step query is when the client requests paging. Paging on the result of a single step query is not possible if it returns multiple records for each base table record. A two-step query allows you to page on the result set returned by the first statement (the primary keys) rather than on the result set of the second statement.

Data service layer limitations

The following limitations apply to the data service layer: Multi-column primary keys are not supported for base tables.

The default graph composer is only able to merge the result sets of the association SQL statements if these result sets do not fetch identical records from tables other than the base table.

When using a two-step process to locate the objects by primary keys and then retrieve all the data given those primary keys, it is not always possible to sort the results of the final query. You cannot order the result of the XPath to SQL query and then propagate the ordering to the result of the associated SQL query. If ordering of the result is necessary, it is recommended to use a single-step query.

You should not be using the EJB and data service layer persistence models in the same transaction.

Extensions to SOI service modules (modules that use name-value pair commands) should not use the data service layer.

Extension to BOD service modules must exclusively use the data service layer.

Parametric search queries must use two-step queries (locate the objects by primary keys and then retrieve all the data given those primary keys).

You should never read or update the same data by using the JDBCQueryService and the PhysicalDataContainer within the same transaction. If you do, there is a chance that you read stale data or end up with corrupted data in the database.

Exercise 4: Extending an SOI service with UserData

© Copyright IBM Corporation 2016

Unit summary

This unit was designed to enable you to:

- Describe the purpose and function of BOD command processing in the business logic layer.
- Customize or extend BODs.
- Describe how service data objects (SDOs) are used within the BOD command processing framework.
- Leverage the Java Emitter Template (JET) to generate service modules and implementation code.
- Transform logical objects to physical objects by using business object mediators in the data service layer.
- Use a query template to translate XPath into SQL statements.

© Copyright IBM Corporation 2016

Additional training

For additional training needs, you have several options available for registering.

You can register	At
On site	This or any of our training facilities
Online	http://ibm.com/training/
Email	ibmswedu@us.ibm.com
By telephone	1-800-IBM Teach (1-800-426-8322)
By fax	208-692-6130

© Copyright IBM Corporation 2016

For additional training needs, you have several options available for registering like Onsite, Online , Email , by telephone or by Fax .

Unit 8

Web Services and Messaging

© Copyright IBM Corporation 2016

Unit objectives

This unit is designed to enable you able to:

- Explain the Messaging Services architecture and define predefined messages.
- List the features in WebSphere Commerce that support integration.
- Describe the administration and use of integration features.
- Describe the process of developing new behaviors to support integration.
- Describe WebSphere Commerce RESTful Services.
- Define a service-oriented architecture (SOA) and explain the role of web services in SOA and WebSphere Commerce.
- Develop a web service for use in WebSphere Commerce.
- Deploy a web service in WebSphere Commerce.

© Copyright IBM Corporation 2016

External system integration (Messaging)

After completing this topic, you should be able to:

- Outline the integration-related features that are present in WebSphere Commerce.
- Explain the administration and use of integration features.
- Explain the process of developing new behaviors to support integration.

© Copyright IBM Corporation 2016

Instructor notes:

Purpose — Initial topics start at a high level and then, drill-down to specific developer implementation of web services in WebSphere Commerce.

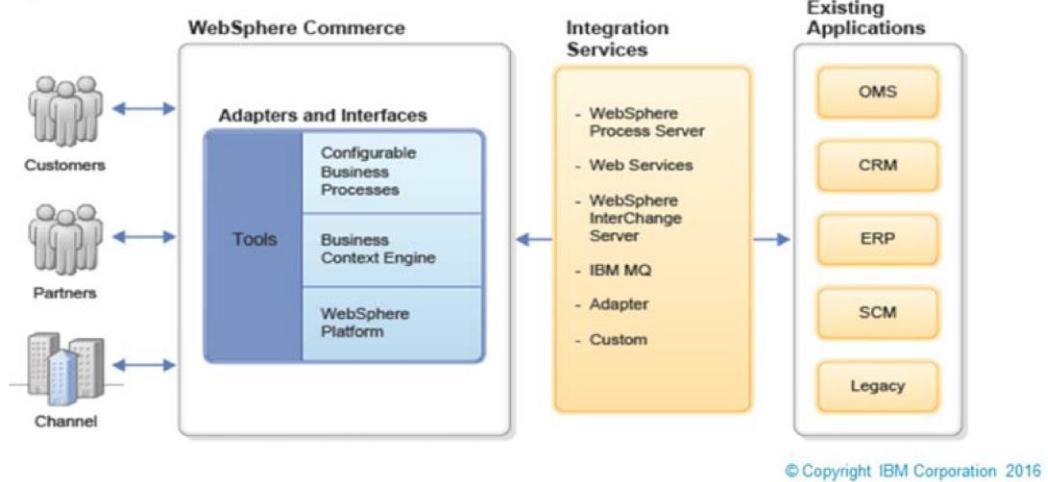
Details —

Additional information —

Transition statement — External system integration

External system integration

- Key feature of a WebSphere Commerce solution.
- Business logic is enabled for integration.
- Built-in adapters and interfaces are provided as common integration points.
- Uses various IBM connectivity solutions to leverage integration standards.
 - WebSphere Enterprise Service Bus
 - WebSphere Process Server
 - WebSphere MQ



© Copyright IBM Corporation 2016

WebSphere Commerce can integrate with:

- Customers by multiple channels
- Back-end systems within the enterprise
- Partners and suppliers

To help Integrators easily connect with external systems, WebSphere Commerce provides Reference applications. Reference applications contain documentation and sample code to integrate with a back-end system.

The WebSphere Commerce Messaging system gives WebSphere Commerce the ability to communicate with its external environment. This communication includes sending messages to and receiving messages from back-end systems or external systems, as well as sending notifications to customers and administrators that events occurred within WebSphere Commerce.

For example, you can set up the Messaging system to send email messages that notifies your customers that their orders are shipped. You can configure WebSphere Commerce to send a message to a back-end system whenever an order is created at your store. The back-end system uses the order information to do necessary Order Fulfillment processing. The back-end system can later send order status messages back to WebSphere Commerce indicating that order delivery is done, or an order invoice is issued. An email can also be sent to update the customer.

Inbound messages are used to run commands in WebSphere Commerce based on messages that are coming from back-end systems. Outbound Messaging system can generate outbound messages in order to update back-end systems with events that took place, such as a new customer order.

Instructor notes:

Purpose — WebSphere Commerce is integrated with a wide range of other software, including packages such as MAS 90, QuickBooks, One World, PeopleSoft, Oracle Financials, CommercialWare, and OrderMotion.

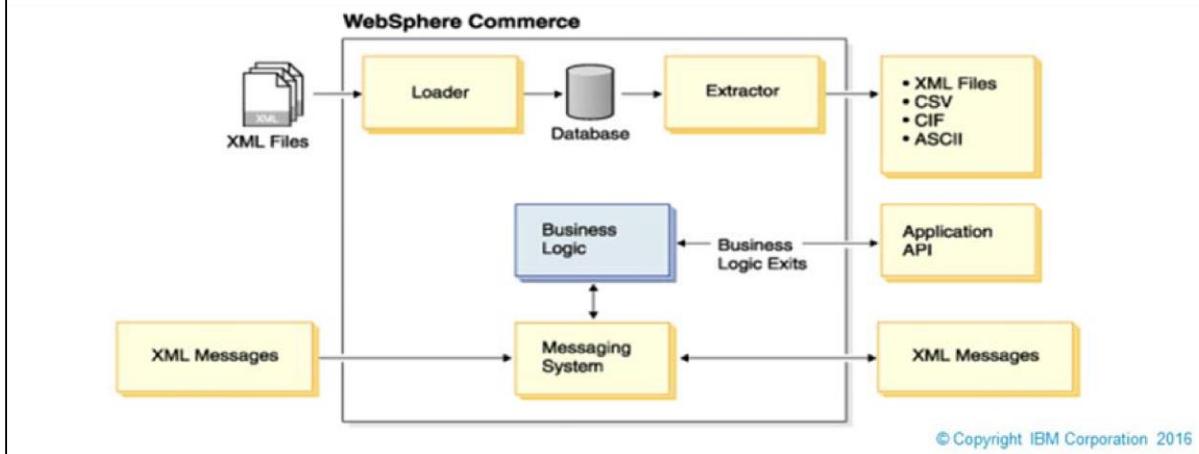
Details — For example, you can set up the messaging system to send email messages that notifies your customers that their orders are shipped. You can configure WebSphere Commerce to send messages to a back-end system whenever an order is created at your store. The back-end system uses the order information to do necessary order fulfillment processing. The back-end system can later send order status messages back to WebSphere Commerce indicating that order delivery is done, or an order invoice is issued. An email can also be sent to update the customer.

Additional information —

Transition statement — Internal System Integration

Internal integration facilities

- Data load utility
 - WebSphere Commerce command-line utility to populate database.
 - Use to load various files, including XML.
- Data extract utility
 - WebSphere Commerce command-line utility to extract data from tables in various formats.
- XML messages can also provide input to the messaging system.
 - Based on content, messages can signal execution of business logic.



The preceding diagram shows WebSphere Commerce components and utilities that are used to communicate with external systems. For example, XML files can be used as input by the data load utility that uses the files to populate the WebSphere Commerce database. The data extract utility can be used to extract data from the WebSphere Commerce database into various file formats. XML messages are also used as input to the messaging system, which is based on the content of the XML messages the Messaging system can perform business logic, or send messages to an external system.

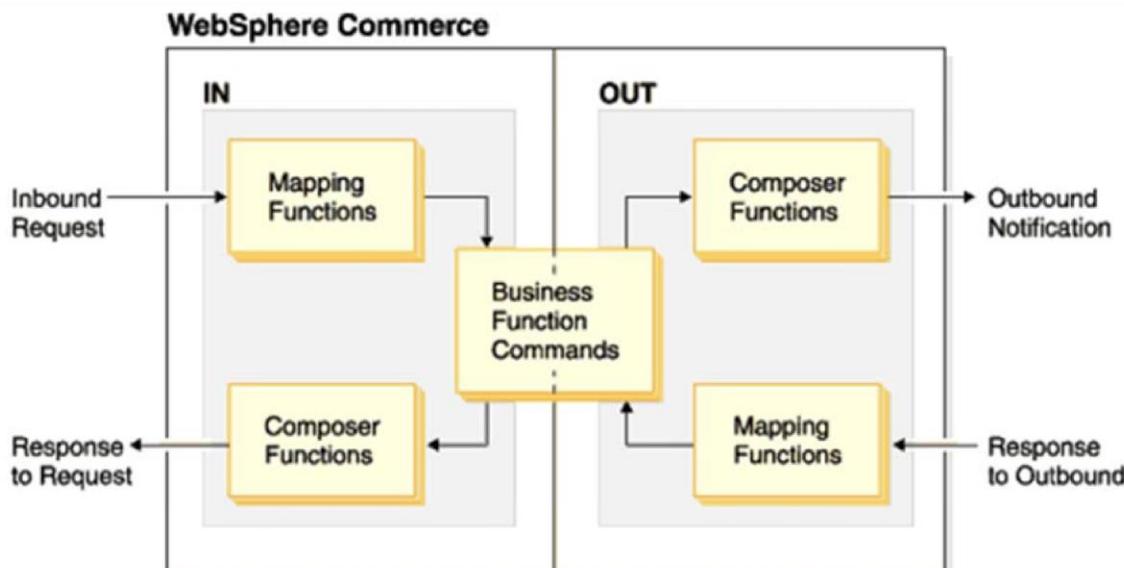
Instructor notes:

Purpose — Details —

Additional information —

Transition statement — Next, Messaging services architecture

Messaging Services Architecture



© Copyright IBM Corporation 2016

The Inbound Messaging system can receive XML-based messages via the HTTP protocol and WebSphere MQ. The Outbound Messaging system provides a means to send email notifications to customers and administrators, send XML-based messages by using the Listener for WebSphere MQ and WebSphere InterChange Server transports, and to write messages to a file.

Instructor notes:

Purpose — The messaging services that are provided allow for simple and complex integrations to file message-oriented middleware, XML, SOAP, and custom integrations.

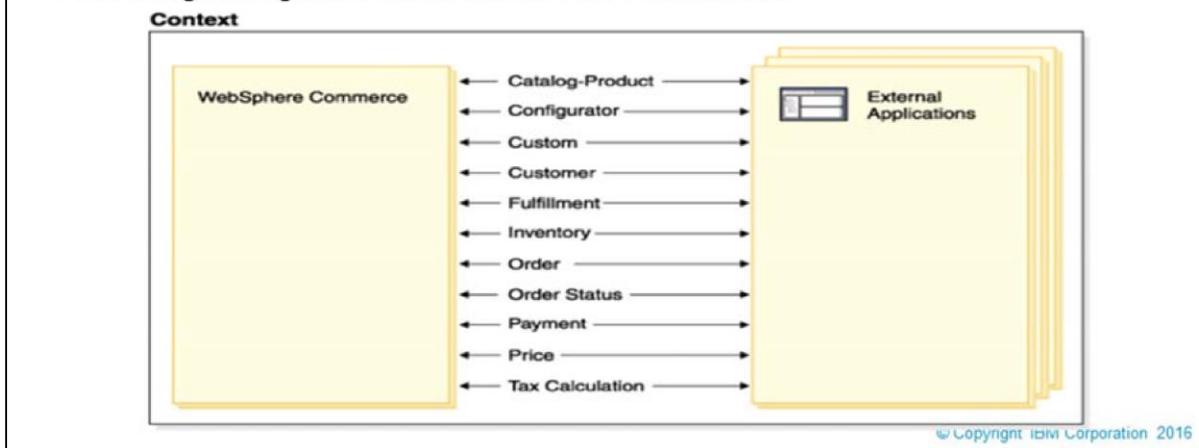
Details —

Additional information —

Transition statement — Next, you examine predefinedmessages.

Predefined messages

- WebSphere Commerce provides predefined inbound and outbound messages.
- Messages provide common messaging functionality.
- Architecture provides flexibility for adding new messages or transports.
- Architecture that is designed to send and receive predefined messages in XML.
- Messages might be associated with commands.



To speed up development, WebSphere Commerce provides many predefined inbound and outbound messages. The predefined messages provide common messaging functionality for fulfillment and back-end messaging needs. In addition, WebSphere Commerce provides architecture for adding new messages and transports.

The messaging system is prepared to send and receive a number of predefined messages in XML format. This format offers a high degree of readability, making the messages easy to modify and maintain. You can also use the legacy message format. However, the XML message format is recommended. You can also add new messages. For new inbound messages, you can associate them with either existing WebSphere Commerce commands, or commands that you created.

Instructor notes:

Purpose — With the volume of predefined messages available, the choice many times focuses on customizing existing messages rather than starting from scratch to develop a message set or messaging application. One of the mistakes that many developers make in the implementation of WebSphere Commerce is to try to complicate rather than extend the significant infrastructure that is available.

Details —

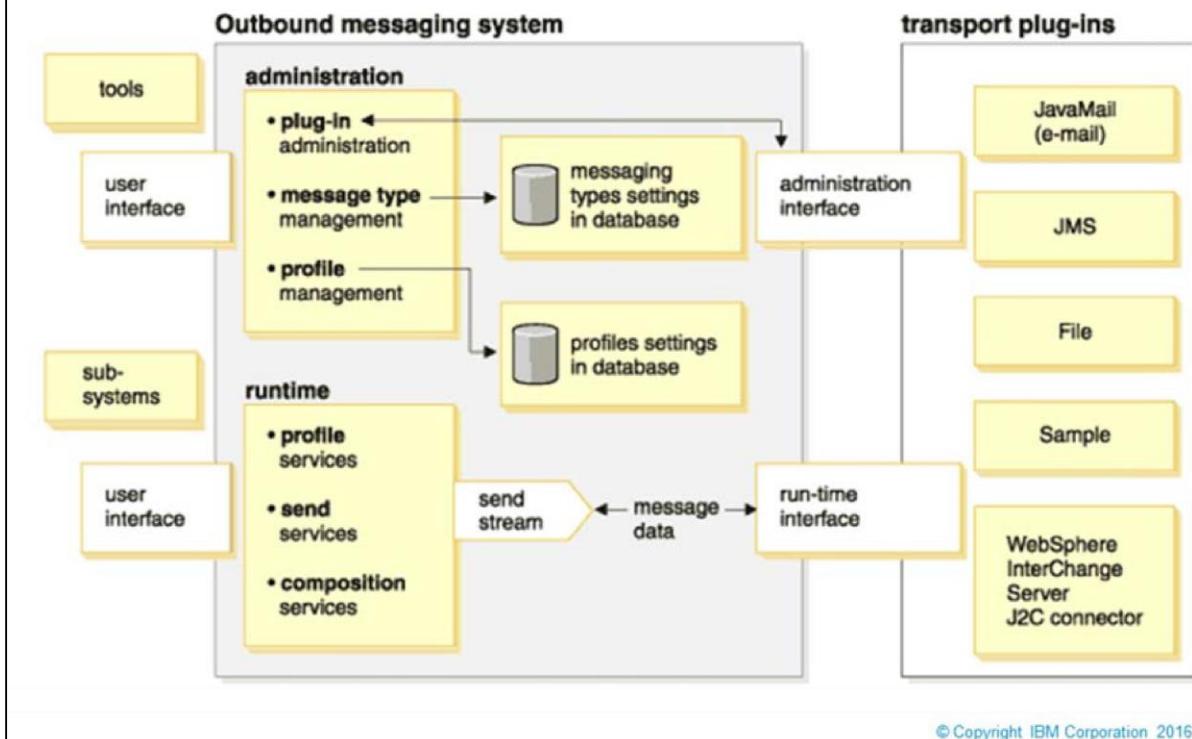
Additional information —

Transition statement — Next, you examine the outbound messaging service.

Outbound Messaging Services

© Copyright IBM Corporation 2016

Outbound messaging services



© Copyright IBM Corporation 2016

The WebSphere Commerce messaging system allows you to manage all aspects of defining and sending messages that are generated within WebSphere Commerce. It allows you to control the manner in which administrators, customers, back-end, and fulfillment center systems are notified of various events, such as customer orders or system errors.

To configure the outbound messaging system, use the administration console. The messaging system can send messages by using transports such as email by using SMTP and UTF-8 encoding. For email, the supported outbound protocol is SMTP. The message encoding depends on the preferred language of the user, and the store or site default language. Optionally, you can configure the messaging system to send messages to a back-end or fulfillment center system by using WebSphere MQ.

The outbound messaging system provides a highly customizable messaging environment. These features include:

- Composition services:** Some of the WebSphere Commerce messages use the message composition services. By using JSP files, the composition services generate a message before it is sent through the transport. If the composition service is used for a message, it runs a JSP page and passes the information such as an order number or a store number. When the template is executed, the JSP page might retrieve any additional information necessary for the message from the database by using Data beans. The output is generated and the formatted message is sent through the transport. You can modify the message templates just as you would any other JSP page. Some of the features of the generated message that you might want to modify includes:

- The layout of the message.
- The information about your store, the order, or the customer that is retrieved from the database and displayed on the page.
- The text of the messages to the customer such as "Thank you for ordering with us."
- The format of the output generated. For example, you might want to send a message in HTML format, plain-text format, or XML format.

Multiple message transmissions support

- Multiple notification messages over the same transport Support for three processing types includes:

- sendTransacted: Use for messages that should be sent upon successful completion of the current transaction.
- sendImmediate: Use for messages that should be sent when the event takes place in WebSphere Commerce. The message is sent whether the transaction commits or not.
- sendReceiveImmediate: (request/reply) Use for messages that require a response message from the back-end system.

Instructor notes:

Purpose — Describe the outbound messaging services available.

Note: Ensure transport attributes, for example email addresses, and file locations are valid. The messaging system does not validate attributes; incorrect attributes result in failure to send the message.

Details —

Additional information — sendTransacted uses the MSGSTORE table to hold the message until a job kicked off by the WebSphere Commerce scheduler delivers the message. This allows for the message transmission to be tried again if it fails one or more times. The number of failures defaults to three times, at which point the message is made inactive. "Dead letter" messages can be reviewed in the administration console tools of WebSphere Commerce.

Transition statement — Next, you examine the sending services.

Sending services

- Sending messages
 - public void sendImmediate()
 - public void sendReceiveImmediate()
 - public void sendTransacted()
- Getting the reply
 - public byte[] getResult() (tied to *sendReceiveImmediate*)
- Setting properties of the message
 - public void setMsgTypeName(String)
 - public void setPriority(Integer)
 - public void setStoreID(Integer)

© Copyright IBM Corporation 2016

The following methods are provided with the outbound messaging system sending services:

public void sendReceiveImmediate()

This method is used to perform a request/reply send. This type of send is used with the WebSphere MQ-JMS transport for back-end integration messages. The content of the reply is stored internally and can be accessed via the *getReply()* method.

Hint: To perform a send-receive by using the WebSphere MQ-JMS transport, you must ensure that you set the mode attribute appropriately, by using either the administration console or the *setConfigData()* method in the configurable message data services.

public void sendTransacted()

This method stores the message in the MSGSTORE database table. At a predetermined time, the WebSphere Commerce scheduler invokes a job that sends all messages that are stored in batch mode. Using this method ensures that a send occurs only after the caller commits or terminates successfully. This method should be used if blocking a call by using the *sendImmediate()* method cannot be tolerated.

public void sendImmediate()

This method is used to send a message a single time. Unlike the *sendTransacted*, the message is not stored and in the event of failure it is lost.

public byte[] getResult()

This method is used to retrieve the result of the *sendReceiveImmediate()* method. To obtain the result, it should be called after the *performExecute()* method, which executes the **SendMsgCmd** command. If the *SendReceiveImmediate* sending service is used, the Calling command can optionally get the result (that is, the Reply message) back. If the response is an XML message, users can process the reply message this way:

```
byte[] result =  
sendMsgCmd.getResult(); String  
xmlResponse = new  
String(result);  
CommandProperty cmd =  
MessageMapperGroup.getObjectForMessage(xmlResponse, "WCS.INTEGRATION");  
where WCS.INTEGRATION is the name of the message mapper. To do the preceding  
step requires defining the message content in the user_template.xml file and the  
corresponding DTD, just like adding a new Inbound XML message to the messaging  
system. The command name can be a dummy command name.
```

```
if (cmd != null) { TypedProperty tp = cmd.getRequestProperties();
    where tp contains all the mapped parameters.
public void setMsgTypeName(String msgType)
    This method is required. It is used to set the message type for the current message, by using the
    message type name.
public void setStoreID(Integer storeId)
    This method is required. It is used to retrieve message profile information for the store. To
    retrieve site-level information, the site-level store ID can be used. The messaging system
    attempts to retrieve a profile that is based on the store entered. If none exists, it attempts to
    retrieve a profile that is based on the default site identifier.
public void setPriority(Integer priority)
    This method provides optional initialization information. The priority integer specifies limits the
    profiles retrieved. Only those profiles whose priority range includes this integer is retrieved for
    the current message.
public void setConfigData(String key, String value)
public void setNLConfigData(String key, String language, String value)
```

Instructor

notes:

Purpose —

Details —

Additional information —

Transition statement — Next, creating outbound messages.

Creating an outbound message

1. Create a controller command.
 - Build the content of the message and send by using send services.
2. Add a message type to message type table.
3. Add a Struts entry.
 - Should include full path of JSP and device format (if using Mobile Commerce)
4. Assign the new message to a transport profile.
 - Profile includes transport (such as email) and device format.
5. Create a message object by using SendMsgCmd task.
 - Set the message type and store ID by using send services.
6. Invoke SendMsgCmd.execute().

© Copyright IBM Corporation 2016

1. Create a controller command or customize an existing task command to send a message with a certain message ID or name. A command needs to build the content of the new outbound message and send the message to the back-end system by using the send services of the outbound messaging system.
2. Add a row into MSGTYPES table assign a msgtype_id with a new return view, which is based on the command. Use a message type ID number above 1000. Use your company name as a prefix to the name of the view and message type name.
3. Create a row with the name in Struts and assign the devicefmt_id, and JSP page (with the full path).
4. Create a profile in the PROFILE table by using the administration console, and assign the message type to the correct transport (for example, email) and device format (for example, standard device format). A valid device format ID, as specified in the Struts configuration file, must be specified for each transport to be used. This device format ID (for example, DEVICEFMT_ID) must match with the one added into Struts configuration files in order to be able to pick the right JSP template file for composition. Most predefined message types (regardless for which transport they were designed) are designed to use the standard device format. The picking choice is strictly dependent on how the entries for the Struts configuration file are created. The default values of device format ID might change in future releases.
5. Create a messaging system object by using the SendMsgCmd Task command. Use the setMsgType(String) and setStoreId(Integer) described in the previous slide.
6. Invoke the SendMsgCmd.execute() method.

Instructor notes:

Purpose — Details —

Additional information —

Transition statement — Next, Inbound messaging services

Inbound Messaging Services

© Copyright IBM Corporation 2016

Inbound messaging services

- WebSphere MQ (JMS)
 - Bindings or client mode
 - MQ Listener must be enabled
 - Communication either serial or parallel
- Program adapter
 - Receives inbound XML over HTTP
- Message mappers
 - Converts inbound XML to CommandProperty object.
 - All WebSphere Commerce components might use the CommandProperty object.
 - Might also be used with commands.

© Copyright IBM Corporation 2016

Inbound messages over WebSphere MQ

The WebSphere MQ adapter allows you to integrate back-end and external systems with WebSphere Commerce by using WebSphere MQ. The WebSphere MQ adapter allows WebSphere Commerce to receive messages from back-end systems and external systems. The messages that are sent can be either XML messages or Legacy messages.

You can set up WebSphere MQ by using MQ Java in one of two modes:

Bindings mode

WebSphere Commerce is installed on the same machine as the WebSphere MQ server, and it connects to the WebSphere MQ server through WebSphere MQ Java by using the Java Native Interface (JNI). Since communication is through direct JNI calls to the queue manager API, rather than through a network, Bindings mode provides better performance than Client mode that is done by using network connections.

Client mode

WebSphere Commerce is installed on one machine and the WebSphere MQ server is installed on a back-end system.

To verify WebSphere MQ connections, queues, and channels, run test programs to put messages into and get messages from WebSphere MQ queues. For details, refer to your WebSphere MQ documentation.

Listener for WebSphere MQ

The Listener for WebSphere MQ is a component of WebSphere Commerce that enables integration with back-end systems by processing Inbound messages via MQ.

The Listener for WebSphere MQ has a set of predefined messages that help integrate WebSphere Commerce business processing with back-end or external system processing. Each incoming message activates processes within WebSphere Commerce to update database tables or perform other operations. In addition to the existing predefined messages, the Listener supports message extensions and new messages. The Embedded Messaging Server is only available in the WebSphere Commerce Developer Edition.

The specific documentation on enabling the Listener for WebSphere MQ is available at the following link:
https://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.

Parallel versus serial message processing in the Listener for WebSphere The Listener for WebSphere MQ can process Inbound messages in two ways: **Serial processing**

Each message is put in a line or queue, and handled one after another. In this method, each message must wait until processing of the previous message is complete.

Parallel processing

A number of messages can be processed at the same time. Instead of having each message wait for the previous one to be completed, many of them can be run simultaneously.

Although Parallel processing generally results in faster throughput, it is not suitable for all types of requests. The serial nature of the transactions must be preserved in some situations. For example, if a new customer registers at your store, then makes a correction to their address information, and then makes a Purchase Order, you would want the order of these transactions to be preserved when processing them. You could not perform the address modification or the Purchase Order unless the account is already created. Likewise, you would not want to fulfill a Purchase Order without having the correct shipping information.

Although it is generally preferable to use Parallel processing where possible, you have to decide where it is appropriate to use this method on your data.

Program adapter

The Program adapter allows external systems to communicate with WebSphere Commerce by passing XML requests over the HTTP protocol. The Program adapter provides external systems such as Procurement systems with a common way to communicate with WebSphere Commerce through HTTP, allowing WebSphere Commerce to act as a supplier to these systems, for Buyer-Supplier transactions.

The following steps illustrate the overall flow of an XML over HTTP request:

1. An external system sends an XML message to WebSphere Commerce over HTTP via a POST request. For example, `http://<host_name>/webapp/wcs/stores/servlet/`.
2. The request is mapped to the Program adapter.
3. The Program adapter passes the XML request to the appropriate Message Mapper.
4. The Message Mapper converts the XML request into a CommandProperty object and passes it back to the Program adapter.
5. The Program adapter prepares the command for execution and passes it to the WebController for execution.
6. The Program adapter generates the proper XML response and returns the XML response to the external system that made the request.

When the Program adapter receives the XML request, it must verify the credentials of the external system that sent the request. Not all XML requests can be processed. Even if the XML request can be mapped to a WebSphere Commerce command, there must be some verification to ensure that the request should be processed.

Message Mappers

A Message Mapper is a mechanism that takes an XML message and converts it to a CommandProperty object. It provides a common interface so that messages can be converted to CommandProperty objects and used by all WebSphere Commerce components.

Instructor notes:

Purpose — Describe the main inbound messaging service messages included in WebSphere Commerce. **Details —** With the maturity of a product like WebSphere Commerce, you expect several connection alternatives. **Additional information —**

Transition statement — Next, Supporting inbound (XML) messages.

Supporting inbound (XML) messages

1. Define a DTD for a new inbound XML message.
 - Save to <WCS>/wc.ear/xml/messaging.
2. Add a DTD to the file system.
 - DTD can be registered by using WebSphere Commerce Configuration Manager.
3. Create or customize a command to handle message.
4. Update the user_template.xml for new message.
 - Models inbound message templates
5. Update XML configuration directory for EAR file.

© Copyright IBM Corporation 2016

In addition to the supported XML and WebSphere Commerce messages, you can add support for new inbound messages.

There are two primary methods for adding new Inbound messages.

The recommended method is to add a new inbound XML message by using the user_template.xml inbound message template definition file. In this file, you can indicate the controller command that the new inbound message invokes, define the elements of the message, and indicate the command parameters to which each element corresponds. When the message is received, the XML Message Mapper identifies the command to be run and the parameters to be used. The command is then invoked by using the site administrator authority. For security reasons, you must ensure that only authorized persons can access and modify the user_template.xml message template definition file; otherwise unauthorized users can write a new inbound message, and invoke any WebSphere Commerce command as site administrator.

If you do not want to use the inbound XML message template definition files together with the XML Message Mapper, you can also implement the NewInboundMessage command to add new messages. This command is invoked when the Message Mapper does not recognize the message as an existing Legacy message, or as an XML message defined in the inbound XML message template definition files. Since the NewInboundMessage command is not pre-programmed, you have full control over the processing that takes place once it is invoked. However,

this method requires considerable programming effort, particularly where there are several new messages.

WebSphere Commerce allows you to modify or extend the functionality of all inbound messages by modifying the WebSphere Commerce Controller command that each message runs. You can provide additional pre-processing or post-processing statements to any inbound message command used, or you can override the existing processing entirely. To do this, you need to know Java programming.

When an Inbound message is received from a back-end system, its information is processed into

command parameters and a WebSphere Commerce controller command is invoked along with all the provided parameters. When the command is run, the `performExecute()` method is invoked, which in turn invokes three methods, in the following order:

- `doPreProcess()`
- `doProcess()`
- `doPostProcess()`

When you first install WebSphere Commerce, only the `doProcess()` method contains programming statements. You can add pre-processing statements by extending the command and implementing the `doPreProcess()` method, or you can add post-processing statements by implementing the `doPostProcess()` method. Alternatively, you can implement either the `doProcess()` or the `performExecute()` method to overwrite the entire process.

Instructor notes:

Purpose — Details —

Additional information —

Transition statement — Next, Summarize web services in WebSphere Commerce

Web services in WebSphere Commerce

After completing this topic, you should be able to:

- Describe the relationship between IBM service-oriented architecture and WebSphere Commerce.
- Develop web services for WebSphere Commerce.
- Implement a web service for WebSphere Commerce.
 - Configure the message mapper.
- Add a web service to WebSphere Commerce.
- Access a web service from WebSphere Commerce.

© Copyright IBM Corporation 2016

Instructor notes:

Purpose — Initial topics start at a high level, and then drill-down to specific Developer implementation of web services in WebSphere Commerce.

Details —

Additional information —

Transition statement — Service-oriented architecture

REST Services

© Copyright IBM Corporation 2016

REST services overview

- Lighter weight than SOAP-based web services
- Easily adopted by various clients
- Use existing HTTP verbs
 - GET, POST, PUT, DELETE
- Provide responses in any Internet media type
 - JSON, XML, HTML, and so on.
- REST services help facilitate the invocation of:
 - “classic” controller commands
 - the activation of data beans
- The REST creates custom REST resource handlers that start controller commands to
 - create, update and delete operations
 - activate data beans to retrieve data

© Copyright IBM Corporation 2016

Notes:

REST services provide an alternate way of implementing a client/server communication model other than SOAP-based web services. These lighter weights, simpler services are easily adopted by a wide variety of clients. Some examples are discussed later in this presentation. Rather than defining an action verb within the service request as SOAP services do, REST services use the existing GET, POST, PUT, and DELETE verbs supported by HTTP. The response format back to the client can be any Internet media type. Common response types include JSON, XML, and HTML.

This unit is not intended to introduce all concepts of REST. It is focused on the implementation of REST by WebSphere Commerce.

Instructor notes:

Purpose — Details —

Additional information —

Transition statement — Next: REST services in WebSphere Commerce

Creating and Consuming REST APIs

- WebSphere Commerce provides a utility to create configuration-based controller command and data bean mappings.
- The REST utility auto-generates
 - The configuration mapping files
 - The sample handler code for your custom data bean or controller command
- You can create custom handlers to access to your own data beans and controller commands

© Copyright IBM Corporation 2016

Definitions

- Resource
 - Logical entity that can be created, retrieved, updated, and deleted
- Representation
 - The resource that is rendered in a specific format (e.g. JSON)
- Resource handler
 - A JAX-RS class that provides the various method handlers for a resource
 - Uses the @Path annotation to indicate which URLs the class handles
 - @Path("stores/{storeId}/cart")
- Data object mapper
 - A configuration file that transforms a resource representation to/from BOD attributes
- Entity provider
 - A Java class that transforms between Java types and representation formats

© Copyright IBM Corporation 2016

Notes:

The following definitions are useful for working with REST services:

Resource: Logical entity that can be created, retrieved, updated, and deleted

Representation: The resource that is rendered in a specific format(for example, JSON)

Resource handler: A JAX-RS class that provides the variousmethod handlers for a resource

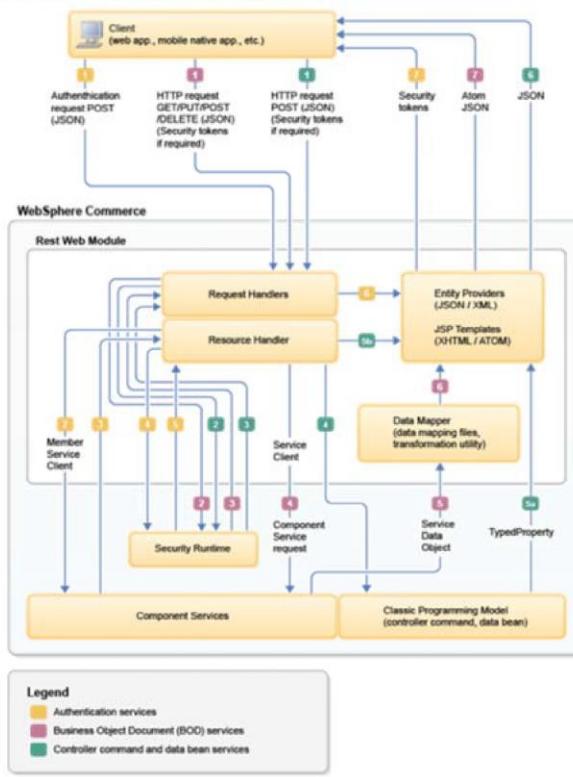
Uses the @Path annotation to indicate which URLs the class handles

 @Path("stores/{storeId}/cart")

Data object mapper: A configuration file that transforms aresource representation to/from BOD attributes

Entity provider: AJava class that transforms between Java types and representation formats

RESTful web service framework



© Copyright IBM Corporation 2016

Notes:

This diagram shows the interactions between the REST web module and other Commerce subsystems when handling authentication requests and regular requests.

There are two flows in the preceding diagram. Authentication Services:

1. Authenticates a registered user or creates a guest user.
2. Invokes a Member service request.
3. Returns an authenticated identity.
4. Creates authentication tokens.
5. Returns the WCToken and WCTrustedToken tokens.
6. Generates a response in the requested format.
7. Returns a response object to the client.

Business Services:

1. Makes a service request with security tokens.
2. Invokes request handlers and BCS.
3. Verifies security tokens by runtime.
4. Maps JAX-RS resources to WebSphere Commerce component services, generates, and sends service requests.
5. Returns the result as a Service Data Object (SDO).
6. Maps the SDO to the requested data format.
7. Returns the response object to the client.

Command and Data Bean Services:

1. Makes a service request with security tokens.
2. Invokes request handlers and BCS.
3. Verifies security tokens by runtime.
4. Maps JAX-RS resources to:
 1. WebSphere Commerce controller commands to perform the command.
 2. WebSphere Commerce data beans to perform data beanactivation.
5. Returns the results in:
 1. TypedProperty by controller command.
 2. TypedProperty by resource handler.
6. Returns the response object to the client.

JSON and XML response formats are supported by default.

Instructor notes:

Purpose — Details —

Additional information —

Transition statement — Next: Registering resources and providers

Calling REST services

- HTTP methods
 - GET, PUT, POST, DELETE
 - Use "X-HTTP-Method-Override" if PUT and DELETE are not supported
- Resource URI
 - http://hostname/wcs/resources/store/<storeId>/<resource_name>/<identifier>
- Input and output format is JSON
 - XML also supported (converted to JSON)
- Tips
 - For POST/PUT methods, always set content-type to application/json or application/xml
 - After authentication, pass WCToken for HTTP requests and WCToken and WCTrustedToken for HTTPS requests

© Copyright IBM Corporation 2016

Notes:

A client calls a REST service by identifying the resource that it wants to act on and the action it wants to take. The action is one of GET, PUT, POST, or DELETE. If PUT and DELETE are not supported, you can use the "X-HTTP-Method-Override" property in the header to specify the action. The resource to act on is specified by the URI. The URI has a fixed portion "/wcs/resources/store" followed by a client-specified portion. The client specifies the ID of the store for the request, the name of the resource and, if needed, an identifier for the resource.

The default input and output format for WebSphere CommerceREST services is JSON. XML is also supported as an input format but it is converted to JSON before the request is processed.

When you are passing input data in a POST or PUT request, make sure that you set the content-type attribute in the header to the Internet media type you are using. When making requests for an authenticated user, make sure that the authentication tokens are included in the header. The WCToken is needed for regular requests and both the WCToken and WCTrustedToken are needed for secure requests.

Instructor notes:

Purpose — Details —

Additional information —

Transition statement — Next: Caching service responses

REST service annotations

- Are annotated so you view and test the APIs with Swagger, an interactive REST service interface
- The REST Discovery API generates a list of REST resources and APIs by parsing annotations on a resource handler.
- WebSphere Commerce REST and Search REST services use the standard Swagger specification (version 1.2).
- WebSphere Commerce REST service and Search REST services are annotated with a combination of custom REST and JAX-RS annotations available from the Apache Wink

© Copyright IBM Corporation 2016

REST Customization Steps

1. Database preparation
 - You must first create a table and associated keys to gain access to the data bean
 - Create a controller command
2. Use the REST utilities to generate “configuration-based” XML mappings for your data beans and controllers
 - Using this RESTClassicSampleGen.bat utility
3. Create REST resource handlers
 - The resource handler represents the entry point
4. Customize WebSphere Commerce to use the new REST data
 - For example, a store widget or a page
5. Use the SWAGGER UI to annotate your REST resource Handler

© Copyright IBM Corporation 2016

Testing REST services

The screenshot shows the "Poster" plug-in interface for Mozilla Firefox. On the left, the "Request" tab is active, displaying a POST request to <https://myhost.com/wcs/resources/store/10001/loginIdentity>. The "Content to Send" section contains a JSON payload:

```
{
  "logonId": "tester1",
  "logonPassword": "tester11"
}
```

On the right, the "Response" tab shows the JSON response from the server:

```
{
  "WCToken": "507%2cg6c6e5Rb71b1z6XnJcm5i7S%2fwBWcmUdXW0IPEWtPSSiFCNQvInd6PKid8C58ndN31L09uD%2fRT4%2bH%0d%0aOpigjIW5zsIkPx3%2fG1cdqV1zNkxRN9mu2I%2fA4UnfNDU9t%2f4hJ8XIJJwxE6XZP00%3d",
  "WCTrustedToken": "507%2c3mujh5UiLbNo3HlsJ1f9Xu3Qyks%3d",
  "personalizationID": "1319834244560-4",
  "useId": "507"
}
```

The response also includes the following headers:

Content-Type	application/json
Vary	Accept
Content-Language	en-US
Content-Length	302
Date	Wed, 16 Nov 2011 23:25:29 GMT
Server	WebSphere Application Server:7.0

- “Poster” plug-in for Firefox
- Issue services requests and examine the response

© Copyright IBM Corporation 2016

Notes:

The Poster plug-in for Firefox is helpful for testing REST services. You can manually enter a service request that will include JSON content and then invoke the service. The JSON response is returned along with the response headers.

Other options exist for testing REST services. For the purposes of this course, Poster is selected for this course.

Instructor notes:

Purpose — Details —

Additional information —

Transition statement — Next: Checkpoint

Swagger

- The Swagger framework queries the REST Discovery API to display your REST resource handlers in alphabetical order.

The screenshot shows the Swagger UI interface for a REST API. At the top, there is a header with the Swagger logo and a search bar containing the URL "http://example.com/api". A green "Explore" button is also present. Below the header, a list of resource handlers is displayed in a table format:

Resource Handler	Description	Show/Hide	List Operations	Expand Operations	RA
aggregate	An aggregate of responses from multiple REST API requests.	Show/Hide	List Operations	Expand Operations	RA
api	Information about the resources and APIs available on the server.	Show/Hide	List Operations	Expand Operations	RA
approval_status	Accurate, secure transactions require that a second individual approve some electronic marketplace actions before they proceed. This individual, called an approver, can accept or reject requests to perform a specific action. This class provides RESTful services to retrieve order and buyer approval status record details and to approve or reject them.	Show/Hide	List Operations	Expand Operations	RA
assigned_coupon	This class provides RESTful services to add, get, update, and delete assigned coupons for the current shopping cart.	Show/Hide	List Operations	Expand Operations	RA
assigned_promotion_code	This class provides RESTful services to add, get, update, and delete assigned promotion codes for the current shopping cart.	Show/Hide	List Operations	Expand Operations	RA
associated_promotion	This class provides RESTful services to access retrieve associated promotions.	Show/Hide	List Operations	Expand Operations	RA
cart	This class provides RESTful services to add, update, delete, and get items in the shopping cart and checkout. It also provides services to get usable shipping and payment information by delegating to the Order BOD service.	Show/Hide	List Operations	Expand Operations	RA

At the bottom right of the interface, there is a copyright notice: © Copyright IBM Corporation 2016.

With the supported REST annotations in place, the REST Discovery API generates resource listings and API declarations that are used by Swagger to present the RESTful API in the form of interactive documentation. When you are viewing this documentation, you can discover the available REST resources and methods, and test the API from inside the Swagger interface.

The Swagger user interface

The Swagger interface is composed of static HTML, CSS, and JavaScript, and it is accessed through your web browser. The Swagger framework queries the REST Discovery API to display your REST resource handlers in alphabetical order. The following screen capture shows the Swagger interface, specifically the sampleresource handler that was annotated earlier:

Swagger

The screenshot shows the Swagger UI interface for a REST API. At the top, there's a green header bar with the Swagger logo and a search bar containing the URL "http://example.com/wcs/resources/api". Below the header, there's a sample description of a REST handler. The main area contains two sections: one for a GET method and one for a POST method. Each section includes a brief description, a code snippet in JSON or YAML, and a table for parameters. The parameters table has columns for Parameter, Value, Description, Parameter Type, and Data Type. A note at the bottom right indicates the copyright belongs to IBM Corporation 2016.

sample : This is a very simple REST handler that extends the configuration-based REST classic handler. It demonstrates how to create REST resources based on a DataBean and a ControllerCommand with minimal coding, as all the input/output information is stored in a configuration mapping file.

GET /store/{storeId}/sample/catalogEntry
A sample REST GET that executes the CatalogEntryDataBean based on the request parameters and profile specified.

POST /store/{storeId}/sample/catalogEntryUpdate
A sample REST POST that executes the CatalogEntryUpdateCmd based on the request parameters and profile specified.

Response Class
Model : Model Schema

```
com.ibm.commerce.rest.classic.sample.SampleCatalogEntryClassicRestHandler$CatalogEntryUpdateResponse {  
    catentryId (string, optional): The catalog entry identifier..  
    partnumber (string, optional): The catalog entry part number.  
}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
storeId	0	The store identifier.	path	string
responseFormat	application/json	The response format. If the request has an input body, that body must also use the format specified in "responseFormat". Valid values include "json" and "xml" without the quotes. If the responseFormat isn't specified, the "accept" HTTP header shall be used to determine the format of the response.	query	string

© Copyright IBM Corporation 2016

Where each resource handler class has a short description of the service.

Show/Hide expands or collapses the API content of each class.

List Operations displays a summarized view of the API content, where a row is displayed for each method's path, with a short description of the service.

Expand Operations displays an expanded view of the API content, where rows are displayed for each method's response class, parameters, and status codes. The API can be dynamically tested by selecting **Try it out!** in the expanded view.

Raw displays the raw information that the Swagger UI uses to populate the page.

Enabling and disabling the REST Discovery API

You might want to prevent Swagger from displaying your RESTful API in a web browser, especially in a production environment that is serving live traffic. By disabling the REST Discovery API, you can prevent your REST services from being retrieved and displayed in Swagger. Alternatively, you can enable the REST Discovery API to view and test your REST services with Swagger.

Service-oriented Architecture

© Copyright IBM Corporation 2016

Service-oriented architecture

- Architectural style that is used to build distributed systems that deliver application functionality as services to user applications or other services.
- It does the following tasks:
 - Leverages open standards to represent software assets as services
 - Provides a standard way of representing and interacting with software assets
 - Allows individual software assets to become building blocks that can be reused in developing other applications
 - Shifts focus to application assembly rather than implementation details
 - Creates new applications from existing components (internal use)
 - Integrates with applications outside of the enterprise (external use)

© Copyright IBM Corporation 2016

Service-oriented architecture provides the concept for describing, programmatically discovering, and using operations that are packaged as services. SOA is being used in various forms for many years. SOA is implementing various distributed computing technologies such as CORBA or messaging middleware.

In the past, the ability of the underlying technology to interoperate across the enterprise limited the effectiveness of service-oriented architectures. The key goal of a service-oriented architecture is to achieve loose coupling between interacting software agents.

The next question that typically arises is "What is a service?"

Instructor notes:

Purpose — Describe service-oriented architecture (SOA).

Details — One of the key disadvantages of other SOA-like architectures was that there was still a level of coupling between the client and server. For example, CORBA required compatible ORBs, and the interface was defined in the OMG's IDL language. CORBA is designed more for Distributed Object Communication. It was suited for tightly coupled transactional systems, but it was not considered to be flexible.

SOA with web services is focused on business-process interoperability and documents. Web services with SOAP RPC are fairly simple and easy to understand, particularly for someone who comes from a CORBA background. Designing an SOA with a supporting infrastructure that uses document-based communication is not necessarily as simple.

Additional information —

Transition statement — Next, look at defining a service.

Defining a service

- “A service is an application function that is packaged as a reusable component that can be used in a business process.”
- Services provide the following features:
 - Expose a well-defined interface.
 - Appear as a self-contained function.
 - Use messages to hide implementation details.
 - Operate independently, despite the state of other services.
 - Provide information to a requester or facilitates a change to business data from one valid, consistent state to another.

© Copyright IBM Corporation 2016

Services are generally considered to have a component appearance from the perspective of a service requester. They should be loosely coupled. Requesters should need to understand only the content of the messages that are going back and forth and should not need to understand the underlying APIs, file formats, or platforms that the service uses. Services should have a well-defined interface that is independent of any particular implementation technology. Finally, services should be stateless.

Every message that is sent must contain enough information for the provider to process it. While stateless resources are the norm, there is a standard, WS-Resource, defined for Stateful Web services. For more information about WS-Resource, see the document at http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf.

Services can be coarse-grained, fine-grained, or somewhere in between. A coarse-grained service is one that exposes a high-level business function that, when invoked, calls many other internal services. A fine-grained service is one that implements a specific function and only that function. Collaboration is needed between a company's IT development and business management to define services. At a high level, a coarse-grained service would represent a business process. This is not a trivial part of SOA design.

Software component: "Software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system." (*Component Software: Beyond Object Oriented Programming* by Clemens Szyperski)

Instructor notes:

Purpose — Describe a service.

Details — With a loose coupling service, an interface is not tied to a particular platform.

Additional information —

Transition statement — Next, Web service development approach

Development approach

- Extending Commerce with web services is a useful method of integrating Commerce with supporting systems.
- Defining a service that allows Commerce to integrate with existing systems (by using either inbound or outbound messages) is also useful in cases where it is necessary to support parallel development.
 - For example, during development of a store, another team might be working on an order fulfillment and shipping application.

© Copyright IBM Corporation 2016

Instructor notes:

Purpose — Outline the use of web services in a WebSphere Commerce application. Why use web services?

Details —

Additional information —

Transition statement — Next, Common uses of web services

Common uses of web services

- Interoperability between heterogeneous systems
- B2B interactions
 - Lower cost than EDI and proprietary alternatives
- Multiple requesters
 - Particularly for heterogeneous platforms, protocols, and channels
- Encapsulation of legacy systems
- Isolation from changing systems
- Enterprise standardization of services
- Integration of third-party packages
 - Tax packages, shipping calculation, order fulfillment
- Process choreography
- Portal development

© Copyright IBM Corporation 2016

B2B: Business-to-Business

EDI: Electronic Data Interchange defined standardized formats for electronic data exchange between businesses that required the use of proprietary Value-Added Networks (VANs).

Process choreography is the coordination of the execution of Business Processes in a particular sequence. Portal is a web-based application that serves as an entry point to a set of tools and applications.

Instructor notes:

Purpose — Describe the common uses for web services.

Details —

Additional information —

Transition statement — Next, look at the IBM view of web services.

Types of web services available

- WebSphere Commerce supports the following web services
 - Web service framework using the WebServicesRouter web module
 - WebSphere Commerce service module (SOI/BOD Component service Model)
 - SOI uses controller commands\data beans\access beans
 - BOD uses pattern matching and DSL
 - Representational State Transfer (REST) services

© Copyright IBM Corporation 2016

Notes:

SOI – Service oriented integration

Comparison of SOI and BOD service modules in WebSphere Commerce:

https://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/csdcompare.htm

REST services are JAX-RS REST services also known as the WebSphere Commerce REST API is built on top of Apache Wink. Further details discuss later in this lecture.

https://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.webservices.doc/concepts/cwvrest.htm?view=embed

Instructor notes:

Purpose — Explain the various types of web services supported by WebSphere Commerce

Details — Introduce the various types of web services. The Web service framework, SOI, BOD, and REST services are discussed in further detail later in the lecture.

Additional information —

Transition statement — Goals of WebSphere Commerce Services

Goals of WebSphere Commerce web services

- To promote structured WSDL descriptions to define objects that are used for invoking services
- To promote the adoption of industry-accepted standards for service definitions, such as OAGi
- To achieve greater interoperability with other systems that adopted the same standards
- To promote separation between client and server through publishing XML schema of data objects used
- To promote use of web service development tools, such as WebSphere Commerce Developer or Rational Application Developer, for developing web service clients that can communicate with WebSphere Commerce

© Copyright IBM Corporation 2016

• WSDL definition should define all services.

Services are explicitly declared. Use an industry standard service definitions.

• Existing command pattern is still used to represent the business logic.

No new paradigm introduced to support web service requests. The web service channel can also use existing URL-based controller commands.

• Client and server are independent.

The server can be replaced and support the same WSDL. Any client can communicate with the server through the defined service definition.

• Tooling support:

- Leverage existing web service tooling.

- Platform and language independent.

- .NET client can act as a client.

Order Processing

The order preprocessing service is an inbound service for orders that are submitted and transferred to an external Order Management System (OMS) but are modified post-transfer and require preprocessing by WebSphere Commerce before external Order Management System can process them.

Instructor notes:

Purpose — Describe the inbound included web services.

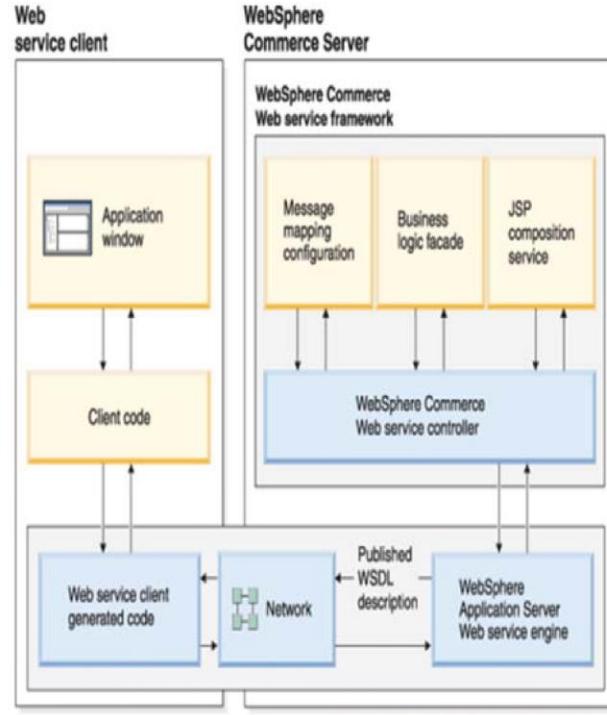
Details —

Additional information —

Transition statement — Continue looking at service provider information for deployment of WebSphere Commerce web services.

WebSphere Commerce as a Provider

- WebSphere Commerce runs on WebSphere Application Server.
 - Has the same access to web services support as any application that runs on WebSphere
 - Uses standard Rational Application Developer web services creation tools
- Additional WebSphere Commerce support for web services:
 - Extra runtime layer that simplifies the process of exposing WebSphere Commerce controller commands as web services
 - Uses message mapping to map service requests into command framework



© Copyright IBM Corporation 2016

When you enable business operations in WebSphere Commerce as web services that the external systems can access, WebSphere Commerce becomes the service provider.

The deployment model for web services is to have a central server with published WSDL that defines the services it supports. From there, clients, whether they are external systems, web applications, or rich client applications, connect to the central server and invoke services the publicly available WSDL define.

The WebSphere Commerce web service framework uses the approach of mapping the XML (SOAP body) request into the name-value pair parameters that are passed to the service command that is executed.

In terms of overall web service request handling, the WebSphere Application Server Web Service Engine is responsible for delegating the request to the WebSphere Commerce web service framework, and the framework is responsible for processing the request and generating the response. Request processing consists of resolving credentials to associate with the request, converting the SOAP body into name-value pairs, mapping to a controller command, executing the command, and using the JSP composition service to create the response.

From the high-level perspective, success and application exceptions are handled in a similar fashion. The only difference is the JSP page that is used to compose the response: when an exception occurs, the error view is used to determine the JSP page that composes the response. For example, taking the OAGi approach, the resulting document for the request would contain the status of the operation. This means that if an application exception occurs (that is, command execution results in an error), the resulting Business Object Document would still be created, but indicate failure. A Web Service Fault is returned only if an external error, such as an exception or a SOAP Fault, returned by the JSP page, occurs when preparing the request for processing.

Note: Standard OAGi messages are not supported. Although the OAGi style of interaction is used, WebSphere Commerce adopts only the OAGi message envelope and verbs. Simplified nouns that better represent the structure of how the business objects are modeled in WebSphere Commerce are

provided.

Enabling WebSphere Commerce as a provider

1. Identify business logic to expose as a web service.
2. Identify controller command that represents business logic.
 - You might need to create a new controller command or nest commands.
 - New commands must be registered in the command registry and deployed.
3. Identify mandatory and optional parameters for the command.
4. Create a WSDL description for the web service.
 - Use tooling as available.
5. Register the WSDL description.
6. Compose a response (a JSP, normally).
7. Deploy the service.
 - Include related files and resources.

© Copyright IBM Corporation 2016

Instructor notes:

Purpose — Details —

Additional information —

Transition statement — Next: WebSphere Commerce as a service consumer

WebSphere Commerce as a service consumer

- WebSphere Commerce uses messaging to invoke Remote Web services.
 - Web service over HTTP connector
 - Web service over JMS connector
- Component client API called from (task) command.
 - API uses invocation service.
 - Invocation service requires deployed configuration files to determine communication.
 - Each component has a separate configuration file.
- Access external web services from commands in standard Java EE ways.
 - JAX-RPC proxies
 - SAAJ



© Copyright IBM Corporation 2016

When WebSphere Commerce acts as a service consumer, a component client API is called from your task command. The client API uses the invocation service and the invocation service requires a deployed configuration file to determine how to communicate with the remote component. Each component has a separate configuration file to configure the client API. Each store can also have a version of the configuration file that takes precedence over the default configuration.

This allows the store to override some or the entire configuration without changing the default configuration.

Web service over HTTP connector

This connector will send web service requests over HTTP. The web service HTTP JCA connector wraps the web service request message inside a SOAP envelope and allow the security credentials to be specified. The additional SOAP header information is the interaction specification properties that are associated with the web service JCA configuration. The JCA sets these properties when constructing the SOAP message. Additionally, this connector throws exceptions when SOAP faults are returned as part of the service request.

As a feature of this particular connector, when an exception occurs when executing the request based on the connection properties, this connector waits for a specified duration before allowing the same request to execute again. This feature prevents the back-end system from being flooded with a series of requests when it is unavailable or unresponsive. Certain communication exceptions are caught and cached and assigned a timeout value. When requests based on the same connection information are made, the cached exception is thrown instead of executing the request. When the timeout expires, the request is made as usual. The purpose of this feature is to provide the caller with an exception while trying to avoid making repeated requests to a back-end system that currently is not responding.

Web Service over JMS Connector

The web service JMS JCA is an extension of the current JMS messaging JCA Connector that wraps the message inside a SOAP envelope and allows security credentials to be specified. This connector throws exceptions when SOAP faults are returned as part of the service request.

Enabling WebSphere Commerce as a consumer

- 1.** Identify the external system.
 - If not already provided, create a web service on the external system to provide the business logic.
- 2.** Customize the web service client invocation XML file.
- 3.** Create a web service client API for WebSphere Commerce to construct the outbound message.
- 4.** Identify commands that consume the service.
 - A task command should call the client API to send the message.

© Copyright IBM Corporation 2016

If the external system does not use the same WSDL, but has an existing service API defined, mediation is required to transform outbound service requests and responses between the two formats.

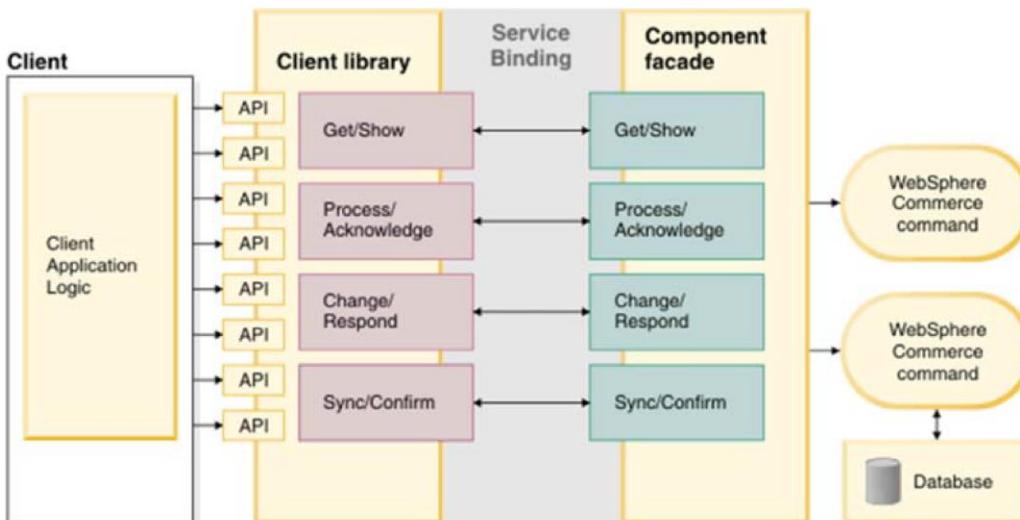
When WebSphere Commerce acts as a service consumer, a component client API is called from your task command. The client API uses the invocation service and the invocation service requires a deployed configuration file to determine how to communicate with the remote component. Each component has a separate configuration file to configure the client API. Each store can also have a version of the configuration file that takes precedence over the default configuration.

This allows the store to override some or the entire configuration without changing the default configuration.

For each component that is to consume external web services, a client API is required to construct different request messages and then call the invocation service. The client API is essentially a Java utility class that contains construction code to create the appropriate message and send it. No business logic is performed in the client API. Using the client API to construct the message simplifies the business logic by providing a programming friendly interface. This allows other Java applications to use the same code to make remote calls when the invocation service does not depend on the WebSphere Commerce Runtime environment.

Web services and BOD messages

- Service interfaces are defined by using the BOD message structure.
- BODs provide a consistent message structure and model for messaging by using XML.
 - BODs are independent of communication mechanisms.
 - Only nouns and verbs are used, not standard OAGi messages.



© Copyright IBM Corporation 2016

WebSphere Commerce Service interfaces are defined by using the OAGi message structure, from the Open Applications Group. The OAGi standard provides a consistent message structure and model for messaging using XML. The OAGi standards describe Business Object Documents (BODs) as nouns and the services that interact with those nouns are called verbs.

Although the OAGIS style of interaction is used, WebSphere Commerce adopts only the OAGIS message envelope and verbs. Simplified nouns that better represent the structure of how the business objects are modeled in WebSphere Commerce are provided. Although OAGIS might provide a noun definition for the same business object, WebSphere Commerce supports only a small percentage of the structure OAGIS provides. Instead of reusing the OAGIS definition with limited support, WebSphere Commerce defines business objects that better represent what is supported.

Creating WebSphere Commerce services (BOD style)

- Determine if a new service is needed or an existing service can be extended
- If business logic is already created, use the JSP page model with WebSphere Commerce Services since you have most of the controller commands written
- If the business logic does not exist, develop the logic as a component service and expose the functionality as a web service
- When custom information is required, there are three options:
 - Include custom information in the UserData area of an existing noun

© Copyright IBM Corporation 2016

Guidelines for using BOD-style web services

- Only a WebSphere Commerce service module should use its external facade client for outbound requests
 - For example, only the Order service module should use the order external facade client
- Using InsertMore commands:
 - InsertMore commands enable you to include change control information in the response of a Get service
 - Example: InsertMoreCatalogEntryDataCmd for products (catalog entries)
 - Use InsertMore commands to retrieve data from an external system and include the data in a response
 - Try to batch the external system request into one request and merge the result set back together

© Copyright IBM Corporation 2016

If a Client application or another service module requires order information, the default order facade client should be used to make a service request to the WebSphere Commerce order service. The WebSphere Commerce Order service would then make an outbound request if necessary.

The Get request access profile determines whether Change Control information is returned as part of the response. The InsertMoreData command registration point registers a generic implementation that uses the Business Object Mediator to resolve and include Change Control information. To include Change Control information in the response of a Get service:

1. Identify the InsertMoreData command interface name that the service implementation uses to resolve the command implementations to include additional information in the Show response. Typically the InsertMore command interface is in the same Java package as the GetNounNameCmd interface with a class name of InsertMoreNounNameDataCmd. For example, the InsertMore command interface for the CatalogEntry noun is com.ibm.commerce.catalog.facade.server.commands.InsertMoreCatalogEntryDataCmd.
2. Identify the Access profile in which to include Change Control metadata. Typically, the access profile is an access profile that is used for authoring purposes. The naming convention is IBM_Admin_Details. To identify the Access profile, see the Get expression that is used by the authoring JSP file.
3. Create an entry in the Command Registry table (CMDREG) that registers the InsertMore command that is identified in Step 1 with the access profile identified in Step 2 to call the generic InsertMore to include the Change Control metadata with the response. The following sample registration statement registers the InsertMore Change Control metadata for CatalogEntry Get requests when the Access profile is IBM_Admin_Details.

```
insert into cmdreg (storeent_id, interfacename, classname, target) values
(0, 'com.ibm.commerce.catalog.facade.server.commands.InsertMoreCatalogEntryDataCmd+IBM_Admin_Details.0',
```

```
'com.ibm.commerce.foundation.server.command.bod.bom.  
InsertMoreNounChangeControlMetaDataCmdImpl', 'Local');
```

Because multiple InsertMore commands can be registered for the same Access profile, the *.index* notation is used to allow for more than one InsertMore command to run and the order is based on the registered index.

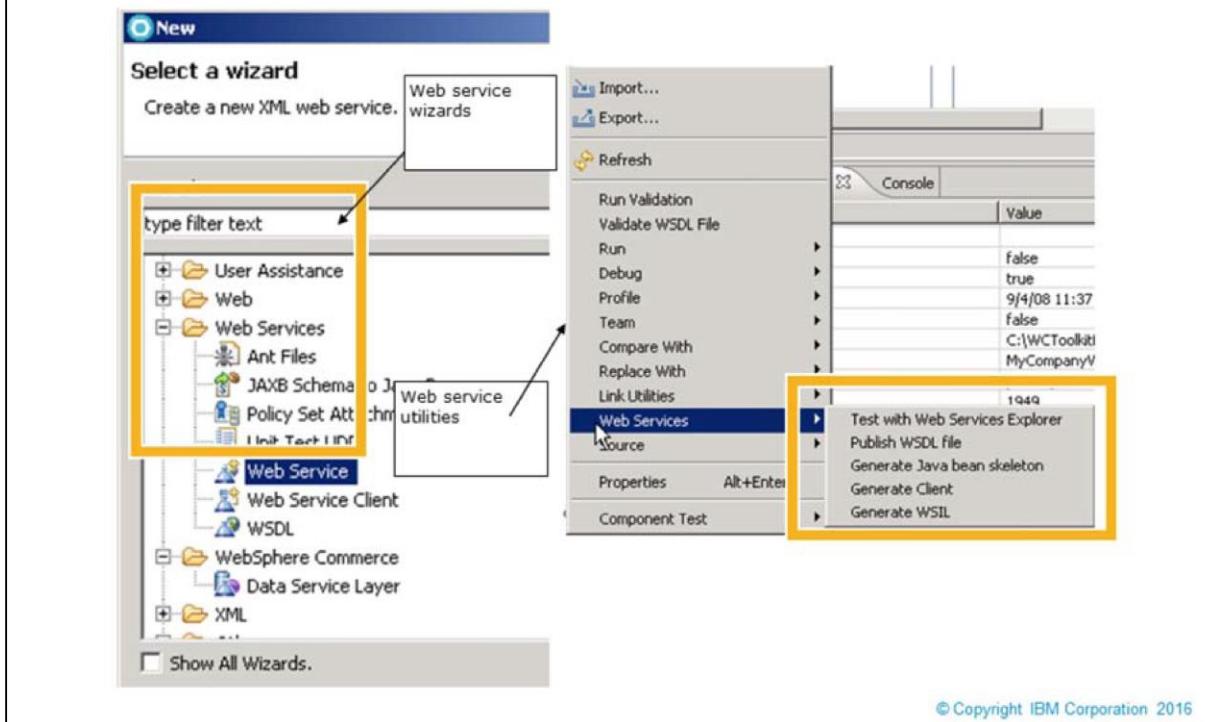
Instructor notes:

Purpose — Details —

Additional information —

Transition statement — Next, IBM IDE Support for web services

IBM IDE Support for web services



© Copyright IBM Corporation 2016

WebSphere Commerce Developer supports web service development.

The WebSphere Commerce Developer tools handle much of the complexity of web service development, enabling you to focus on the complexity of the Business Process.

To use WebSphere Commerce Developer to develop web services, you must enable web service and, optionally, XML capabilities. By default, these capabilities are disabled because they are used only for developing web services or web service clients.

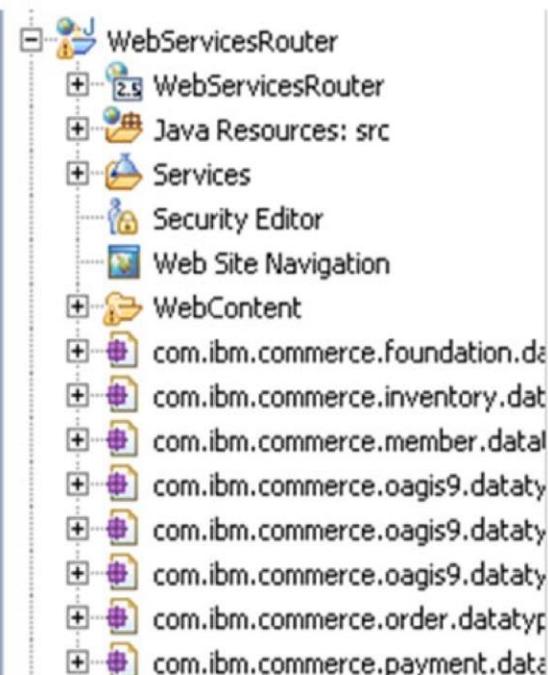
To enable the web service and XML capabilities:

- Select **Window > Preferences**.
- In the Preferences window, expand **Workbench > Capabilities**. In the right pane, select **Web Service Developer**.

- Select **XML Developer**. This capability provides additional XML development features.

Developing web services

- WebServicesRouter project
 - Contains the WSDL, XML, and JAX-RPC mapping of services
 - Required by web service engine to map request to generic endpoint
 - Enables Web services for HTTP
- Defining new services
 - Create new WSDL file
 - Create new JAX-RPC mapping file
 - Update any dependent files
- Extending existing WSDL
 - Add operation to appropriate WSDL
 - Modify JAX-RPC mapping files
 - Adding a parameter to an object used in a service operation requires updates to the object's XML schema file
- WebSphereCommerceExtensions-Logic contains business logic



© Copyright IBM Corporation 2016

While the web service plumbing resides in the WebServicesRouter project, as with all of Commerce, the Business Logic resides in the WebSphereCommerceExtensionsLogic project. The point of this is to reuse the commands that are already written and to perform the same Business Logic no matter what interface (Commerce GUI, web services, and so on) is used.

Instructor notes:

Purpose — Continue describing the IDE support for web services in Rational Application Developer.
Details —

Additional information —

Transition statement — Next, continue with MessageMapping.

Web Service message mapping

- Inbound XML messages must be mapped to WebSphere Commerce controller commands
- Individual XML elements must be mapped to entries in command's RequestProperties
- Define command and property mappings in **webservice_*_template.xml** files

© Copyright IBM Corporation 2016

When a SOAP message comes in, the message must be mapped to a command. All the individual elements of the message must also be mapped to the properties the command is expecting in the RequestProperties object.

The configuration for this mapping is in the `webservice_*_template.xml` files.

Instructor notes:

Purpose — Describe the process for mapping a Service to a Command.

Details —

Additional information —

Transition statement — Next, look at a partial example of Message Mapping.

Sample Message Mapping Template

```
<TemplateDocument>
  <DocumentType>SyncCustomer</DocumentType>
  <StartElement>Customer</StartElement>
  <TemplateTagName>SyncCustomerMap</TemplateTagName>
  <CommandMapping>
    <Command CommandName="com.ibm.commerce.usermanagement.commands.UserRegistrationAddCmd">
      <Constant Field="URL">noURL</Constant>
      <Constant Field="viewTaskName">ConfirmSyncCustomerSuccessBOD</Constant>
      <Constant Field="errorViewName">ConfirmSyncCustomerErrorBOD</Constant>
    </Command>
  </CommandMapping>
</TemplateDocument>

<TemplateTag name="SyncCustomerMap">
  <Tag XPath="DisplayName" Field="displayName"/>
  <Tag XPath="Authentication/Logon" Field="logonId"/>
  <Tag XPath="Authentication/Password" Field="logonPassword"/>
  ...
  <Tag XPath="Contact/FirstName" Field="firstName"/>
  <Tag XPath="Contact/LastName" Field="lastName"/>
</TemplateTag>
```

© Copyright IBM Corporation 2016

1. Notice in the TemplateDocument section that the StartElement handled by this mapping is Customer.
2. When an XML message comes in with this Customer StartElement, it is mapped to the command with the interface name of com.ibm.commerce.usermanagement.commands.UserRegistrationAddCmd. This interface will be used to find the actual Implementation class in the CMDREG database table.
3. The SyncCustomerMap is used to map the XML elements to the properties found in the RequestProperties parameter sent to the setRequestProperties method of the UserRegistrationAddCmd.

Instructor notes:

Purpose — Describe how the SOAP message is mapped to a Command and the elements of the RequestProperties.

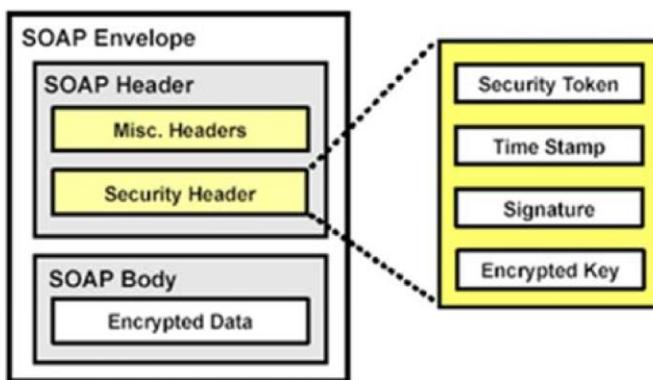
Details —

Additional information —

Transition statement — Next, look at the web service security.

Web services and credentials

- WebSphere Commerce web services support
 - WS-Security Basic Authentication
 - SOAP Message Security 1.0
 - OASIS standard
- When consuming web services
 - All WebSphere Application Server WS-Security supported specifications
 - SOAP Message Security 1.0
 - UsernameToken Profile 1.0
 - X.509 Certificate Token Profile V1.0



© Copyright IBM Corporation 2016

The WS-Security specification (<http://www.oasis-open.org/committees/wss/>) defines several techniques for implementing Web service security.

By default, all Web service requests are given the authentication of the generic user. However, since much of the WebSphere Commerce business logic that is exposed through web services cannot be executed by the generic user, WS-Security can be leveraged to allow different credentials to be used.

Basic Authentication

One approach to web service security is *Basic Authentication*. Under this approach, the user's credentials are attached to the header of the SOAP envelope. The chief shortcoming of Basic Authentication is that the authentication information is found in plain text within the SOAP message and, if the underlying transport protocol is not secure, is easily accessible by an attacker with appropriate network monitoring tools. As a consequence, Basic Authentication should always be used in conjunction with a secure underlying transport protocol, such as HTTPS.

The following is an example of a SOAP request that uses Basic Authentication:

```

<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding"
                   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/'>
      <wsse:UsernameToken>
        <wsse:Username>DavidB</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/'>
        </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
</soapenv:Envelope>
```

```
</wsse:Security>
</soapenv:Header>
<soapenv:Body>
<!-- message pay load -->
</soapenv:Body>
</soapenv:Envelope>
```

In compliance with the latest WS-Security specification, the user credentials are found within the Security Node of the SOAP header. This node is the placeholder for the security information that should be associated with the web service request.

When Basic Authentication is used to authenticate a user, the WebSphere Commerce Web Service Controller extracts the credential information from the SOAP header and call the Business Context Service to begin a new session that is based on the credentials found. This establishes an activity that is associated with the specified user, but the activity exists only for the lifetime of request processing. It is assumed that the activity is not needed to process subsequent requests, and the activity token is not returned to the calling client. After the request has been serviced, the activity is completed and can no longer be used for other requests.

Instructor notes:

Purpose — Describe the basic web services authentication and discuss other methods available.

Details — WebSphere, activity token, and protocol authentication are also available.

Additional information — When WebSphere Global Security is enabled, the WebSphere Application Server Web Service Engine authenticates the user and places their credentials on the thread. The Web Service Controller will, in turn, create a temporary activity with the user ID obtained from the thread authentication context.

This approach leverages the WebSphere security and Web Service Engine to authenticate the user and requires no additional implementation besides that for reading the credentials from the thread.

The **activity token** is a way to establish a session for a normally session-less protocol.

WebSphere Commerce web service framework leverages the custom token pluggable architecture that is provided by WebSphere Application Server Web Service Engine to place the activity token into the SOAP header of the request. The only difference between this form of authentication and those discussed previously is that the activity is not completed after the request is serviced. The activity is kept alive to be used for subsequent requests.

The use of **protocol authentication** is to have a generic flag such that if the credentials are not found on the Web Service Requests and this flag is set, instead of executing the request with the authority of a generic user, the default messaging user is used. This generic flag, named **secureTransportProtocol**, is found in the Messaging section of the WebSphere Commerce configuration file. When its value is set to true, the Web service framework assumes that the protocol is configured to provide validation of the request credentials and the user ID for the activity will be that of the default messaging user.

Transition statement — Next, Checkpoint

Checkpoint

1. What command-line utilities can be used to load data in to a WebSphere Commerce database?
2. In which project should customized web service logic be placed?
3. Which three areas are customizable when using web services with WebSphere Commerce?

© Copyright IBM Corporation 2016

Instructor notes:

Purpose — Details —

Additional information —

Transition statement — Next, Checkpoint solutions

Checkpoint solution

1. The following command-line utilities can be used: data load utility, data extract utility.
2. WebSphereCommerceExtensionsLogic (used for logic, WebServicesRouter is used for WSDL files, and so forth).
3. Message mapping configuration, business logic (BOD and controller commands), response composition (JSPs).

© Copyright IBM Corporation 2016

Instructor notes:

Purpose — Provide the checkpoint solutions.

Details —

Additional information —

Transition statement — Next, Exercise introduction

Exercise 5: Version 8 REST Services

© Copyright IBM Corporation 2016

Unit summary

This unit was designed to enable you to:

- Explain the messaging services architecture and define predefined messages.
- List the features in WebSphere Commerce that support integration.
- Describe the administration and use of integration features.
- Describe the process of developing new behaviors to support integration.
- Define a service-oriented architecture (SOA) and explain the role of web services in SOA and WebSphere Commerce.
- Develop a web service for use in WebSphere Commerce.
- Deploy a web service in WebSphere Commerce.

© Copyright IBM Corporation 2016

Instructor notes:

Purpose — Details —

Additional information —

Transition statement —

Additional training

For additional training needs, you have several options available for registering.

You can register	At
On site	This or any of our training facilities
Online	http://ibm.com/training/
Email	ibmswedu@us.ibm.com
By telephone	1-800-IBM Teach (1-800-426-8322)
By fax	208-692-6130

© Copyright IBM Corporation 2016

For additional training needs, you have several options available for registering like Onsite, Online , Email , by telephone or by Fax .

Unit 9

The slide features a blue header bar with the text "IBM Training" on the left and the IBM logo on the right. The main content area has a light gray diagonal striped background. In the center, the text "WebSphere Commerce V8" is on one line, and "Introduction to WCBD" is on the line below it, both in a dark blue font. At the bottom right of the slide, there is a small copyright notice: "© Copyright IBM Corporation 2016".

This section describes an overview of the WebSphere Commerce V8 Introduction to WCBD.

Unit objectives

This module is designed to provide you:

- An overview of the WebSphere Commerce Build and Deployment Tool
- Understand how to configure and run the Build process
- Understand the Toolkit Deployment process
- Steps for customizing the toolkit deployment process

© Copyright IBM Corporation 2016

This course is designed to enable you to understand the WebSphere Commerce version 8 Build and Deployment Tool. Following sections gives the understanding of how to configure and run the build process. You will also receive an overview of the Toolkit deployment process and the commands used. You will learn how to customize the toolkit deployment process and an overview of WCBD for Commerce on Cloud.

WebSphere Commerce Build and Deployment Tool Overview

- Based on WebSphere Commerce best practices and satisfies most build and deployment needs
- Works with SCM systems, including CVS, Subversion, Rational
- Supports deployment of:
 - Customized database assets
 - Java EE assets to WebSphere Application Server
 - Web server assets
- Build packages > repeatedly deployed to multiple environments:
 - Production
 - Staging and testing
 - Sandbox and development
- Provides logging and notification
- Highly customizable using Apache Ant scripts

© Copyright IBM Corporation 2016

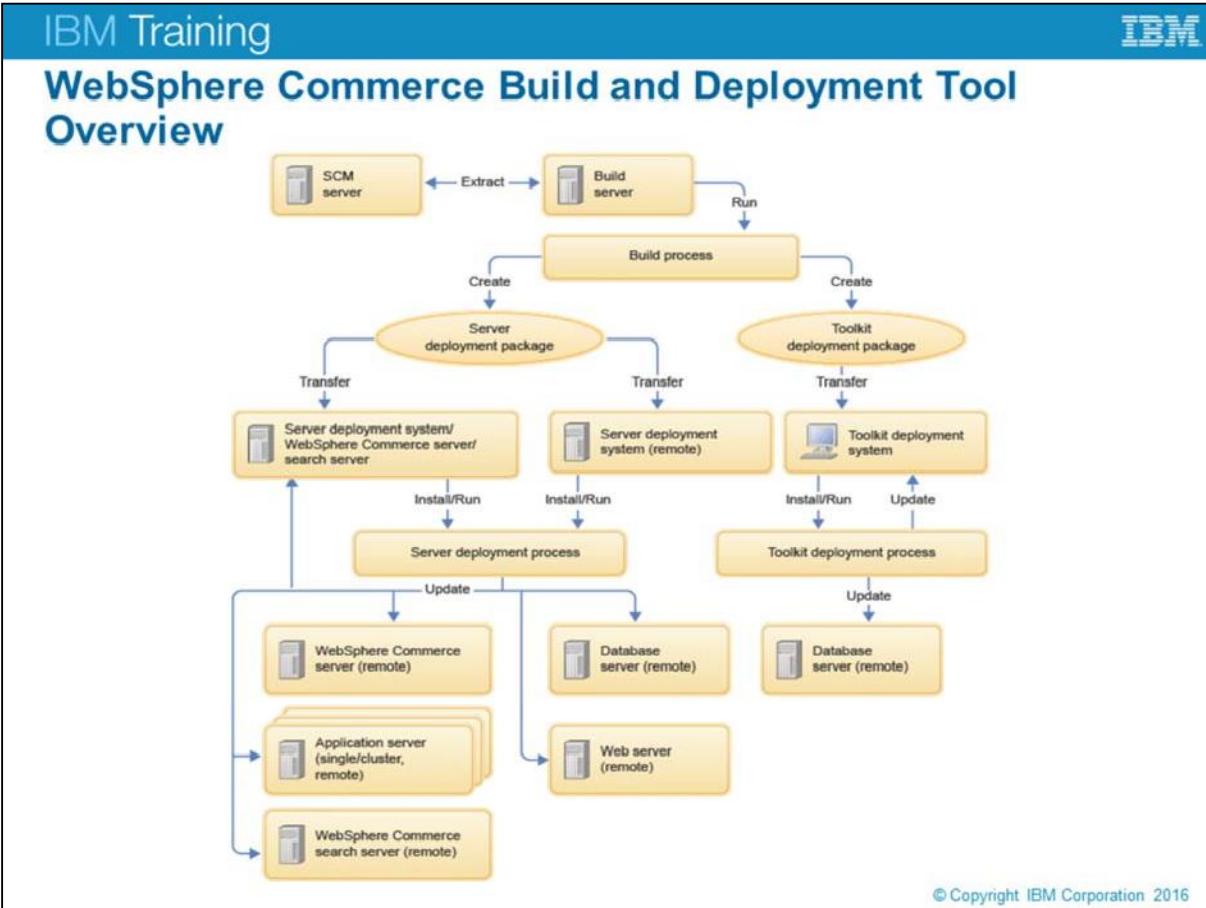
WebSphere Commerce Build and Deploy tool (WCBD) is a tool that works with the developer tools. Administrators might use the output of the tool to deploy the changes. It allows a full deployment of all code that is contained in the code release or it can perform a partial deployment.

When a partial deployment is complete, the WCBD system tracks the changes and deploys only new content.

The WebSphere Commerce Build and Deployment tool has the following features:

- Designed based on best practices that are found in the WebSphere Commerce Information Center.
- By default, the tool satisfies most typical build and deployment needs, leaving only few setup steps for a quick startup.
- The build process works with Source Configuration Management (SCM) systems, including CVS, Subversion, and Rational ClearCase. It can be extended to support other SCM systems.
- The tool supports deployment of customized database assets through SQL files and XML files that work with the WebSphere Commerce loading utilities (massload, idresgen, acupload, acpnlsload, acugload).
- It supports deployment of Java EE assets to the WebSphere Application Server by using the wsadmin tool. Rollout update for clustered environment is supported.
- It supports deployment of web server assets using FTP, SCP, or SFTP, and can be extended to support other transfer/deployment methods.
- Built packages can be repeatedly deployed to multiple target environments such as your production, staging, testing, sandbox, and development environments.
- It provides logging and mail notification functionality, which helps in problem determination.
- It is highly customizable by using Apache Ant scripts to suit project-specific needs. The default scripts are modular, and can be reused to save time when you are customizing your build and

deployment process.

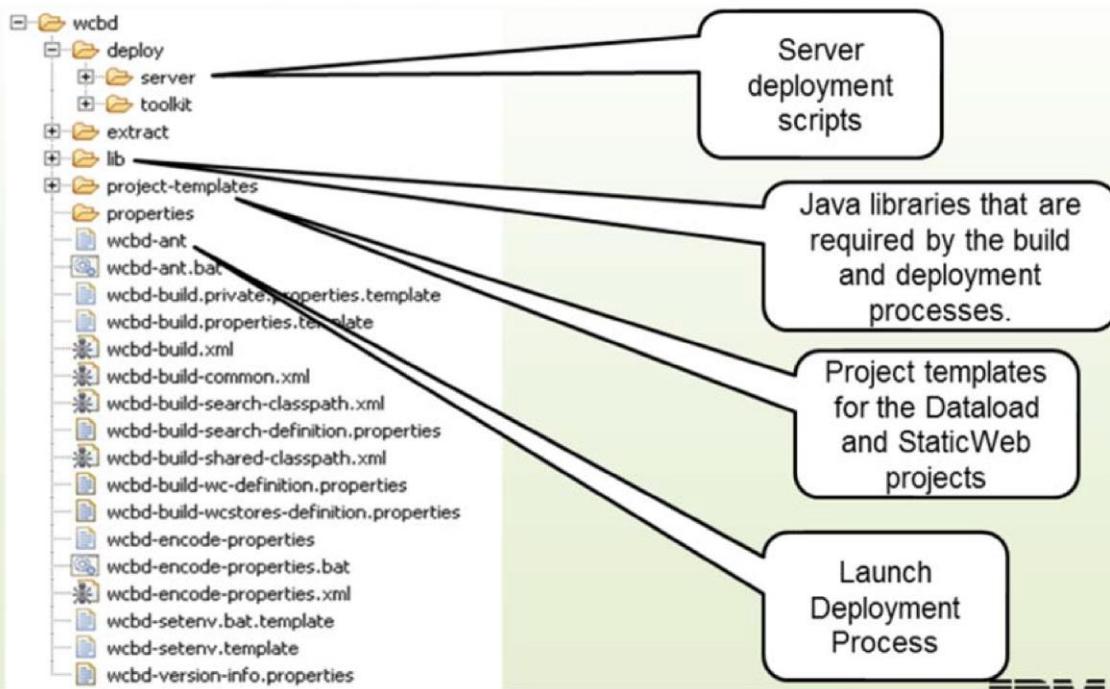


The diagram illustrates the system interactions and process flows of a typical setup of the Build and Deployment tool.

The diagram shows three main processes:

1. On the build server, the build process runs. It extracts source code from an SCM system and performs build-related tasks such as compilation and EJB deployment. It then creates two deployment packages (one for server, one for toolkit), which are used to deploy the customized assets.
2. On the server deployment system, the server deployment package is installed, configured, and run. It deploys customized assets to the database, the application server (specifically the WebSphere Commerce Enterprise Application), the web server, and the WebSphere Commerce server.
3. On the toolkit deployment system, which is the machine with WebSphere Commerce Developer Version 8.0 that is installed, the toolkit deployment package is installed, configured, and run. It deploys customized assets to the database and the toolkit workspace.

Location and structure of Build and Deployment Tool



© Copyright IBM Corporation 2016

As WCBD is an automated system, some planning must be involved which requires that the files that are involved, be placed in a structured set of directories.

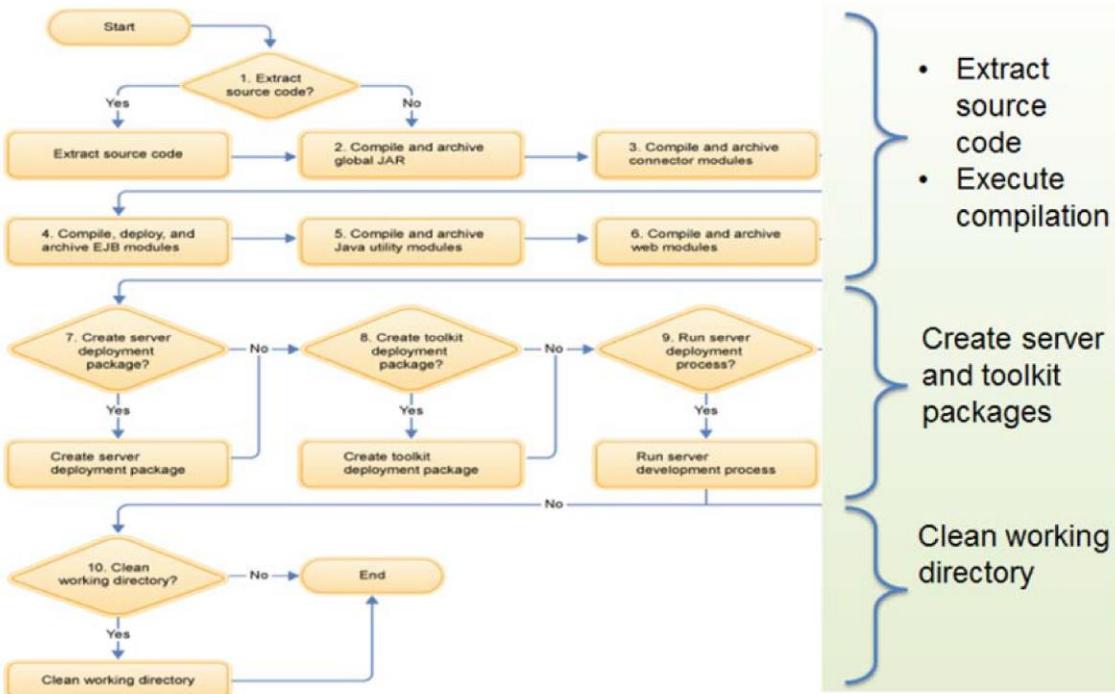
The Build and Deployment tool is installed as part of the WebSphere Commerce product installation to <WC_installdir>/wcbd. If you are installing WebSphere Commerce Developer, it is installed to <WCDE_installdir>/wcbd.

Configure and run the Build Process

© Copyright IBM Corporation 2016

This section describes an overview of Configure and run the Build Process.

High-level build process flow

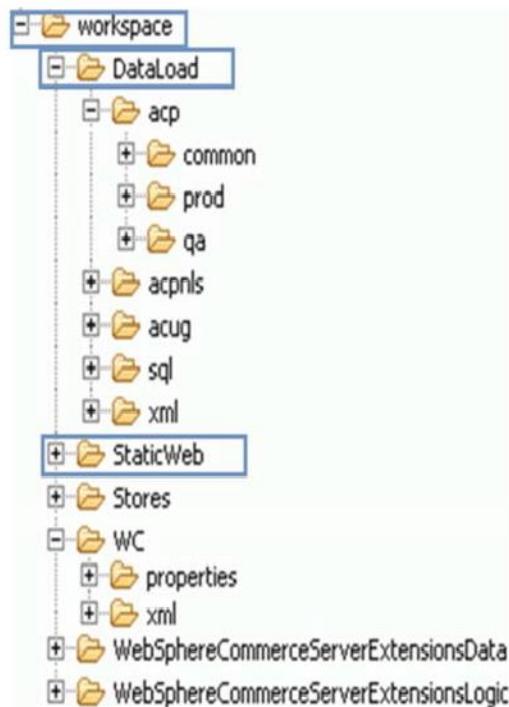


© Copyright IBM Corporation 2016

1. If set to run, extract source code with the user-provided Antscript.
2. Consolidates all Java source code from the source, compiles, and creates a global JAR that is used to compile each module individually. This is to help resolve classpath circular dependencies during the compilation phase using javac.
3. For each connector module set to build, compile and create an expanded resource archive (RAR).
4. For each Enterprise JavaBean (EJB) module set to build, compile, deploy, and create an EJB Java Archive (JAR) file.
5. For each Java utility module set to build, compile and create a JAR file.
6. For each Web module set to build, compile and create an expanded Web Archive (WAR).
7. If set to run, create a server deployment package by consolidating assets from source, archived modules and the WebSphereCommerce Build and Deployment tool.
8. If set to run, create a toolkit deployment package by consolidating assets from source, archived modules and the WebSphereCommerce Build and Deployment tool.
9. If set to run, unzip the server deployment package and run the server deployment process.
10. If set to run, clean the working directory.

WCBD tool repository structure

- The Build and Deployment tool is configured by default to use a specific reference repository structure.
- It is recommended to use the default repository structure to reduce the configuration effort of the build process.



© Copyright IBM Corporation 2016

Default repository structure in WCBD

Workspace : contains modules or projects that are part of the WebSphere Commerce Developer workspace.

Workspace/Dataload : Holds the data files for WCBD in supported file formats. The structure of Dataload project provides a way to separate different types of data files and different target environments.

workspace/StaticWeb : This project or module holds staticwebserver assets.

Setting up the source extraction Ant script

- The <WCBD_installdir>/extract directory includes sample scripts for some SCMs
- Copy <WCBD_installdir>/extract/wcbd-sample-extract-*.* files as <WCBD_installdir>/extract-*.*
- Open <WCBD_installdir>/extract-*.* files and edit them using a text editor based on the comments given in each file.

SCM	Sample script
CVS	wcbd-sample-extract-cvs.*
ClearCase	wcbd-sample-extract-clearcase.*
Local filesystem	wcbd-sample-extract-local.*
Subversion	wcbd-sample-extract-svn.*

© Copyright IBM Corporation 2016

Procedure

1. If WCBD_installdir/extract/wcbd-sample-extract-scm.private.properties exists, copy the file as WCBD_installdir/extract-scm.private.properties.
2. Copy WCBD_installdir/extract/wcbd-sample-extract-scm.properties as WCBD_installdir/extract-scm.properties.
3. Copy WCBD_installdir/extract/wcbd-sample-extract-scm.xml as WCBD_installdir/extract-scm.xml.
4. For AIX/Linux operating systems, change the file permission with the following command:
chmod 755 WCBD_installdir/extract-scm.*
5. Open WCBD_installdir/extract-scm.xml with a text editor and replace the name attribute of the root project element from wcbd-sample-extract-scm to extract-scm.

For example, if the SCM is CVS the WCBD_installdir/extract-cvs.xml should have the following line:

<project name="wcbd-sample-extract-cvs" default="all">

changed to:

<project name="extract-cvs" default="all">

6. If step 1 is applicable, open WCBD_installdir/extract-scm.private.properties with a text editor and configure the properties according to the comments in the file.
7. Open WCBD_installdir/extract-scm.properties with a text editor and configure the properties according to the comments in the file.

Build Configuration

- Process of configuring build settings explained in the next 2 slides
- Three main build setting files with their build configuration content.

File	Description
setenv.bat	Path information for Ant, Java, WAS, WC, and other components
build.private.properties	Contains information like passwords that are needed. The information is encoded during the build process.
build.properties	Contains the general settings that are used for the build.

© Copyright IBM Corporation 2016

setenv.bat:

The environment variables are set in this file that are called by the mass loading utilities and other WebSphere Commerce utilities.

This file contains the path information for Ant, Java, WAS, WC, and other components. For Example : open setenv.bat file to modify the ANT_HOME and WAS_HOME environmental properties to the correct values.

build.private.properties:

These properties provide security-sensitive settings, such as user names and passwords, to the build process. For security, the values of these properties are encoded at the beginning of the build process. They can also be encoded by the stand-alone properties file encoding utility manually. The original template file for these properties is WCBD_installdir/wcbd-build.private.properties.template.

build.properties:

These properties provide general settings to the build process. The original template file for these properties is WCBD_installdir/wcbd-build.properties.template.

Configuring the build settings - 2

- Copy the <WCBD_installdir>/wcbd-setenv.bat.template file as <WCBD_installdir>/setenv.bat or <WCBD_installdir>/setenv depending on the operating system.
- Copy the <WCBD_installdir>/wcbd-build.properties.template file as <WCBD_installdir>/build.properties
- Copy the <WCBD_installdir>/wcbd-build.properties.template file as <WCBD_installdir>/build.properties
- Open <WCBD_installdir>/setenv <WCBD_installdir>/setenv.bat with a text editor and set ANT_HOME to <Ant_installdir> and WAS_HOME to <WAS_installdir>

© Copyright IBM Corporation 2016

Configuring the build settings - 3

- Open <WCBD_installdir>/build.private.properties with a text editor and configure the properties in the file referring to the link in IBM Knowledge Center:
https://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rdewcbdbldpvtprop.htm?view=embed
- Open <WCBD_installdir>/build.properties with a text editor and configure the properties in the file referring to the link in IBM Knowledge Center:
https://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rdewcbdbldprop.htm?view=embed

© Copyright IBM Corporation 2016

Running the build process

- Ensure a meaningful build label such the build date or the tag name of the SCM given to each build
- Navigate to the <WCBD_installdir> directory
- Run the build process with the following command, where `build-label` is the build label to be used
 - On Windows, `wcbd-ant.bat -buildfile wcbd-build.xml -Dbuild.label=build-label`
 - On other operating systems, `./wcbd-ant -buildfile wcbd-build.xml -Dbuild.label=build-label`
- Upon completion, the server and toolkit deployment packages are created in the following location:
 - <WCBD_installdir>/dist/server/wcbd-deploy-server-build-label.zip
 - <WCBD_installdir>/dist/toolkit/wcbd-deploy-toolkit-build-label.zip
- These packages are now ready to be installed and configured for deployment

© Copyright IBM Corporation 2016

Advanced build features

The WebSphere Commerce Build and Deployment tool provides many advanced build features to support complex configurations.

- Support for multiple target environments
- Extracting source code by incremental update
- Excluding static Web assets in Web modules from the deployment packages
- Excluding EAR assets from the deployment packages
- Stand-alone properties file encoding utility
- Support for multiple build types
- Centralized server deployment
- Email notifications of build status
- Additional logging and tracing
- Debug options for EJBDeploy

© Copyright IBM Corporation 2016

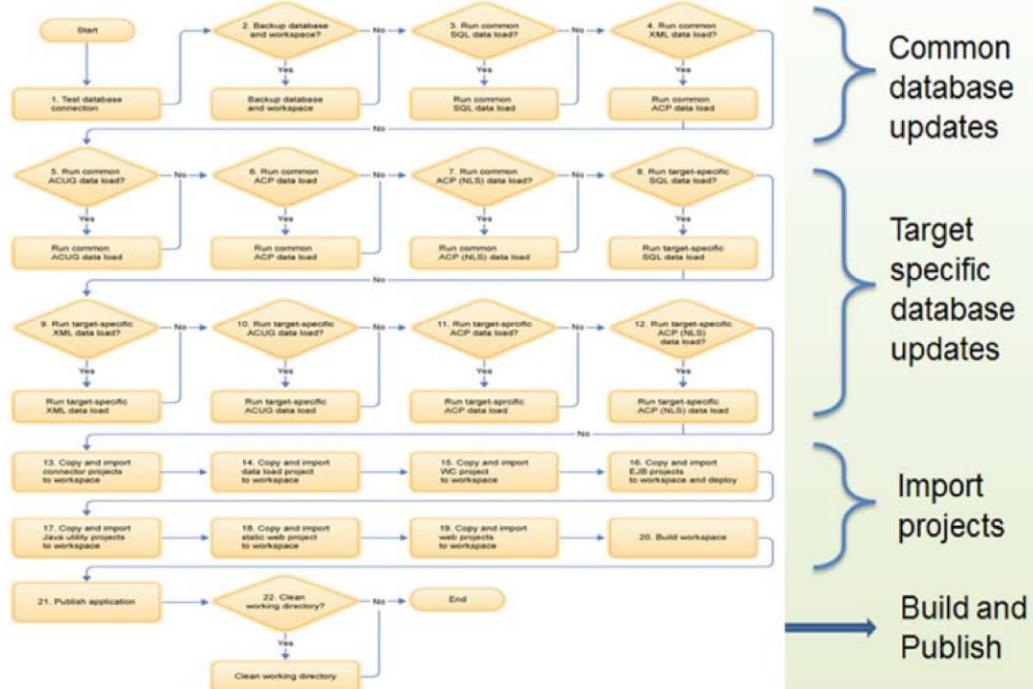
Refer to online IBM Knowledge Center for details of each of the advanced features listed –
https://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/cdewcbdadvbuild.htm?view=embed

Toolkit Deployment Process

© Copyright IBM Corporation 2016

This section describes an overview of the Toolkit Deployment Process.

Toolkit Deployment Process



© Copyright IBM Corporation 2016

1. Test the connection to the database to determine if any of the dataload steps are set to run.
2. If set to run, back up the Derby database and the workspace.
3. If set to run, perform common SQL dataload.
4. If set to run, perform common XML dataload. This result is identical to running the idresgen tool, then running the massloadtool.
5. If set to run, perform common ACUG dataload. This result is identical to running the acugload tool.
6. If set to run, perform common ACP dataload. This result is identical to running the acpload tool.
7. If set to run, perform common ACP (NLS) dataload. This result is identical to running the acpnlsload tool.
8. If set to run, perform target-specific SQL dataload.
9. If set to run, perform target-specific XML dataload. This result is identical to running the idresgen tool, then running the massloadtool.
10. If set to run, perform target-specific ACUG dataload. This result is identical to running the acugload tool.
11. If set to run, perform target-specific ACP dataload. This result is identical to running the acpload tool.
12. If set to run, perform target-specific ACP (NLS) dataload. This result is identical to running the acpnlsload tool.
13. For each connector project (module) set to include, copy to the workspace and import/refresh.
14. For the dataload project (module) set to include, copy to the workspace and

import/refresh.

15. For the WebSphere Commerce EAR and WebSphere Commerce search EAR projects (modules), copy assets to the workspace and import/refresh. EAR properties and XML files will be copied to the properties and xml directories in the WebSphere Commerce Developer installation directory.
16. For each EJB project (module) set to include, copy to the workspace, import/refresh, and deploy.
17. For each Java utility project (module) set to include, copy to the workspace and import/refresh.
18. For the static Web project (module), copy to the workspace and import/refresh.
19. For each Web project (module) set to include, copy to the workspace and import/refresh.
20. Compile the workspace.
21. Publish the Websphere Commerce application to the test server.
22. If set to run, clean the working directory.

Configuring and running toolkit deployment process – (1)

- Installing the toolkit deployment package

Procedure

- Create the WCBD_deploy_toolkit_common_dir directory
- Transfer the WCBD_installdir/dist/toolkit/wcbd-deploy-toolkit-build-label.zip file from the build system to the toolkit deployment system and decompress it to WCBD_deploy_toolkit_common_dir.

- Running the toolkit deployment process

Procedure

- Change directory to WCBD_deploy_toolkit_dir
- Run the toolkit deployment process with the following command, where target-env is the identifier for the target environment
`wcbd-rad-ant.bat -buildfile wcbd-deploy.xml -Dttarget.env=target-env`

© Copyright IBM Corporation 2016

Installing the toolkit deployment package:

The directory WCBD_deploy_toolkit_common_dir/wcbd-deploy-toolkit-build-label is created and it is referred to as WCBD_deploy_toolkit_dir.

Tip: If the deployment package directory name is long, you might encounter problems when you extract the package due to path size limitations on the operating system.

Running the toolkit deployment process: Before you begin

The WebSphere Commerce Test Server must be stopped and WebSphere Commerce Developer must be closed before the toolkit deployment process can be run.

Before running the toolkit deployment process, it is highly recommended that the following recovery measures be taken: Back up the database.

Refer to the documentation of your database management system (DBMS) for details.

Note: Backup for Apache Derby database is automated in the toolkit deployment process.

Back up your workspace. The WebSphere Commerce Build and Deployment tool provides the functionality to back up such assets if it is enabled. Refer to the toolkit deployment properties files for details.

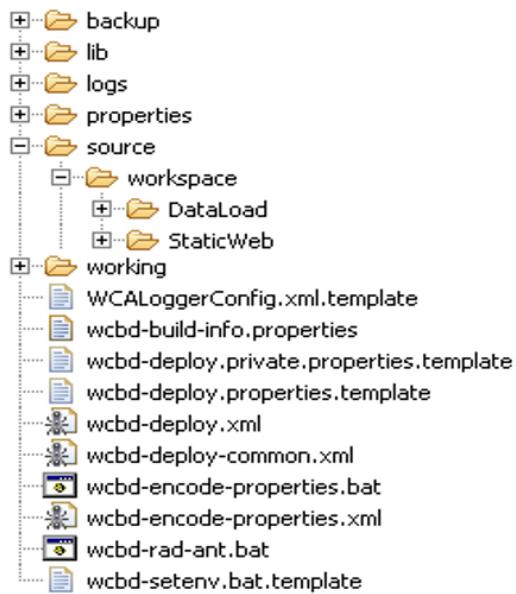
Important: Not taking recovery measures will result in not being able to recover if a deployment failure occurs.

In addition to providing information to the console, the build process generates a log in the following directory:

WCBD_deploy_toolkit_dir/logs/build-label-target-env-tstamp/wcbd-deploy-toolkit.log

Configuring and running toolkit deployment process – (2)

- Toolkit deployment directory file structure



© Copyright IBM Corporation 2016

Configuring and running toolkit deployment process – (3)

- Configuring the toolkit deployment settings

Procedure

- Copy the WCBD_deploy_toolkit_dir/wcbd-setenv.bat.template file as WCBD_deploy_toolkit_dir/setenv.bat
- Copy the WCBD_deploy_toolkit_dir/wcbd-deploy.private.properties.template file as WCBD_deploy_toolkit_dir/deploy-target-env.private.properties, where target-env is an identifier for the target environment
- Copy the WCBD_deploy_toolkit_dir/wcbd-deploy.properties.template as WCBD_deploy_toolkit_dir/deploy-target-env.properties
- Open WCBD_deploy_toolkit_dir/setenv.bat with a text editor and set WCDE_HOME to WCDE_installdir
- Open WCBD_deploy_toolkit_dir/deploy-target-env.private.properties with a text editor and configure the properties in the file by referring to the Security-sensitive toolkit deployment configuration properties topic
- Open WCBD_deploy_toolkit_dir/deploy-target-env.properties with a text editor and configure the properties in the file by referring to the Toolkit deployment configuration properties topic

© Copyright IBM Corporation 2016

Security-sensitive toolkit deployment configuration properties topic can be referred here :

http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rdewcbdtktddeploypropvprop.htm

Toolkit deployment configuration properties topic can be referred here :

http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rdewcbdtktddeployprop.htm

Tip: To save time when configuring a new toolkit deployment package, you can provide the fully or partially pre-configured, target-environment-specific properties files and include them in the WCBD_installdir/deploy/toolkit directory on the build system. Such assets are included in the new toolkit deployment package, reducing the amount of configuration required

Configuring and running toolkit deployment process – (4)

- The Build and Deployment tool provides advanced toolkit deployment features to support more complex configurations
 - Loading procedural SQL files
 - Loading data by using the Data Load utility
 - Additional logging and tracing
Ant provides both a verbose and a debug option to provide additional tracing that might be useful for problem determination. To enable verbose mode, provide either the -v or -verbose option when running the wcbd-rad-ant.bat command

© Copyright IBM Corporation 2016

For more details refer to knowledge center here :

http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/cdewcbdadvt toolkit.htm

Configuring and running the toolkit delta deployment process – (1)

- Configuring and running the toolkit delta deployment process
- Running the toolkit delta preparation process

Procedure

- Open a command prompt and change to the WCBD_target_deploy_toolkit_dir directory.
- Run the following command:
`wcbd-rad-ant.bat -buildfile wcbd-delta-prepare.xml -Dtarget.env=target-env -Dbaseline.dir=WCBD_baseline_toolkit_deploy_dir`

- Running the toolkit deployment process in delta mode
 - Provide the following additional argument to the wcbd-rad-ant command and follow the same process as specified in the toolkit deployment process
`-Ddelta.mode=true`

© Copyright IBM Corporation 2016

Running the toolkit delta preparation process:

The delta preparation process compares the baseline and the target based on the deployment settings in the target to identify changes (also known as delta) between them and configures the delta assets and deployment settings as appropriate.

Before you begin

The target environment must be at the same code level as the baseline, otherwise delta deployment might introduce inconsistencies.

Both the baseline and the target deployment packages must be installed and available.

The deployment settings must have been configured for a full deployment in the target, so that the delta preparation process can determine the assets that need to be compared. Refer to the Configuring the toolkit deployment settings topic for detail.

Note: In order for the delta deployment process to delete modules that have been deleted from the target, you must add the deleted modules to the appropriate module list properties in `WCBD_target_deploy_toolkit_dir/deploy-target-env.properties`.

The following files and directories are created in `WCBD_target_deploy_toolkit_dir` when the delta preparation process completes:

- The new deployment configuration properties file, `delta-deploy-target-env.properties`, that is configured for and will be used for delta deployment
- The new source directory, `delta-source`, that contains only the changes between the baseline and the target, which will be used for the actual delta deployment
- The file `delta-summary-target-env.xml`, that contains a summary of the comparison between the baseline and the target, with all files and their delta type listed

- The log file, logs/delta-target-env/wcbd-delta-prepare.log, of the delta preparation process
- **Running the toolkit deployment process in delta mode:**
- After the toolkit delta preparation process is run to on the target deployment directory, the toolkit deployment process can then be run in delta mode to perform the actual delta deployment.
- To run the toolkit deployment process in delta mode, you simply provide an additional Ant property, delta.mode, to specify as so in command line. The delta toolkit deployment configuration properties file, *WCBD_target_deploy_toolkit_dir/deploy-target-env.properties*, will be used for the deployment instead.

Configuring and running the toolkit delta deployment process – (2)

- Running the overall delta deployment process in one step

Procedure

- If it does not exist, create and configure the `baseline.properties` file.
- Open a command prompt and change to the `WCBD_target_deploy_toolkit_dir` directory
- Run the following command:

```
wcbd-rad-ant.bat -buildfile wcbd-delta-deploy.xml  
-Dtarget.env=target-env
```

© Copyright IBM Corporation 2016

The overall delta deployment log file `WCBD_target_deploy_toolkit_dir/logs/delta-target-env/wcbd-delta-deploy.log` is created. Upon a successful delta deployment, the `baseline.dir` property in `baseline.properties` will also be set to `WCBD_target_deploy_toolkit_dir`. Subsequent delta deployment can then use the current target as the next baseline.

Configuring and running the toolkit delta deployment process – (3)

- Toolkit deployment configuration properties

Snapshot of some of the properties are shown in the table:

Property	Summary
<code>source.dir</code>	The deployment source directory.
<code>module.dir</code>	The directory in \${source.dir} where the modules to be deployed are located.
<code>log.dir</code>	The deployment log directory.
<code>log.file</code>	The deployment log file.
<code>[inProp] working.dir</code>	The working directory used to store temporary files that are generated by the deployment process.
<code>backup.dir</code>	The directory where the Derby database and the workspace are backed up.
<code>run.backup</code>	Whether the Derby database (if it is the current database type) and the workspace should be backed up before the code is deployed.
<code>connector.module.list</code>	The comma-separated list of connector modules to be deployed.
<code>ejb.module.list</code>	The comma-separated list of EJB modules to be deployed.
<code>java.module.list</code>	The comma-separated list of Java utility modules to be deployed.
<code>web.module.list</code>	The comma-separated list of Web modules to be deployed.
<code>openlaszlo.web.module.list</code>	The comma-separated list of OpenLaszlo Web modules to be deployed.
<code>data.module.name</code>	The name of the module that contains data assets.
<code>static.web.module.name</code>	The name of the module that contains static Web server assets.
<code>db.name</code>	The database name.
<code>db.schema.name</code>	The database schema name.
<code>jdbc.url</code>	The JDBC URL.
<code>jdbc.driver</code>	The JDBC driver class name.
<code>jdbc.driver.path</code>	The classpath that contains the JDBC driver library and other libraries it depends on.
<code>datasource.indy.name</code>	The JNDI name of the default datasource that will be mapped for entity beans in EJB modules that are deployed.
<code>sql.onerror</code>	The action to perform when a statement fails as part of the <sql> task.
<code>[inProp] procedural.sql.file.name.id</code>	The string identifier in the name of a SQL file which indicates that the file is a procedural SQL file.
<code>[inProp] procedural.sql.delimiter</code>	The delimiter that separates procedural SQL statements in a procedural SQL file.
<code>wcs.dtd.path</code>	The path to the wcs.dtd file that is used for loading ACP and ACUG data.
<code>wca.logger.config.file</code>	The path to the WCALogger config file which is used by the WebSphere Commerce loading utilities.
<code>wca.logger.output.dir</code>	The directory in which logs will be created by the WebSphere Commerce loading utilities.

© Copyright IBM Corporation 2016

- These properties provide general settings to the toolkit deployment process. The original template file for these properties is in WCBD_installdir/deploy/toolkit/wcbd-deploy.properties.template
- Refer to the knowledge center link for complete details on these properties files - http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rdewcbdtktddeployprop.htm

Configuring and running the toolkit delta deployment process – (4)

- Security-sensitive toolkit deployment configuration properties

Property	Summary
db.user.name	The database user name that is used by the sql Ant task and the WebSphere Commerce loading utilities to connect to the database.
db.user.password	The database user password that is used by the sql Ant task and the WebSphere Commerce loading utilities to connect to the database.

© Copyright IBM Corporation 2016

- These properties provide security-sensitive settings, such as user names and passwords, to the toolkit deployment process.

Customizing the toolkit deployment process

© Copyright IBM Corporation 2016

This section describes an overview of Customizing the toolkit deployment process.

Customizing the toolkit deployment process – (1)

Procedure

If new functions are required in the Ant build

- Create a helper Ant build file that is called `WCBD_deploy_toolkit_dir/project-deploy-common.xml`
- Open the file with a text editor and add the following line after the root project element open tag:
`<import file="${basedir}/wcbd-deploy-common.xml" />`
- Add new targets to the file as needed. The existing helper target and any targets that might depend on it can be overridden using the specification of the Ant import task

If new security-sensitive properties are required for the new changes

- Copy the `wcbd-deploy.private.properties.template` file as
`WCBD_deploy_toolkit_dir/project-deploy.private.properties.template`
- Add the new properties to the end of the file

© Copyright IBM Corporation 2016

Customizing the toolkit deployment process – (2)

If new properties are required for the new changes

- Copy WCBD_deploy_toolkit_dir/wcbd-deploy.properties.template file as WCBD_deploy_toolkit_dir/project-deploy.properties.template
- Add the new properties to the end of the file

If new libraries are required by Ant for the new changes

- Copy WCBD_deploy_toolkit_dir/wcbd-setenv.bat.template file as WCBD_deploy_toolkit_dir/project-setenv.bat.template
- Update the CLASSPATH variable in WCBD_deploy_toolkit_dir/project-setenv.bat.template

To include the new or modified functions in the overall toolkit deployment process

- Copy WCBD_deploy_toolkit_dir/wcbd-deploy.xml as WCBD_deploy_toolkit_dir/project-deploy.xml
- If the new helper Ant build file project-deploy-common.xml was created, change the following line in WCBD_deploy_toolkit_dir/project-deploy.xml:
`<import file="${basedir}/wcbd-deploy-common.xml" />`
to:
`<import file="${basedir}/project-deploy-common.xml" />`
- Modify the logic of the toolkit deployment process to incorporate the new changes

© Copyright IBM Corporation 2016

Custom Asset Deployment Steps

- Customized assets for WebSphere Commerce are deployed in two stages
 - Packaging
 - The first stage is twofold: compiling and then packaging
 - Deployment
 - In this stage, the deployment packages are deployed to the target server environment.
- Deployment changes have been made in the following areas
 - JEE packaging
 - Configuration files
 - Application updates

© Copyright IBM Corporation 2016

Exercise 6: WebSphere Commerce Build and Deployment

© Copyright IBM Corporation 2016

Unit summary

This unit was designed to enable you to:

- An overview of the WebSphere Commerce Build and Deployment Tool
- Understand how to configure and run the Build process
- Understand the Toolkit Deployment process
- Steps for customizing the toolkit deployment process

© Copyright IBM Corporation 2016

Additional training

For additional training needs, you have several options available for registering.

You can register	At
On site	This or any of our training facilities
Online	http://ibm.com/training/
Email	ibmswedu@us.ibm.com
By telephone	1-800-IBM Teach (1-800-426-8322)
By fax	208-692-6130

© Copyright IBM Corporation 2016

For additional training needs, you have several options available for registering like Onsite, Online , Email , by telephone or by Fax .