



IBM Software Group

## WebSphere® Commerce Feature Pack 2

*WebSphere Commerce Portal Integration*

*Portlet Programming Model*



@business on demand.

© 2007 IBM Corporation  
Updated May 15, 2007

Welcome to the WebSphere Commerce Feature Pack 2 Portal Integration portlet programming model presentation.

## Agenda

- Architecture overview
- Programming model
  - ▶ MVC style
- Customization

This presentation discusses the architecture overview, the MVC programming model and customization details.

## WebSphere Commerce Portal Integration enhancements

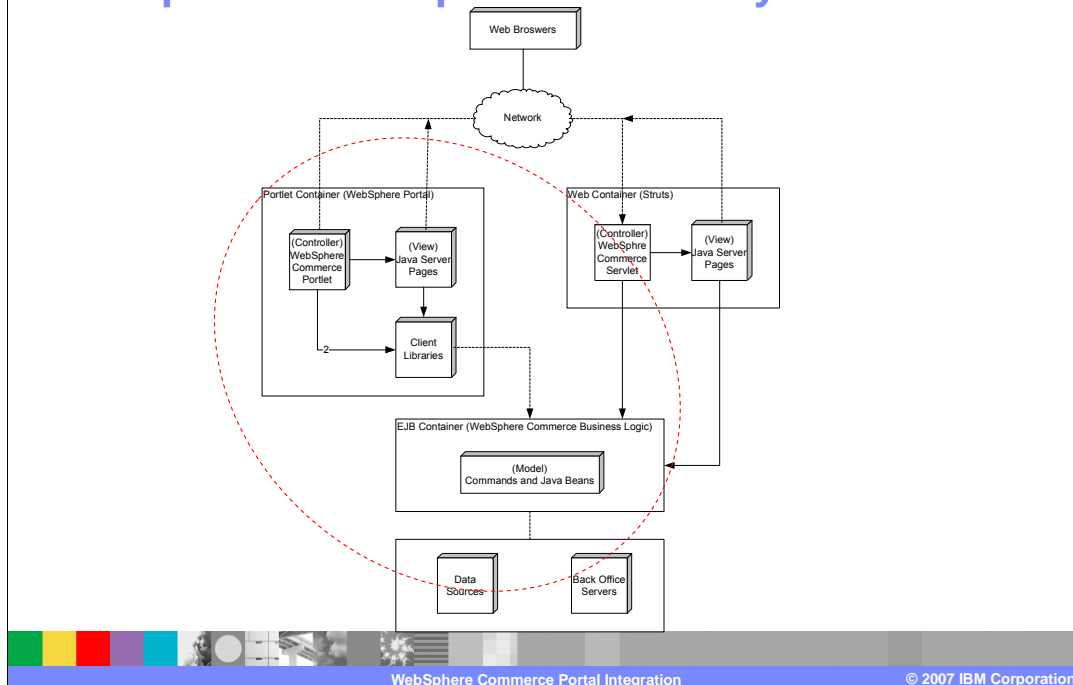
- Feature enhancements
  - ▶ Deploy a standard SOA based solution
  - ▶ JSR-168 compatibility
  - ▶ Support for portlet-to-portlet communication
  - ▶ Leverage the portlet tag library
  - ▶ Allow easier customization
- Platform support for WebSphere Commerce V6
  - ▶ WebSphere Portal V6
  - ▶ Rational® Application Developer V7

As part of the integration roadmap between WebSphere Commerce and WebSphere Portal, WebSphere Commerce has released an enhancement to WebSphere Commerce V6. The key change in this major revision is to align with the IBM® SOA programming model by having portlets as the presentation layer, to WebSphere Commerce business functionality. The goal is to enable customers to leverage the capabilities provided by WebSphere Portal Server, while accessing information such as products and orders in WebSphere Commerce, through Web services. Other key enhancements for WebSphere Commerce Portal integration include:

1. JSR-168 compatibility which enables you to build portlets that are compliant to the JSR-168 standard,
2. Portlet-to-portlet communication where portlets written to access WebSphere Commerce functionality can use portlet-to-portlet communication provided by such portlet containers as WebSphere Portal,
3. A portlet tag library which allows your portlets to support linking, and
4. WebSphere Commerce functionality exposed as services invoked by portlets.

Platforms supported by WebSphere Commerce Portal solution are WebSphere Portal V6 with Rational Application Developer V7 development environment and WebSphere Commerce V6 with Rational Application Developer V6 development environment. Additionally, LDAP configuration is provided with this platform support.

## WebSphere Portal presentation layer



WebSphere Commerce supports multiple presentation layers; you can use an appropriate presentation layer based on your business requirements. For example, if your business processes are represented by WebSphere Commerce business logic and data, then Struts is an appropriate choice for your presentation layer. However, if you want to aggregate business processes from WebSphere Commerce with those from other non WebSphere Commerce applications, then it is appropriate to use WebSphere Portal for the presentation layer.

Both the Struts and WebSphere Portal presentation layers follow the model view controller (MVC) design pattern. The consequent separation of the presentation logic and the business logic enables a Web designer to develop the presentation layer at the same time as an application developer implements business logic.

In the Struts framework, a browser request is routed to a servlet that acts as a controller. Using local Java™ calls, the controller calls the model for processing. The controller then dispatches the appropriate view to render data. The model encapsulates all business logic implemented using the command pattern. The JSP™ pages retrieve data from the database using data beans, and then format the output.

In the WebSphere Portal framework, the browser request is routed to a portlet that acts as a controller. The portlet calls client libraries, then the client library sends a service request to WebSphere Commerce business logic for processing. When the portlet renders the data, it dispatches to a JSP page in the portlet container. The JSP pages use tags that delegate to the client libraries to retrieve data from the WebSphere Commerce system.

## WebSphere Portal client architecture

- Architecture follows MVC pattern:
  - ▶ *processAction* calls WebSphere Commerce (Update) commands through Web services
    - Hides internals with façade
  - ▶ *doView* makes calls to the data source to (Read) retrieve content
    - Renders JSPs in the WebSphere Portal Server not the WebSphere Commerce Server
- Single portlet can render various displays:
  - ▶ Calls different tasks on a business façade

The key change in the architecture is to render content in the WebSphere Portal server instead of on the WebSphere Commerce server. Another goal is that portlets have to be written according to the portlet development guide without any need for a special WebSphere Commerce-Portal framework. There should be no WebSphere Commerce-Portlet base class which developers must extend. The goal is to treat WebSphere Commerce functionality as services with portlets being the window into those services.

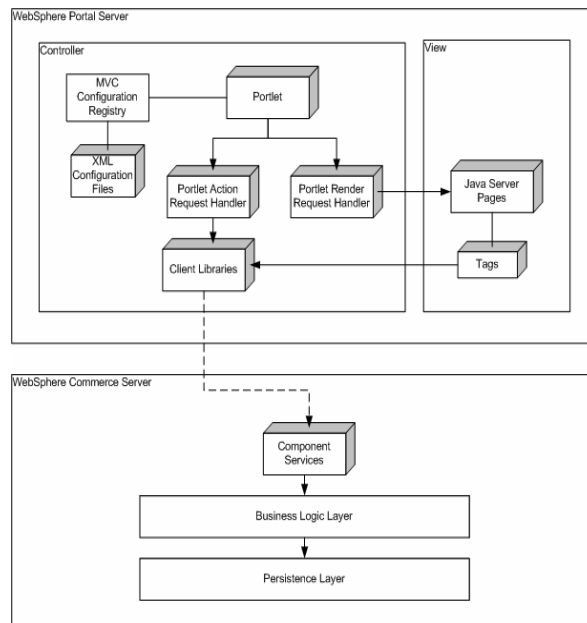
Requests are served by the WebSphere Portal Server which uses a Web services style interface to perform create, read, update and delete operations against the WebSphere Commerce system. Contextual information is managed in the Portal server to maintain information such as language and currency consistently across request in a session.

The update calls from WebSphere Portal to the WebSphere Commerce server need to be fronted using a façade to hide the communication mechanism. These update calls only occur in the *processAction* method of the portlet. Helper APIs need to be provided to aid in invoking services on the WebSphere Commerce server. These helper APIs should include libraries to help in managing contextual information and accessing business services such as catalog and order. To display content, the *doView* calls dispatch to a JSP which retrieve WebSphere Commerce content using a tag library.

A portlet can call many different tasks. Each action invokes a business task on a business façade.

## MVC portlet components

- MVC portlet
- Configuration registry
- Event handler
- Portlet JSP
- getData tag
- Client library



WebSphere Commerce Portal Integration

© 2007 IBM Corporation

WebSphere Commerce has provided an optional integration framework intended to reduce development effort when calling WebSphere Commerce services from your portlet. You do not need to use the framework to make use of the client libraries and the WebSphere Commerce foundation tag library in your portlet. The diagram shown depicts the interactions between the key pieces of WebSphere Commerce Server and WebSphere Portal Server. This slide describes the first two pieces of the framework.

### Portlet

A portlet is similar to a servlet, except that it has parts that contribute to the MVC design pattern. MVCPortlet is a generic implementation of the MVC pattern that allows portal administrators to set up one or more portlets, each with its own configuration, to call various WebSphere Commerce business services.

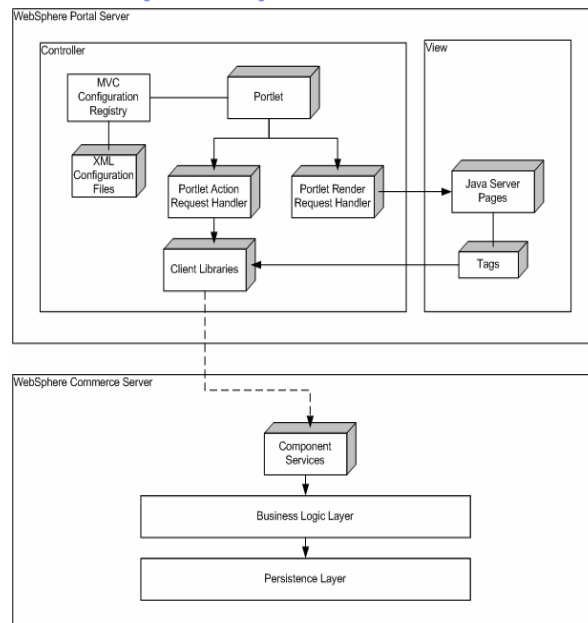
Portlets have multiple states, view modes, and event and messaging capabilities. Portlets run inside the portlet container of the WebSphere Portal Server, similar to the way a servlet runs on an application server. The portlet container provides a runtime environment where portlets are instantiated, used, and then destroyed. Portlets rely on the WebSphere Portal infrastructure to access user profile information, participate in window and action events, communicate with other portlets, access remote content, look up credentials, and store persistent data.

### Configuration registry

The portlet configuration registry is a cached version of configurations related to all portlet operations, such as portlet actions and portlet renders. These static configurations are defined in XML files that are packaged as part of the WebSphere Commerce sample portlet application. You can find these files on the file system after you install the portlet application to the WebSphere Portal Server.

## MVC portlet components (cont.)

- MVC portlet
- Configuration registry
- Event handler
  - ▶ Action
  - ▶ Render
- Portlet JSP
- getData tag
- Client library



This slide describes the event handler piece of the framework.

Portlets and servlets have different processing and rendering sequences. A servlet does all of its processing in the service() method. A portlet, on the other hand, uses two-phase processing that is split between an action phase and a render phase. This split is necessary to accommodate communication between portlets before rendering output in the service stage. The action phase is guaranteed to be completed before a portlet is called to render. The WebSphere Commerce and WebSphere Portal integration framework is designed in a flexible way so that each portlet request can be handled differently using an event handler. For each action event and render event, you can define a corresponding event handler in the portlet MVC configuration file. Although the render phase is always called, the action phase may not be called in certain circumstances.

### Portlet Action Handler

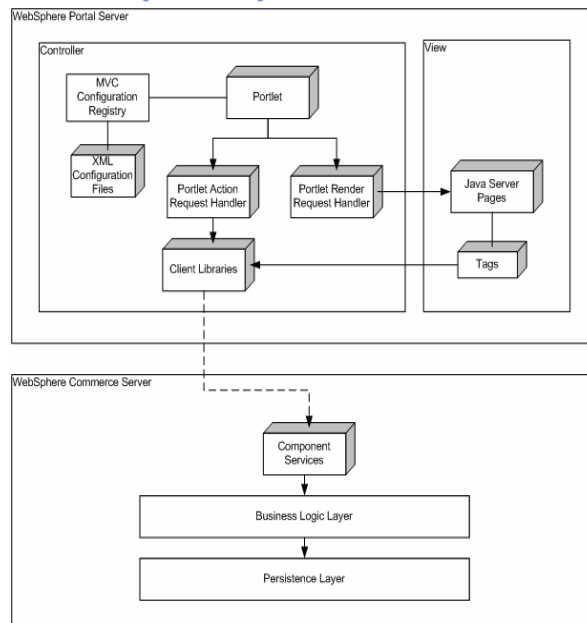
This handler, `MVCPortletActionHandler`, retrieves two properties, `clientLibrary` and `clientMethod`, and then calls the method, passing a parameter map that is composed of name-value pairs given in the URL. When the client library returns a response, this handler takes the response name-value pair that was returned from the client library and adds it to the URL. These are added as render parameters so that the context and core business data are not lost in an event of a page refresh. A page refresh issued from the portlet container does not involve `processAction()`, so certain business data might be lost as a result of a refresh request.

### Portlet Render Handler

This handler, `MVCPortletRenderHandler`, does not call WebSphere Commerce services. Instead, it checks for the current state and mode of the portlet to determine the appropriate JSP file path to be used when calling the request dispatcher for rendering.

## MVC portlet components (cont.)

- MVC portlet
- Configuration registry
- Event handler
- Portlet JSP
- getData tag
- Client library



This slide describes the last three pieces of the framework.

### Portlet JSP file

The Portlet JSP file is a view template for displaying business data and getting inputs from the portal user. Special UI tags, such as portlet tags and WebSphere Commerce foundation tags, are used on these pages to avoid inline JSP Java coding. Portlet tags are provided by WebSphere Portal to access information specific to the portal environment, whereas WebSphere Commerce foundation tag library tags are for retrieving populated service data objects from WebSphere Commerce services.

### Client library

A client library is a WebSphere Commerce Component Service interface for client side invocation. Client libraries should not be aware of any artifacts specific to WebSphere Portal, such as Portlet Session and Credential Service Vault. Upon each invocation, business objects, such as BusinessContextType and AuthenticationCallbackHandler, are passed to the interface so that it can hand over specific information to the service binding layer.

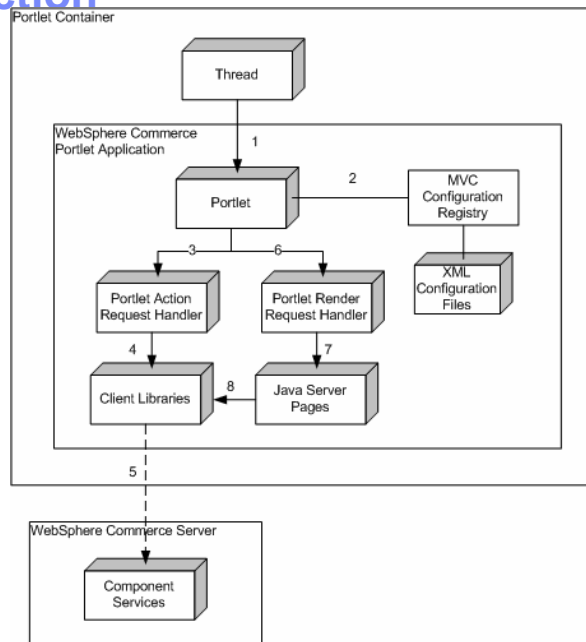
### WebSphere Commerce foundation tag library

A `getData` action tag is provided as part of the WebSphere Commerce UI runtime. It retrieves populated service data objects from WebSphere Commerce services.



## Framework interaction

- Action (Update)
  - ▶ 1,2,3,4,5
- Render (Read)
  - ▶ 1,2,6,7,8,5



Using the WebSphere Commerce Portlet provided, a portlet developer can create or remove actions and renders through configurations. The following description summarizes the interactions:

1. The request from the WebSphere Portal user is directed to the presentation layer in the WebSphere Portal Server's portlet container.
2. The portlet container then calls the processAction() method in the WebSphere Commerce Portlet where the definition of the requested action is looked up from the MVC configuration stored in the Configuration Registry.
3. The designated PortletActionHandler for this portlet action retrieves the client library interface definition and the required parameters from the Configuration Registry.
4. The PortletActionHandler gathers the parameter map from the request, along with the required business context and authentication callback handler (which can be acquired through the credential plug-in), and passes them to the pre-configured client library method defined in the Configuration Registry.
5. The client library retrieves the required request parameters from the parameter map and converts them into a message which is then forwarded to a WebSphere Commerce service on WebSphere Commerce Server. The designated WebSphere Commerce component performs the requested business operation. After the operation is completed, a response is returned to the client library. The PortletActionHandler analyzes the response and generates the required render parameters for the upcoming render request. In the event of an exception or service fault received from the client library, the PortletActionHandler displays the error view.
6. The Portlet Container generates another render request and calls the render method in the WebSphere Commerce Portlet for rendering a page back to the WebSphere Portal user. The Portlet Container calls the MVCPortlet's render method where the definition of the requested render is looked up from the MVC configuration stored in the Configuration Registry.
7. The designated PortletRenderHandler retrieves the JSP file path from the Configuration Registry and assigns the portlet JSP page to the request dispatcher for rendering.
8. The JSP file retrieves data from the WebSphere Commerce tag library.
9. The WebSphere Commerce tag library calls the client library.

## Customization and extensibility

- MVC style – using the provided WebSphere Commerce MVCPortlet class
  - ▶ Uses generic implementation of the MVC pattern
  - ▶ Allows portlet designers to set up portlets to call WebSphere Commerce services
  - ▶ Reduces code redundancy
  - ▶ Maintains consistent behavior across all WebSphere Commerce portlet actions
  
- Web service style – using WebSphere Portal general programming techniques
  - ▶ Invokes back-end WebSphere Commerce Web services using client library
  - ▶ Requires portlet developer to write more code and configuration for any customization logic

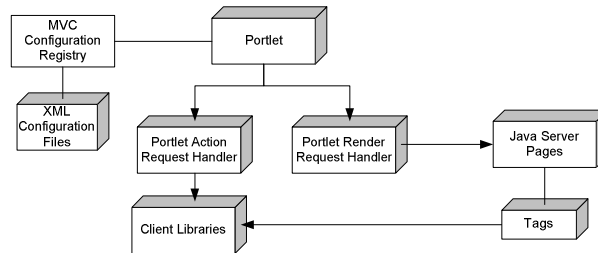
There are two main approaches to Customizing WebSphere Commerce Portal integration:

1. The MVC style, which uses the provided WebSphere Commerce MVCPortlet class or
2. The Web service style, which uses WebSphere Portal general programming techniques

For the MVC style, the MVCPortlet, is a generic implementation of the MVC pattern. It allows you to set up one or more portlets, each with its own configuration, to call various WebSphere Commerce services. This programming pattern greatly reduces code redundancy and maintains a consistent behavior across all WebSphere Commerce portlet actions.

For the Web service style, a portlet can invoke the backend Web services provided by WebSphere Commerce or a vendor directly without the need for a client library. Although this model is less restrictive than the previous one, you have to write more code and configuration for any customization logic.

## MVC style programming pattern



- MVC portlet assets:
  - ▶ MVC configuration files – portlet-config.xml
  - ▶ Event handlers – action handler and render handler
  - ▶ Portlet JSPs
- WebSphere Commerce services assets:
  - ▶ getData tag – get-data-config.xml
  - ▶ Client libraries

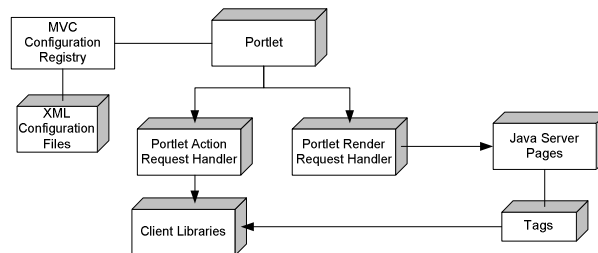
With the MVC style, there are several customizable assets, described in the next two slides:

**Portlet Action and Render Configuration** - Each portlet can have one or more MVC configuration files and they are loaded in the order that is specified in the portlet deployment descriptor. In each MVC configuration file, an action definition is required to define how the request can be mapped into a WebSphere Commerce service request. A render definition might also be required, depending on the nature of the request.

**Event Handlers** - Although the MVC configuration allows you to define any customized event handler, the use of two generic event handlers is recommended: 1) a generic portlet action handler, MVCPortletActionHandler, which calls client libraries through configurations specified in the MVC configuration file and 2) a generic portlet render handler, MVCPortletRenderHandler, which returns the proper portlet JSP page according to the current portlet mode.

**Portlet JSP file** - The Portlet JSP file is a view template for displaying business data and getting inputs from the portal user. These JSP files contain markup tags that provide a consistent, clean, and complete user interface. The portal page is displayed using skins and themes defined by the portal designer or administrator. For portlets to appear integrated with an organization's portal or user's customized portal, they should generate markup that invokes the generic style classes for portlets, rather than using tags or attributes to specify colors, fonts, or other visual elements. Portlets are allowed to render only markup fragments, which are then assembled by the portlet framework for a complete page. Portlet output should contain complete, well-structured, and valid markup fragments. This helps to prevent the portlet's HTML code from corrupting the portal's aggregated HTML code.

## MVC style programming pattern (cont.)



- MVC portlet assets:
  - ▶ MVC configuration files – portlet-config.xml
  - ▶ Event handlers – action handler and render handler
  - ▶ Portlet JSPs
- WebSphere Commerce services assets:
  - ▶ getData tag – get-data-config.xml
  - ▶ Client libraries

**WebSphere Commerce foundation tag library** - A `<wcf:getData>` action tag is provided as part of the WebSphere Commerce foundation tag library. This tag retrieves populated service data objects from the WebSphere Commerce services.

**WebSphere Commerce services** - WebSphere Commerce provides services that have been created using existing WebSphere Commerce controller commands and data beans, fronted by a component facade interface. These services can be used on the portlet using the WebSphere Commerce foundation tag library.

**Client Library** - Each WebSphere Commerce service module provides a client library that is responsible for building the messages to be sent to the WebSphere Commerce services. Upon each invocation, business objects, such as `BusinessContextType` and `AuthenticatoinCallbackHandler`, are passed to the interface so that it can hand over specific information to the service binding layer.

## Action example – portlet-config.xml

```
<action
  className="com.ibm.commerce.foundation.client.portal.handlers.MVCPortletActionHandler"
  name="ChangePersonalInformation">
  <set-property name="clientLibrary"
    value="com.ibm.commerce.member.facade.client.MemberFacadeClient" />
  <set-property name="clientMethod" value="updatePerson" />
  <set-property name="renderName"
    value="ChangePersonalInformationDisplay" />
</action>
```

- **ChangePersonalInformation action:**

- ▶ Uses generic action handler:  
MVCPortletActionHandler
- ▶ Calls client library:  
MemberFacadeClient.updatePerson()

This is an example of the Change Personal Information action.

## Action example – Portlet JSP

```
<portlet:actionURL var="ChangePersonalInformationActionURL" secure="true">
  <portlet:param name="actionName" value="ChangePersonalInformation" />
  <portlet:param name="renderName" value="AccountDisplay" />
  <portlet:param name="faultRenderName"
    value="ChangePersonalInformationErrorDisplay" />
</portlet:actionURL>

<form name="<portlet:namespace/>ChangePersonalInformation" method="post"
  action="<c:out value="\${ChangePersonalInformationActionURL}" escapeXml="false" />">

  <td><input type="text" maxlength="40" size="35" id="firstName"
    name="firstName" value="<c:out value="\${contactInfo.contactName.firstName}" />" /></td>
```

This is an example of the Change Personal Information action URL on the portlet JSP.

## Render example – portlet-config.xml

```
<render
  className="com.ibm.commerce.foundation.client.portal.handlers.MVCPortletRenderHandler"
  name="ChangePersonalInformationDisplay">
  <forward mode="view"
    path="/member/ChangePersonalInformationDisplay.jsp" />
</render>
<render
  className="com.ibm.commerce.foundation.client.portal.handlers.MVCPortletRenderHandler"
  name="ChangePersonalInformationErrorDisplay">
  <forward mode="view"
    path="/member/ChangePersonalInformationErrorDisplay.jsp" />
</render>
```

- **ChangePersonalInformation render:**
  - ▶ Uses generic render handler:  
MVCPortletRenderHandler
  - ▶ Forwards to portlet JSP:  
/member/ChangePersonalInformationDisplay.jsp

This is an example of the Change Personal Information render.

## Render example – Portlet JSP

```

<%@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://java.sun.com/portlet" prefix="portlet"%>
<%@taglib uri="http://commerce.ibm.com/foundation" prefix="wcf"%>

<portlet:defineObjects />

<fmt:setBundle basename="nl.MyAccountResource" />

<wcf:getData type="com.ibm.commerce.member.facade.datatypes.PersonType"
  var="person" expressionBuilder="findCurrentPerson">
</wcf:getData>
<c:set var="contactInfo" value="{person.contactInfo}" />

<portlet:actionURL var="ChangePersonalInformationActionURL">
  <portlet:param name="actionName" value="ChangePersonalInformation" />
  <portlet:param name="renderName" value="MyAccount" />
  <portlet:param name="faultRenderName" value="ChangePersonalInformation" />
</portlet:actionURL>
<portlet:actionURL var="MyAccountActionURL">
  <portlet:param name="renderName" value="MyAccount" />
</portlet:actionURL>

<form name="ChangePersonalInformation" method="post"
  action="<c:out value='${ChangePersonalInformationActionURL}' />">

<table cellpadding="0" cellspacing="0" border="0">

  <tr>
    <td><fmt:message key="ChangePersonalInformation.note" /></td>
  </tr>

```

This is an example of the `getData` tag used on the Change Personal Information JSP.



## Render example – get-data-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<wcf:get-data-config
  xmlns:wcf="http://www.ibm.com/xmlns/prod/commerce/foundation"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/xmlns/prod/commerce/foundation ../xsd/get-data-config.xsd ">

  <data-type>
    <name>Person</name>
    <type>com.ibm.commerce.member.facade.datatypes.PersonType</type>
  </data-type>
  <client-facade>
    <data-type-name>Person</data-type-name>
    <class>com.ibm.commerce.member.facade.client.MemberFacadeClient</class>
    <method>getPerson</method>
  </client-facade>
  <expression-builder>
    <name>findCurrentPerson</name>
    <data-type-name>Person</data-type-name>
    <expression-template>{self=true;ibmwcf.ap=$accessProfile$}/Person</expression-template>
    <param>
      <name>accessProfile</name>
      <value>All</value>
    </param>
  </expression-builder>

```

This is an example of the find current person expression builder definition.

## Render example – GetPerson request

```
<_mbr:GetPerson xmlns:_mbr="http://www.ibm.com/xmlns/prod/commerce/9/member" xmlns:_wcf="http://www.ibm.c
<oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
  <oa:CreationDateTime>2007-01-30T17:22:48.859Z</oa:CreationDateTime>
  <oa:BODID>6a709f60-b0b0-11db-81df-82e845bfc4f0</oa:BODID>
  <_wcf:BusinessContext>
    <_wcf:ContextData name="langId">-1</_wcf:ContextData>
    <_wcf:ContextData name="storeId">10001</_wcf:ContextData>
  </_wcf:BusinessContext>
</oa:ApplicationArea>
<_mbr:DataArea>
  <oa:Get>
    <oa:Expression expressionLanguage="wc:XPath">{self=true;ibmwcf.ap=All}/Person</oa:Expression>
  </oa:Get>
</_mbr:DataArea>
</_mbr:GetPerson>
```

This is a snapshot of the data area of the BOD from the previous example.

## getData tag example – Get expression builder

- Returns a list of SDOs – all products in a given category
- Paging information – varShowVerb, maxItems, recordSetStartNumber

```

<expression-builder>
  <name>getCatalogEntryByParentCatalogGroupId</name>
  <data-type-name>CatalogEntry</data-type-name>
  <expression-template>{_wcf.ap=${accessProfile$}/CatalogEntry[ParentCatalogGroupIdentifier[{UniqueID='${catGroupId$}'}]}</expression-template>
  <param>
    <name>accessProfile</name>
    <value>WC_CatalogEntrySummaryProfile</value>
  </param>
</expression-builder>
-----
<wcf:getData
  type="com.ibm.commerce.catalog.facade.datatypes.CatalogEntryType[]"
  var="products"
  expressionBuilder="getCatalogEntryByParentCatalogGroupId"
  varShowVerb="showProducts" maxItems="${pageSize}"
  recordSetStartNumber="${(currentPage - 1) * pageSize}"
  <wcf:param name="catGroupId" value="${param.catGroupId}"/>
</wcf:getData>
<:set var="numEntries" value="${showProducts.recordSetTotal}" />

<c:forEach var="product" items="${products}">
  <portlet:renderURL var="ProductDisplayURL">
    <portlet:param name="renderName" value="ProductDisplay" />
    <portlet:param name="catEntryId"
      value="${product.catalogEntryIdentifier.uniqueID}" />
  </portlet:renderURL>

  <c:out value="${product.description.name}" escapeXml="false" />

```

This is another getData tag example which returns a list of SDOs, along with paging information.

## Event handlers

- PortletActionHandler, PortletRenderHandler interfaces
  - ▶ Can be customized by portlet developer
- BusinessContextType, CallbackHandler objects
  - ▶ Must gather additional runtime information:
    - a business context object
    - an authentication callback handler object
  - ▶ Stored in portlet session and shared across with all portlets
  - ▶ Passed to downstream consumers, such as getData tag, as request attributes:

```
SessionHelper.SESSION_ATTRIBUTE_BUSINESS_CONTEXT_TYPES =  
"com.ibm.commerce.foundation.business_context"  
SessionHelper.SESSION_ATTRIBUTE_CALLBACK_HANDLER =  
"com.ibm.commerce.foundation.callback_handler"
```

In addition to using these generic event handlers, the MVCPortlet allows the portlet developer to customize both the action handler and the render handler.

To code a portlet action request handler, you need to provide additional information. When calling a client library, two mandatory objects are required - a business context object and the authentication callback handler. Both are discussed in subsequent slides.

## Authentication callback handler

- Two types of authentication between WebSphere Portal and WebSphere Commerce:
  - ▶ Basic authentication
    - Requires global security enabled with LDAP on WebSphere Portal only
    - Recommended for production environment
  - ▶ Simulated single sign-on
    - Does not require global security enabled and no LDAP setup for WebSphere Portal and WebSphere Commerce
    - Recommended for development environment only
- Generic authentication callback handler provided to perform simulated single sign-on
- Plug-in utility provided to manage callback handler object in the portlet session:

```
AuthenticationCallbackHandler authenticationCallbackHandler = SessionHelper  
    .getAuthenticationCallbackHandler(portletRequest, contextId);
```

Regardless of which authentication method you choose to use, the client library and WebSphere Commerce services should not be aware of the method when interacting with the portlet in the Portal environment. A callback handler is used to generate an identity token for use in calling a WebSphere Commerce service client library. The idea is to have the underlying service binding layer process the single sign-on request against the WebSphere Commerce server on behalf of the client without knowing the technical details.

In order to retrieve an existing authentication callback handler object from the portlet session, you can write code to call a special helper class using a contextId as shown in the slide.

## Business context

- Used to represent business contextual information defined for the portlet
- Plug-in utility is provided to manage this business context object in the portlet session:

```
BusinessContextType businessContextType = SessionHelper  
    .getBusinessContextType(portletRequest, contextId);
```

A `BusinessContextType` object is used to represent the business contextual information defined for the portlet. This contextual information should be set into the object for synchronizing with WebSphere Commerce services.

In order to retrieve an existing business context object from the portlet session, you can write code to call a special helper class using a `contextId` as shown in the slide.

## Calling client library example

- The following is an example of calling into the CatalogFacadeClient library:

```
CatalogFacadeClient client = new CatalogFacadeClient(businessContext, callbackHandler);  
CatalogType catalog = client.findCatalogByCatEntryId(new Long("10001").longValue());
```

- Upon application exception:

```
} catch (Exception e) {  
    if (e instanceof InvocationTargetException) {  
        Throwable cause = e.getCause();  
        if (cause != null  
            && cause instanceof AbstractBusinessObjectDocumentException) {  
            // Application exceptions
```

Here is an example of how to call the client library, passing the business context object and authentication callback as arguments.

## Authorization

- WebSphere Portal Server
  - ▶ Portlet resource level access control
  
- WebSphere Commerce Server
  - ▶ WebSphere Commerce fine grain access control

Because users sign on to the WebSphere Portal server and Portal performs the authentication, you should still use WebSphere Portal's resource level access control to restrict access on Portal pages, portlets, menu options and so on. However, when it comes to WebSphere Commerce related content, you should use back-end WebSphere Commerce fine-grain access control.



## Customization overview

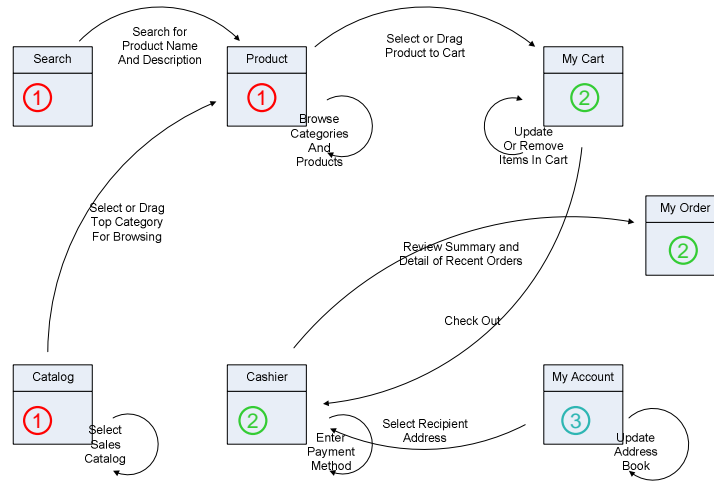
- Presentation layout
  - ▶ Look and feel – theme, skin
  - ▶ Portlet arrangement – page layout
- Portlet design
  - ▶ MVC portlet and WebSphere Commerce
    - MVC configuration
    - SOI interfaces – client library
    - Portlet JSP
  - ▶ Other JSR 168 portlets

Presentation and content are two aspects you should consider when re-using and customizing the sample portlets. With presentation, you should make use of WebSphere Portal's themes and skins to control the portlet frame decoration and the design of the overall Portal page. Also, by using the appropriate resource level access control, you can allow or restrict your Portal users from changing their own page layout. With content, in terms of what to display in the Commerce portlets, you have the option of re-using the default portlet features or building your own using the MVC style programming model.

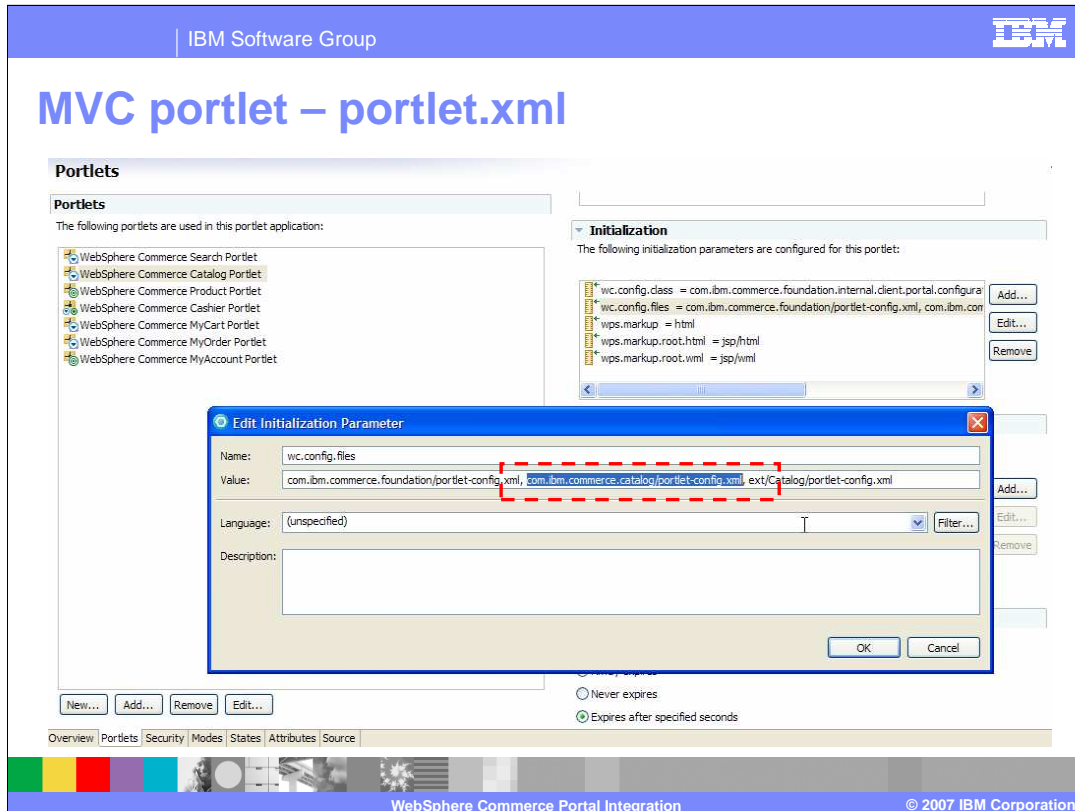
## MVC portlet – sample portlet features

### MVC Portlet Feature Configurations:

- 1) Catalog
- 2) Order
- 3) Member



The following portlet samples make up the WebSphere Commerce portal: My Account, Catalog, Cashier, Search, My Order, Product and My Cart. The diagram displays the shopping flow for an authenticated user in the WebSphere Commerce portlets.

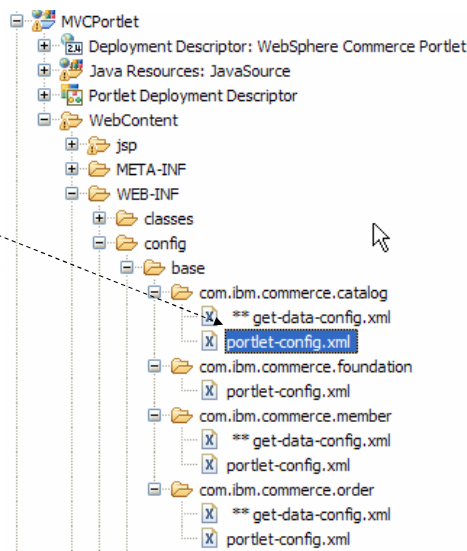
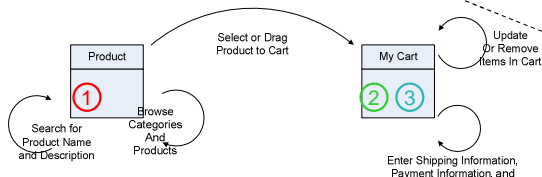


If seven ports are too many and you want to combine a few features into a single portlet; all you need to do is include the set of portlet features that you want inside of the portlet Web module's deployment descriptor as shown in the slide.

# MVC portlet – MVC configuration

## MVC Portlet Feature Configurations:

- 1) Catalog
- 2) Order
- 3) Member



The left of this diagram displays two portlets being used in a modified shopping flow for anonymous users. The right of this diagram shows the location of the portlet-config.xml file.

IBM Software Group 

## Services provided


Catalog		Order	
Catalog	<ul style="list-style-type: none"> <li>Get Catalog details by ID</li> <li>Get Catalog Details By Identifier</li> <li>Get Master Catalog</li> <li>Get All Catalogs</li> </ul>	Order	<ul style="list-style-type: none"> <li>Prepare Order</li> <li>Submit Order</li> <li>Delete Shopping Cart</li> <li>Add Order Items</li> <li>Update Order Items</li> <li>Delete Order Items</li> <li>Update Ship Info</li> <li>Add Payment Instruction</li> <li>Update Payment Instruction</li> <li>Remove Payment Instruction</li> </ul>
CatalogGroup	<ul style="list-style-type: none"> <li>Get CatalogGroup Summary By ID</li> <li>Get CatalogGroup Summary By Identifier</li> <li>Get CatalogGroup Details By ID</li> <li>Get CatalogGroup Details By Identifier</li> <li>Get CatalogGroup Merchandising Associations By ID</li> <li>Get CatalogGroup Merchandising Associations By Identifier</li> <li>Get Top Categories</li> <li>Get Category With Children Categories</li> <li>Get Category With Children CatalogEntries</li> <li>Get Category With All Children</li> </ul>		<ul style="list-style-type: none"> <li>Find Current ShoppingCart</li> <li>Find By Order Status</li> <li>Get History Orders</li> <li>Get Order By Id</li> <li>Get Usable Shipping Info</li> <li>Get Usable Payment Info</li> </ul>
CatalogEntry	<ul style="list-style-type: none"> <li>Get CatalogEntry Summary By ID</li> <li>Get CatalogEntry Details By ID</li> <li>Get CatalogEntry Merchandising Associations By ID</li> <li>Get CatalogEntry Merchandising Associations By PartNumber</li> <li>Get CatalogEntry Components By ID</li> <li>Get CatalogEntry Components By PartNumber</li> <li>Get CatalogEntry Summary By PartNumber</li> <li>Get CatalogEntry Details By PartNumber</li> <li>Find CatalogEntries Summary By PartNumber</li> <li>Find CatalogEntries Details By PartNumber</li> <li>Find CatalogEntries Summary By Name</li> <li>Find CatalogEntries Details By Name</li> <li>Find CatalogEntries Summary By Description</li> <li>Find CatalogEntries Details By Description</li> </ul>		

WebSphere Commerce Portal Integration © 2007 IBM Corporation

There are four services provided in this feature pack: Catalog, Order, Member and Contract.

The **Catalog** services enable an external system, such as WebSphere Portal, to search for Catalog related information in WebSphere Commerce.

The **Order** services provide shopping cart, order capture, order fulfillment, inventory, and payment function support that can be utilized by WebSphere Portal.

IBM Software Group 

## Services provided (cont.)

<b>Member</b>		<b>Contract</b>
Person	<ul style="list-style-type: none"> <li>Find Current Person</li> <li>Find Person By UniqueId</li> <li>Find Person By Distinguished Name</li> <li>Register Person</li> <li>Update Person</li> <li>Add Address</li> <li>Update Address</li> <li>Delete Address</li> <li>Register Person (for backend)</li> <li>Update Person (for backend)</li> <li>Add Address (for backend)</li> <li>Update Address (for backend)</li> </ul>	<ul style="list-style-type: none"> <li>Contract</li> <li>Get Contract By ID</li> <li>Get Eligible Contract List</li> </ul>
Organization	<ul style="list-style-type: none"> <li>Find Org By UniqueId</li> <li>Find Org By Distinguished Name</li> <li>Register Organization</li> <li>Update Organization</li> <li>Add Address</li> <li>Update Address</li> <li>Delete Address</li> <li>Register Org (for backend)</li> <li>Update Org (for backend)</li> <li>Add Address (for backend)</li> <li>Update Address (for backend)</li> </ul>	

WebSphere Commerce Portal Integration © 2007 IBM Corporation

The **Member** services allow an external system, such as WebSphere Portal, to create, update and search for Members (Organizations, Users, and Member Groups) in WebSphere Commerce.

The **Contract** services provide the ability to retrieve all entitled contracts for a given user in a store and retrieve the contract content by contract Id.

## Sample portlet JSPs

- Catalog
  - ▶ CatalogDisplay
  - ▶ TopCategoryDisplay, StaticCategoryDisplay, DynamicCategoryDisplay
  - ▶ StaticProductDisplay, DynamicProductDisplay
  - ▶ SearchDisplay
- Order
  - ▶ OrderItemDisplay
  - ▶ OrderCheckoutDisplay
  - ▶ OrderConfirmationDisplay
  - ▶ OrderStatusDisplay, OrderDetailDisplay
- Member
  - ▶ AccountDisplay
  - ▶ ChangePersonalInformationDisplay
  - ▶ AddressBookDisplay, AddressDisplay
  - ▶ AnonymousAddressDisplay

Here are a list of JSPs included in the WebSphere Commerce portlets.

## References

- **WebSphere Commerce Portal Integration**

▶ <http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/index.jsp?topic=/com.ibm.commerce.base.doc/concepts/covwhatsnewinthisrelease.htm>



For more information regarding WebSphere Commerce Portal Integration, visit the site indicated in the presentation.



## Feedback

### Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

[Click to send e-mail feedback](#)



You can help improve the quality of IBM Education Assistant content by providing feedback.

## Trademarks, copyrights, and disclaimers

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

IBM	CICS	IMS	MQSeries	Tivoli
IBM (logo)	Cloudscape	Informix	OS/390	WebSphere
eI (logo)/business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

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

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

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

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

Linux is a registered trademark of Linus Torvalds.

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

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. 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 in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

© Copyright International Business Machines Corporation 2007. All rights reserved.

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

