
Welcome to the WebSphere Commerce V6 presentation. This presentation describes methods and best practices for developing and customizing storefront pages.

## Unit objectives

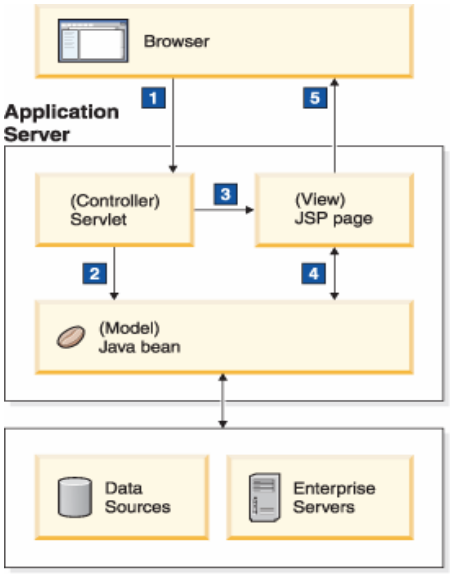
- Describe the view layer in WebSphere Commerce
- Describe the use of the WebSphere Commerce Struts framework
- Understand how to customize the WebSphere Commerce storefront
- Describe JSP™ best practices for WebSphere Commerce




This presentation introduces you to the architecture of the WebSphere Commerce storefront. This presentation introduces the basics of Struts and the use of Struts in WebSphere Commerce. Methods for customizing the storefront and JSP best practices are discussed.

## WebSphere Commerce view layer

- WebSphere Commerce storefront uses JSP pages throughout the entire store and administrative flows
- WebSphere Commerce browser-based tools
- WebSphere Commerce Accelerator
- WebSphere Commerce Administration Console
- Organization Administration Console
- WebSphere Commerce storefronts







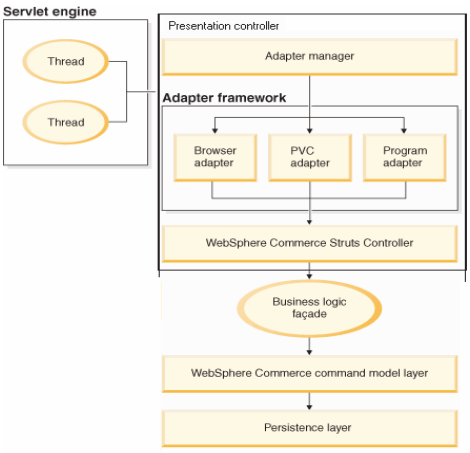
WebSphere Commerce uses Java Server Pages (JSP) to implement the view layer of the Model-View-Controller (MVC) design pattern. The view layer is in charge of retrieving data from the database through the use of data beans and formatting it to meet the display requirements. The view layer determines whether the request is sent to a browser or streamed out as XML. JSP files present a clean separation between data content and presentation.

WebSphere Commerce ConsumerDirect, B2B Direct, and other storefronts use JSP pages throughout the entire store flow. WebSphere Commerce browser-based tools (WebSphere Commerce Accelerator, Administration Console, and Organization Administration Console) use JSP pages in conjunction with XML to drive the user interface.

As of WebSphere Commerce V6.0, the WebSphere Commerce Web application has moved from a proprietary model-view-controller implementation to the Struts open source implementation. In the Struts framework, a browser request is routed to a servlet that acts as a controller. The controller, using local Java calls, calls the model for processing. The controller then dispatches the appropriate view to render data. The model encapsulates all business logic (implemented by following the command pattern) and data (implemented using JSP pages). The JSP pages retrieve data from the database using data beans, and then format the output.

## View layer components

- Adapter framework
- Adapters
- Controller
- Data bean manager
- Data beans
- Display pages (JSP)







In WebSphere Commerce, the server runtime defines the framework for handling system and user requests, and performs the appropriate business logic to process the requests. The framework is built using an MVC design pattern and provides an environment that hosts business logic and handles persistence. It performs such tasks as transaction management and session management. The components that make up, and interact with, the view layer are discussed here.

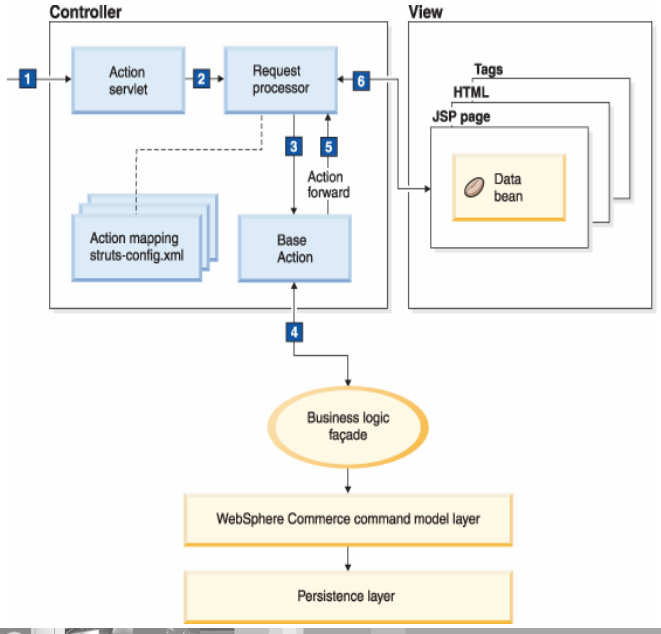
The adapter framework determines which adapter is capable of handling a request and associates the appropriate adapter with the request for response building and session management. WebSphere Commerce adapters are device-specific components that perform processing functions before passing a request to the Struts controller. Specific adapters that come with WebSphere Commerce are the HTTP browser adapter, HTTP PVC (Pervasive computing) adapter, and HTTP Program adapter. The HTTP browser adapter provides support for all browser-based requests. The HTTP PVC adapter is an abstract adapter class that can be used to develop adapters for specific pervasive computing devices, such as a cellular telephone. The program adapter provides support for remote programs invoking WebSphere Commerce commands.

The controller plays a role in enforcing the programming model for the application. For example, the programming model defines the types of commands that an application should write. 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.

WebSphere Commerce data beans inserted into JSP pages allow for the inclusion of dynamic content in the page. The recommended way of activating the data beans within a JSP page is by means of the WebSphere Commerce useBean tag. Alternatively, the data bean manager can be used. Access control is enforced by invoking data beans using the data bean manager. Data beans are Java beans that are primarily used by Web designers. Most commonly, they provide access to a WebSphere Commerce entity.

WebSphere Commerce uses Java Server Pages for display. JSP pages are specialized servlets that are typically used for display purposes. Upon completion of a URL request, the Web controller invokes a view command that invokes a JSP page.

## WebSphere Commerce Struts framework







As of WebSphere Commerce V6.0, the WebSphere Commerce Web application has moved from a proprietary model-view-controller implementation to the Struts open source implementation. Struts is a well-documented framework for Java Enterprise Web application development, which has become an industry standard for deploying model-view-controller applications. Struts enforces best practices and design patterns, boasts a large developer community, and is supported by IBM development tools such as Rational Application Developer. Among key benefits of Struts are its support for dynamic action forms, form validation, versatile tag libraries, and Tiles. WebSphere Commerce has extended the base Struts configuration model to provide the traditional WebSphere Commerce function seen in previous releases.

This diagram expands on the processing of a web-based request through the WebSphere Commerce Struts framework.

Step 1: The Action servlet receives an HTTP request.

Step 2: The Action servlet routes the request to the module's request processor.

Step 3: The request processor passes the request, action form, and action mapping to the base action.

Step 4: Actions use action form data to invoke business logic operations on behalf of the client

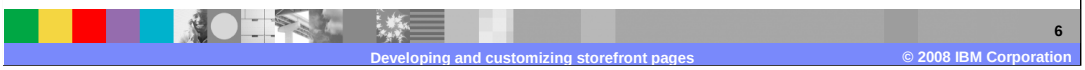
Step 5: An ActionForward object is returned that indicates what view the controller should forward to.

Step 6: The request processor forwards to the appropriate view element when the action is completed.

The next slide describes the function of some key Struts components in more detail.

## Struts configuration files

- WebSphere Commerce V5
  - Commands are registered by adding records to URLREG, VIEWREG, and CMDREG
- WebSphere Commerce V6
  - In the WebSphere Commerce Struts Framework, the URLREG and VIEWREG tables are replaced with Struts configuration files
  - CMDREG remains
  - Product configuration found in struts-config.xml
  - Migration configuration found in struts-config-migrate.xml
  - Customization configuration found in struts-config-ext.xml

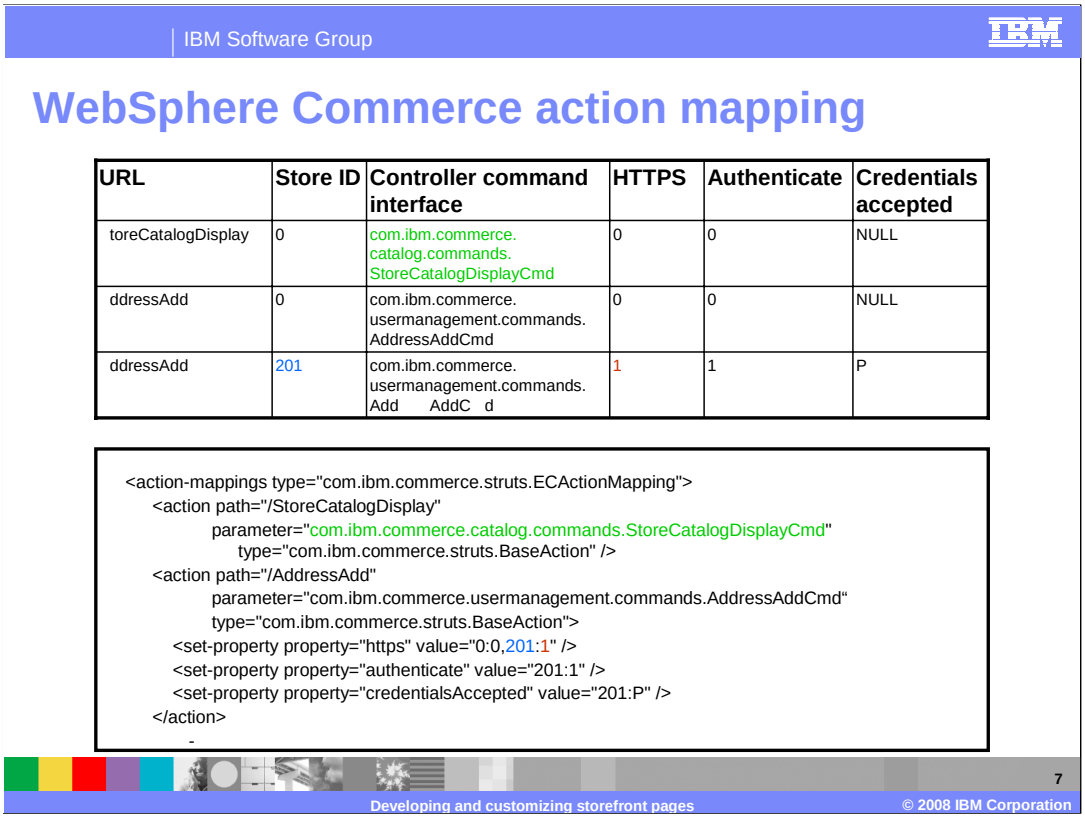





Prior to the introduction of Struts in WebSphere Commerce, all commands were registered by adding records to the URLREG, VIEWREG and CMDREG database tables. The URLREG table mapped supported URLs to controller command interfaces. VIEWREG mapped view names to JavaServer Pages and the CMDREG table mapped command interfaces to implementation classes.

The URLREG and VIEWREG tables have been replaced by Struts configuration files. All major aspects of a Web application are configured declaratively by means of Struts configuration files.

Action-mappings, global-forwards, message-resources, data-sources, form-beans, global-exceptions, and so forth, are among key elements found in Struts configuration files. Of these, WebSphere Commerce extends the semantics for action-mappings, global-forwards, and message-resources. WebSphere Commerce uses multiple Struts configuration files. There can be multiple configuration files defined for each Web module. You can find more information on the Struts configuration files for WebSphere Commerce in the Information Center.




Associating a URL with a controller command interface is accomplished in the WebSphere Commerce Web application with action-mappings elements. The table at the top demonstrates three sample URLREG entries. Store ID is the store reference number for this URI or 0 to mean any store. A value of one in HTTPS indicates that the request was expected to be received on a secure channel (HTTPS). A redirect to the SSL port is issued if it has been received on an insecure channel (HTTP). A value of one in Authenticate indicates that user logon is required for this URI. If a user is not logged on when trying to access this URI, they are redirected to the logon page before being able to proceed. In Credentials Accepted, the value of P indicates that partially authenticated users are entitled to access this resource.

The Struts action-mappings configuration element on the bottom describes these same associations. Note the use of the parameter attribute of the action element to specify the name of the interface of the controller command to invoke. Also note the values of the type attributes and the syntax of the value attribute of the nested set-property element: a comma-separated list of *storeID* : *propertyValue* pairs that defaults to zero.

# WebSphere Commerce view configuration

| View name | Device format ID | Store ID | View command interface | View command implementation class | Properties | HTTPS | Credentials Accepted |
|---|---|---|---|---|---|---|---|
| AddressBookForm | -1 | 0 | com.ibm.commerce.command.ForwardViewCommand | com.ibm.commerce.command.HttpForwardViewCommandImpl | docname=Address-BookForm.jsp | 0 | NULL |
| AddressBookForm | -1 | 201 | com.ibm.commerce.command.ForwardViewCommand | com.ibm.commerce.command.HttpForwardViewCommandImpl | docname=UserArea/AccountSection/AddressbookSubsection/AddressBookForm.jsp | 1 | P |

```
<global-forwards>
   <forward name="AddressBookForm" path="/AddressBookForm.jsp" />
   <forward className=" com.ibm.commerce.struts.ECActionForward" name="AddressBookForm /201"
            path="/UserArea/AccountSection/AddressbookSubsection/AddressBookForm.jsp">
    <set-property property=" resourceClassName" value="com.ibm.commerce.command.HttpForwardViewCommandImpl" />
   </forward>
</global-forwards>

<action-mappings type="com.ibm.commerce.struts.ECActionMapping">
   <action path="/AddressBookForm" type="com.ibm.commerce.struts.BaseAction">
      <set-property property="https" value="0:0,201:1" />
      <set-property property="credentialsAccepted" value="201:P" />
   </action>
</action-mappings>
```




Describing device-specific and store-specific view implementations, accomplished in earlier versions of WebSphere Commerce using the VIEWREG database table, is accomplished in the WebSphere Commerce Web application with a combination of action-mappings and global-forwards elements. The table at the top demonstrates a sample VIEWREG entry. Device format ID is the identifier of the device to which the view is sent. The default device format is -1, which represents an HTTP Web browser. Store ID is the store reference number for this URI or zero to mean any store. Properties contain name-value pairs used by this view, in the form of an HTTP request query string. A value of one in HTTPS 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). In Credentials Accepted, the value of P indicates that partially authenticated users are entitled to access this resource.

The Struts global-forwards and action-mappings configuration elements on the bottom describe these same associations. Note the value of the className attribute of the forward elements and the syntax of the name attribute of the forward elements: a *docname*/ *storeID* pair, whose second constituent defaults to 0. Also note the use of the resourceClassName property to specify the view command implementation class, whose value defaults to com.ibm.commerce.command.HttpForwardViewCommandImpl.

Although not shown here, you can also set the authenticate property for views, with the same syntax and semantics as for URLs as shown on the previous slide.

## Working with JavaServer Pages

- Dynamic templates
- Sun standard JSP 2.0 supported with WebSphere Commerce 6
- Separate dynamic content development from page design work
- Page Designer




Now that you have seen how the Struts framework is used to control and configure the view layer, it is time to look at how pages are created.

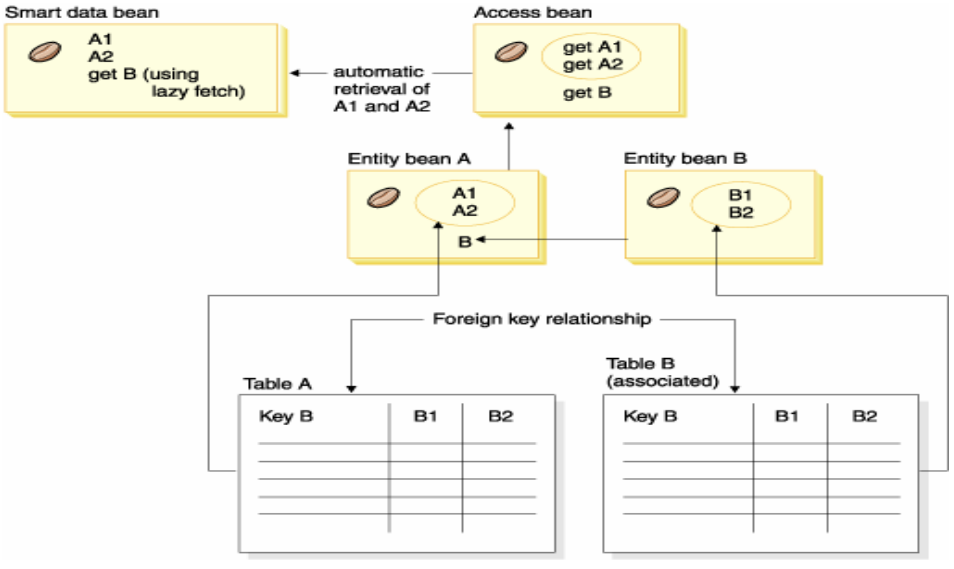
WebSphere Commerce uses JSP technology to build dynamic content from the database and reformat it for online display. JSP technology allows you to insert dynamic content into static Web pages. The JSP page contains the code to retrieve required information and format the output for the browser. When a customer requests a JSP page, the WebSphere Application Server interprets the JSP tags and scriptlets, creates the content in the form of an HTML page, and returns it to the browser.

Using JSP technology to create your dynamic pages allows you to separate the development of the dynamic content from the development of the page design, which is typically created using static HTML. For example, a Web designer can create the visual appearance of the page in HTML or XML. At the same time, the store developers, with programming skills in Java, JavaScript, HTML and JSP technology can add the dynamic content to the page. As the development process evolves, both the Web designer and the store developer can update the pages without changing the other's work.

With minimal training, a Web designer can use a JSP editor to insert the dynamic elements the store developer has created into the store pages.

WebSphere Commerce includes a set of data beans that you can drag anywhere on a JSP page. These beans allow you to retrieve information from the database without writing any code. You can also add your own images, static text, tables, and other elements using Page Designer's WYSIWYG page editing function without any prior programming knowledge.

## Working with data beans







A data bean is a Java bean that is used within a JSP page to provide dynamic content. A data bean normally provides a simple representation of a WebSphere Commerce entity bean. The data bean encapsulates properties that can be retrieved from or set within the entity bean. As such, the data bean simplifies the task of incorporating dynamic data into JSP pages. Store developers should consider properties of the store and globalization issues when developing 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 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.

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 can be costly in terms of performance. 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. Only use command data bean if using a smart data bean is impractical.

## Adding pages to the WebSphere Commerce storefront

- Develop a list of store pages needed
  - Store shopping flow
  - Error pages
- Develop a list of command and view URLs
- Set access control for a page




In order to develop a list of the pages needed to create your store, you need to know the business and functional requirements of the store, in addition to any business processes that have been defined. Refer to the business processes for the consumer direct starter store in the Information Center for an example. These business processes help you understand the flow of the Consumer Direct store and can be used as a guide to create additional business processes for your own store.

Once business processes are available, you can create the shopping flow for your store. The shopping flow reflects the requirements and business processes defined for your store, illustrating how a customer moves through the store. The exception flows in your business processes can help you determine what error pages you need to create for your store. For every new exception flow that you create, you need to create either an error page or an error message.

Just as you developed a list of pages necessary to create the store, you also need to develop a list of the command and view URLs necessary to implement the business processes for your store. Using the shopping flow diagram for your store, and the list of default commands and views, identify the URLs necessary to complete each action. Understanding which command and view URLs are used in the consumer direct stores can also help you determine what URLs you need in your store.

When the consumer direct store is published, access control policies for the store are loaded into the database. You need to specify the access control policies for your new URLs. This presentation does not cover the details of access control, refer to the accesscontrol.xml file or the Information Center for an example of how to create an access control file.

## JSP best practices

- Use the JavaServer Pages Standard Tag Library in place of Java code
- Use the Commerce-specific tag for bean activation
  - wcbase:useBean
- Use Commerce-specific maps to access request parameters
  - <c:out value="${WCParam.catalogId}" />
- Use the StoreErrorDataBean data bean for error handling
  - <c:if test="${!empty storeError.key}">
    <c:set var="errorMessage" value="${storeError.message}" /> </c:if>




The JavaServer Pages Standard Tag Library (JSTL) is a collection of JSP tags that provide standard functionality most commonly sought by JSP page authors. JSTL has support for conditions, iteration, locale-sensitive formatting, and so forth. It also has an expression language that allows page authors to control how data is retrieved and displayed. Store JSP pages should contain JSTL tags and little or no Java code. Any business logic should be delegated to page-friendly data beans, and the remaining presentation logic should be implemented in JSTL.

WebSphere Commerce data beans require activation before they can be used. WebSphere Commerce provides a Commerce-specific version of the useBean tag, wcbase:useBean, that performs data bean activation in a Java-free manner and is the recommended method of data bean activation in store JSP pages.

WebSphere Commerce provides customized versions of the implicit JSP objects param and paramValues to facilitate access to decrypted HTTP request parameters. WCParam is a Map object that maps the name of a request parameter to its single String value. WCParamValues is a Map object that maps the name of a request parameter to a String array of all values for that parameter.

To display store-specific error messages in JSP pages, use the StoreErrorDataBean data bean. This data bean provides methods to retrieve the store error key and the error message parameters, that is, the substitution parameters used in the error messages. In mapping error codes to error messages, the StoreErrorDataBean data bean relies on the existence of the store error message properties file. The store error message properties file should contain error message definitions for all the identified exception conditions in your store flow.

# JSP best practices continued

- Use an appropriate inclusion mechanism
  - Static: <%@ include file=" *filename*.jspf" %>
  - Dynamic: <c:import url=" *filename*.jsp"> ... </c:import>
  - Dynamic: <jsp:include page=" *filename*.jsp" flush=" *true*|*false*"> ... </jsp:include>
- Use the .jspf extension for JSP segments
- Use the escapeXml attribute to preserve HTML formatting
  - <c:out value="${product.listPrice}" escapeXml="false" />
- Ensure XHTML compliance
- Use the POST method for form submission




JSP supports two inclusion mechanisms, static include and dynamic include. The static include directive causes the content of the specified file to be textually inserted into the including file at compile time, that is, when the JSP page is translated into a Java servlet. The dynamic include actions include the response generated by executing the specified page during the request processing phase. Since what is included is the *response* generated by the page, and not the *content* of the page itself, scripting variables declared in the included file are not visible elsewhere in the resulting JSP page. While the dynamic include is more flexible than the static include, it falls short in terms of efficiency. As a consequence, use static includes whenever possible to avoid performance issues.

To enable code development and support tools to differentiate between types of files, use the .jsp extension only for the source files of complete JSP pages. Use the .jspf extension for the source files of JSP segments.

By default, the value of the escapeXml attribute of the JSTL <c:out> tag is true. This default behavior ensures that HTML special characters, such as less than, greater than, ampersand, and quotation marks are converted into their corresponding character entity codes and displayed properly in the HTML page. In some common WebSphere Commerce store scenarios, however, this behavior is counterproductive. One such scenario is the display of prices in globalized stores where the ampersand symbol might be part of a currency character representation. To prevent the conversion in this case, escapeXml should be explicitly set to false as illustrated.

To ensure XHTML compliance, use lowercase for all element and attribute names and enclose all attribute values in double quotation marks. Also, ensure that every element has an end tag or is self-terminating if it is empty. For HTML-compatibility, include a space in all empty element tags before closing the tag. Ensure that the HTML page that your JSP page produces begins with a valid document type declaration. You can ensure XHTML-compliance by using XHTML validators, such as the WebSphere Commerce Developer HTML Validator or the W3C HTML Validator.

The default form submission method is to use the GET method. This has limitations within browsers in the total amount of data that can be submitted and placing exposed and hidden field parameters on the URL address line. Instead, use the POST method for form submission.

## JSP error handling

Error handling for JSP pages can be performed in various ways

- Within the current page

```
SomeDataBean sdb = new SomeDataBean();
sdb.setSomeProperty("");
try { com.ibm.commerce.beans.DataBeanManager.activate(sdb, request); }
catch(Exception ex) { //Handle the exception in whichever way you want.. }
```

- Outside of the current page
  - Page level error handling
  - Application level error handling




For JSP pages that require intricate error handling and recovery, a page can be written to directly handle errors from the data bean on the same page. The JSP page can either catch exceptions thrown by the data bean or it can check for error codes 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. The JSP page should use try-catch blocks to capture the exception so that it can take appropriate action based on the exception type. An example of a JSP snippet using try and catch Java statements is shown.

Another option is to have a separate dedicated error handling page. When an exception occurs in the current JSP page the request is delegated to the error page. When using a separate error handling page, you can have two options, error handling at the page level or at the application level.

When using page level error handling, a JSP page can specify its own default error page from within an exception catch block through the JSP error tag. This enables a JSP page to specify its own handling of an error. For a JSP page that does not contain a JSP error tag, the error falls through to the application-level error JSP page. In a page-level error JSP page, a helper class must be called to roll back the current transaction.

An application under WebSphere Commerce can specify a default error JSP page when an exception occurs within any of its servlets or JSP pages. The application-level error JSP page can be used as a site level or store level 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. Use the application-level error handling strategy only when required. You can include an error handler at the application level by using the deployment descriptor of the Web application. The error handler can be specific to an exception type or can be generic for all exceptions.

## WebSphere JSP viewer

The WebSphere Commerce JSP viewer is composed of three components:

- Web controller
- useBean Tag Library
- XML generator

Formerly known as the JSP Preview Environment




You can use the JSP viewer, formerly named the JSP preview environment, to view your JSP files without the required underlying Java code. The JSP viewer allows JSP developers and Java developers to work simultaneously instead of consecutively: JSP developers do not have to wait for necessary Java code to start developing their pages. The WebSphere Commerce JSP viewer is composed of the Web controller, useBean tag library and the XMLGenerator.

The Web controller in the JSP viewer provides JSP pages with linkages to other JSP pages. Without a WebSphere Commerce environment, the HTML links in the JSP pages can not work. The Web controller mimics WebSphere Commerce run time using a XML configuration file, allowing you to see the JSP pages and simulated store flow without running any business logic. As a result, you can get an early estimation of coding effort and design problems.

The useBean tag library provides JSP pages with data that is pulled from an XML file. In the WebSphere Commerce environment, it instantiates and activates WebSphere Commerce beans, enabled using JavaServer Pages Standard Tag Library technology. All JSP pages in the WebSphere Commerce starter stores use this technology. The tag library is a wrapper to an XML parser that reads in a definition for a data bean and then re-creates that data bean with a series of lists and maps. The data bean is then placed into the proper JSP page scope. The BeanLocation.XML file provides the mapping between the bean ID and the XML file. Pages that use the same data bean end up with the same data set.

The XMLGenerator is a data bean to XML generator that is capable of creating XML files for a given WebSphere Commerce data bean. To generate the XML file, you must have a populated data bean or a stubbed out data bean. The generator starts populating every method on the data bean and checks the results. This population process is done automatically when using the preview tool. In the case of a WebSphere Commerce bean (the one to be populated), you must set parameters. In the case of a stubbed data bean, you do not have to set parameters.

## Storefront customization strategies

- Customize existing components
  - Through the WebSphere Commerce Accelerator
- Add supported features
  - Extensions or modification through the WebSphere Commerce Developer Environment
- Build new customization components
  - New features developed with the WebSphere Commerce Developer Environment

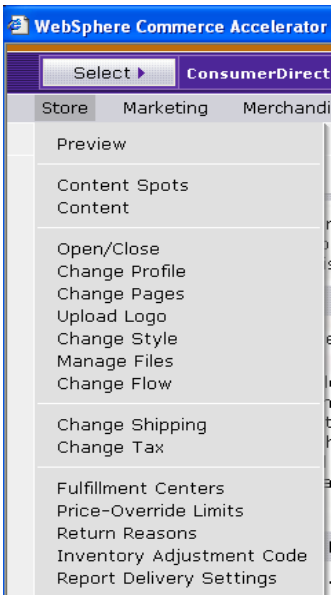



There are many ways to customize a WebSphere Commerce storefront.

Changes to site flow, basic site templates and features can be made from within Accelerator and do not require any coding to change. Existing sample store components can be customized through WebSphere Commerce Developer by adding WebSphere Commerce features such as marketing promotions or auctions. Finally, new components can be created to support features unique to your storefront. This type of customization involves creating new commands and is discussed in the presentation **Developing and customizing store business logic.**

## Customizing existing storefront components

- WebSphere Commerce Accelerator
- Content management
- Store logo
- Change style
- Change store flow







WebSphere Commerce Accelerator provides tools to change various elements of the storefront, such as the store or site's logo, flow, text, and style. WebSphere Commerce starter stores, such as the Consumer Direct starter store, showcase many such configurable elements. Depending on the needs of the store or site, the store developer can make additional choices for flow, text, and style available to the WebSphere Commerce Accelerator tools by modifying the store or site's storefront assets. The next few slides provide background information necessary to understand runtime storefront configuration and describe the JSP page and configuration file changes needed to enable additional choices for flow, text, and style.

## Style configuration of WebSphere Commerce

- Styles
  - Appearance of the store's headers, footers, and sidebars
- Colors
  - Style-color combinations that can be used in the store
- Banners
  - Color-banner combinations that can be used in the store

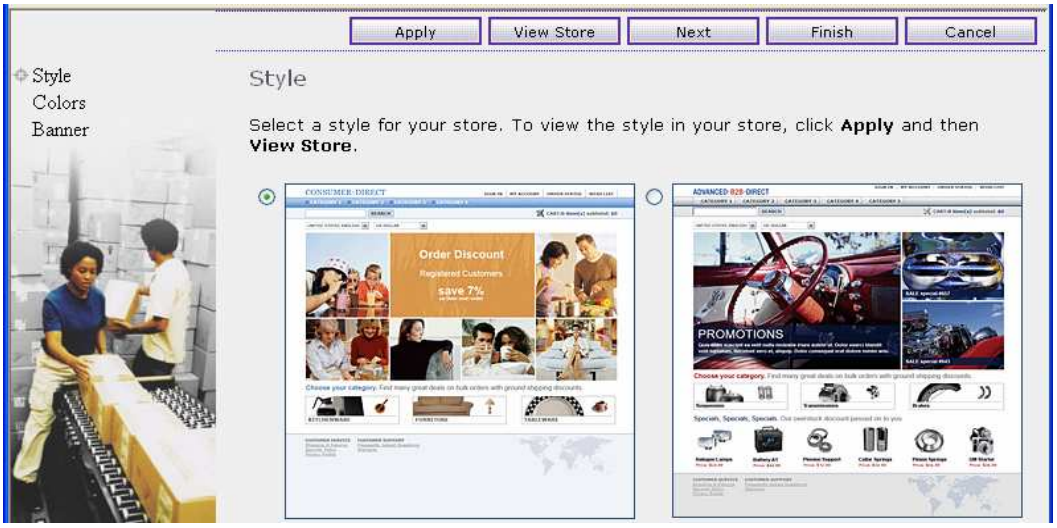



WebSphere Commerce provides the capability to dynamically change the appearance of stores that are based on WebSphere Commerce starter stores by varying style-color-banner combinations. This capability is realized through the Change Style wizard, which relies on the WebSphere Commerce Flow infrastructure.

Three types of options are available in the Change Style wizard: styles, colors and banners. Styles determine the appearance of the store's headers, footers, and sidebars. Each style choice maps to a store directory containing the header, footer, and sidebar JSP pages implementing that style. Colors determine the style-color combinations that can be used in the store. Each color choice maps to its own style sheet and a directory containing the store's images rendered in that particular color. Banners determine the color-banner combinations that can be used by the store. Each banner choice maps to a banner image located under the store's images directory.

The style, color, and banner options available in a given store are defined in the style configuration file, style.xml. The style configuration file refers to features and components defined in the flow repository.  The components defined in the repository map to the file assets such as JSP, image, and Cascading Style Sheet files. Adding new style options requires modifying the style and repository XML files, in addition to adding new JSP pages, images, and style sheets.

# Style modification

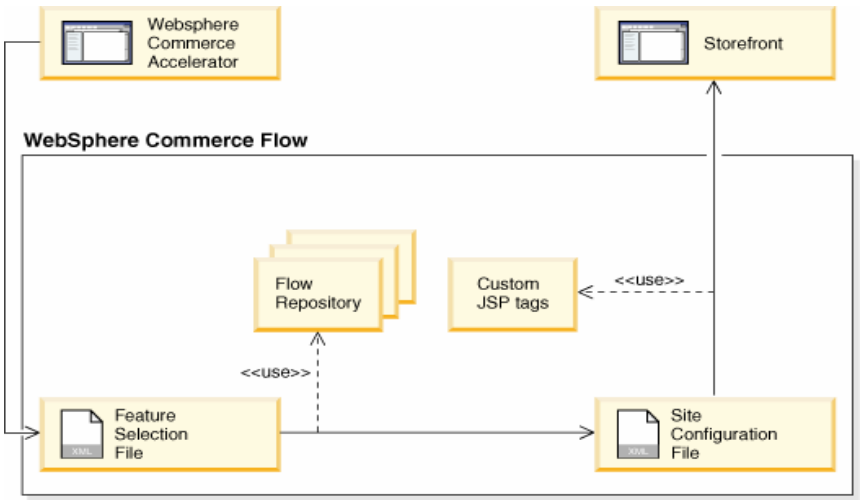






This slide shows the first screen in the Change Style wizard in Accelerator. You can select from the available styles to specify the appearance of the store's headers, footers, and sidebars. Colors and banner can be selected on the later pages of this wizard.

## Site flow infrastructure







WebSphere Commerce Flow describes the configurable features that are available in a store or site, and provides a means to enable or disable individual features. Store or site administrators can configure such features through a UI, such as the Change Flow notebook, changing the flow or style of the store or site without making any changes to its JSP or properties files. After applying the new store configuration, the changes are visible immediately by refreshing the store pages in the Web browser.

The diagram illustrates the basic infrastructure.

The Flow Repository is a collection of XML documents describing the store or site. There are two main parts to the repository, a description of the site components and a description of all the configurable features available for the store. Each element in the Flow Repository has a unique ID so that it can be referenced by other elements in the repository. It can also be referenced from other parts of WebSphere Commerce Flow, such as the Feature Selection File, Site Configuration File, custom JSP tags, and the UI.

The Feature Selection File (FSF) contains the set of store or site features that have been enabled. The FSF is updated by the Flow UI to capture your selections and represents a particular configuration of the store. Initially, the FSF contains the list of preset features. The FSF is applied to the repository to generate the Site Configuration File.

The Site Configuration File (SCF) contains the minimum amount of information that is required to configure the store at run time. Namely, it contains the list of enabled features and the paths (URLs) for exit ports and file references. At run time, the Flow custom JSP tags only use the SCF to determine which portions of the store's JSP pages should be enabled or disabled and which paths should be used.

Your ability to configure store or site features through the UI is accomplished by the WebSphere Commerce Flow custom JSP tags in the JSP files of the stores based on WebSphere Commerce starter stores.

## WebSphere Commerce site flow modification

- Store flows
  - Consumer Direct, Advanced B2B Direct, B2B Direct
- Site flow infrastructure
  - <flow:ifEnabled>
  - <flow:ifDisabled>
- Changing site flow
  - Apply
  - Apply permanently

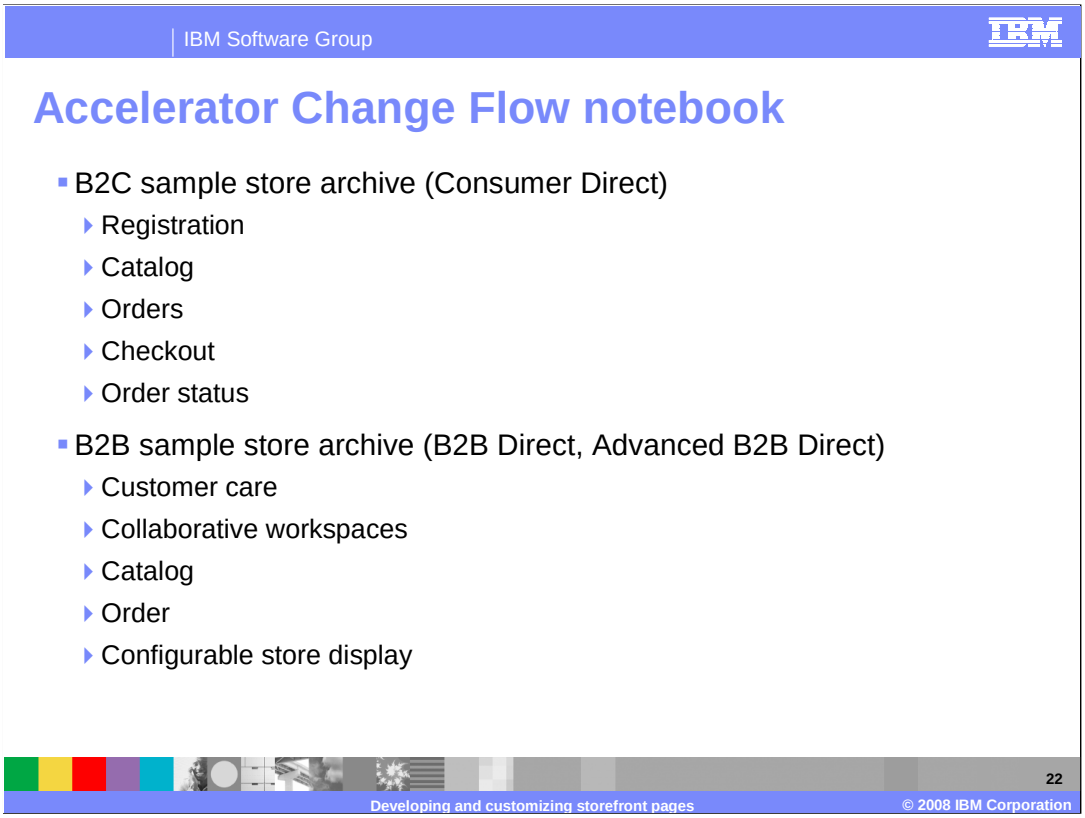



Sample store archives provided with the different versions of WebSphere Commerce allow for modification of the flow and features that your users experience. These features can be applied point in time or permanently.

The JSP files in the starter stores listed include customized tags that WebSphere Commerce Accelerator uses to enable or disable the selected features in your store. The ifEnabled tags enclose the portion of the JSP files that is applicable only when the specified feature is enabled. The ifDisabled tags enclose the portion of the JSP files that is applicable only when the specified feature is disabled.

After configuring the store-flow features, you have two options: Apply or Apply Permanently. If you choose Apply, WebSphere Commerce Accelerator uses the appropriate tags to control how the feature displays in the store. The tags remain in place and the JSP files remain unchanged. However, if you choose Apply Permanently, WebSphere Commerce Accelerator removes the portions of the JSP files that is used. That is, if you have selected to enable a feature, WebSphere Commerce Accelerator leaves only the portion of the JSP file that is enclosed within the ifEnabled tags. The portion of the JSP file that is enclosed within the ifDisabled tags is removed. Both tags are also removed.

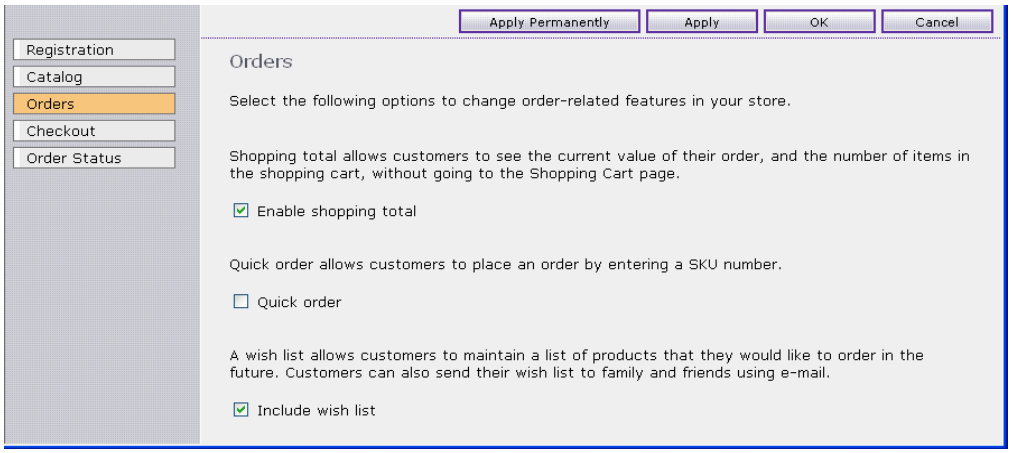
Once you have chosen Apply Permanently, you can no longer reconfigure *any* of the store-flow features within your store. Before making any site flow modifications, you should archive the JSP files for the store. This gives you the option of undoing permanently applied changes if needed.




Each sample store archive that supports site flow includes features that can be delineated by the ifEnabled and ifDisabled JSP tags. A simple search of the JSP files in your archive can give you access to where these features are used within the store pages.

Each of these flow features has specific instances of when they should be used and what the implications to the site data and configuration that should be completed before their enablement. The next slide shows an example of the Orders page from the B2C change flow notebook.

# Orders site flow options





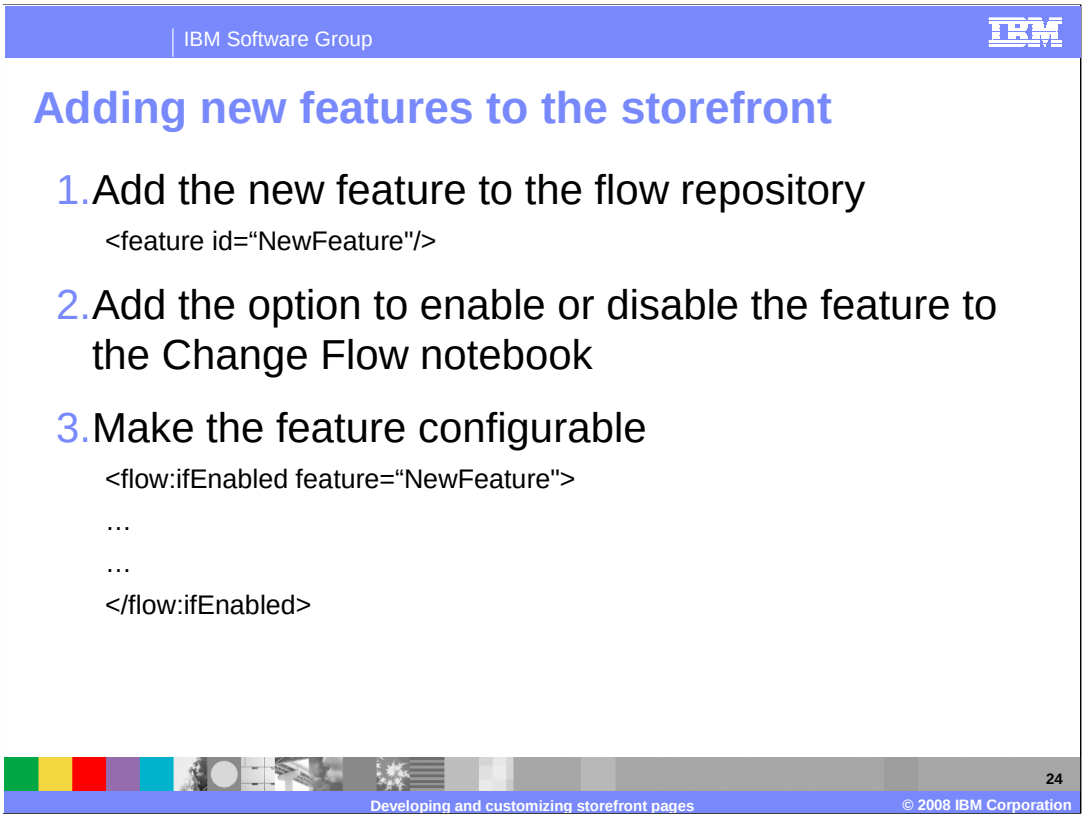


Here you see options for the update of the B2C store model for order options. All of the options are selected by default for the Consumer Direct store archive except for the Quick Order functionality.

Enable shopping total allows customers to see a rollup of their shopping total and number of items without invoking the Shopping Cart page (OrderItemDisplay URL command).

Quick order allows customers to enter a part number and directly add products to their shopping cart without browsing the catalog. This is very useful if for instance you have paper catalogs for different apparel lines that are sent to the customer and can then be quickly converted into online sales. This might require additional site customization if the customer needs to select a specific catalog to get specific catalog pricing or promotions.

Include wish list allows customers to add products to a wish list in addition to the normal shopping cart site flow.




It is possible to make additional features configurable through the change flow notebook.

The first step is to add the new feature to the flow repository. Open the Features.xml file for the store you want to configure. Within the features element in the file, add the line shown. Save and close the file.

Next, you need to add the new feature to the change flow notebook. The easiest way to do this is to use an existing page as a basis. Locate the UI directory for the flow component and list the contents of the directory. You see an XML file for each page of the Change Flow notebook. Make a copy of the OrderStatus.xml file in the same folder and name it NewFeature.xml. Open your new file and remove the second option-group entry. Change all occurrences of OrderStatusPanel to NewFeaturePanel. Also change TrackingStatus and OrderStatus to NewFeature. Change the value of the display-priority attribute of the panel element to 7. This positions the New Feature tab last in the navigation frame displayed with the Change Flow notebook. Save and close the file. Provide the text for the new page by editing the config_locale.properties file for your locale. Copy the Order Status panel section of the file to the bottom of the file and change the keys and values so that it displays the text for your new page.

Finally, make the feature configurable by adding the ifEnabled tag in the JSP page.

## Adding supported features

- Campaigns
- Customer care
- Product advisor
- E-mail activities
- Auctions




WebSphere Commerce provides many features that you can add to your storefront to create the required experience for your customers. Most features require minor JSP customization using WebSphere Commerce Developer. This slide provides a few examples. You can find detailed information on these features and many other in the WebSphere Commerce Information Center.

Campaigns serve to organize your site's marketing efforts. Campaigns are typically created by a Marketing Manager using the WebSphere Commerce Accelerator. They are often associated with a certain set of objectives, which typically match or support any marketing campaigns taking place using traditional means. The page designer is responsible for providing the appropriate e-Marketing Spots on the required pages and in the specified locations. The e-Marketing Spots are defined using a generic WebSphere Commerce bean; the EMarketingSpot bean.

The customer care feature provides real-time customer service support by way of a synchronous text interface using the Lotus Sametime server. A customer can enter the site, and click the Link titled Live Chat with Customer Representative in the left panel of the store page to connect to a customer service representative. The two parties can communicate or chat over the Internet.

The Product Advisor is a tool used to create an interactive online product catalog that provides customers with different ways of finding what they want, called shopping metaphors. Customers with little knowledge of a product category can use the Guided Sell metaphor, which guides them toward appropriate products through a series of questions and answers. Those with more knowledge can use the Product Exploration metaphor, which lets them select preferred product features from a list. Once the selection has been narrowed down through either of the methods mentioned previously, customers can use the Product Comparison metaphor to compare similar products side by side.

E-mail activities allow Merchants to deliver news and promotions to customers using e-mail. This enables Merchants, using the WebSphere Commerce Accelerator, to reach customers who might not have visited your site in some time, or to keep regular customers up to date regarding up coming events or products.

WebSphere Commerce provides an auctioning component that lets you sell products to the highest bidder. This component provides an ideal environment for implementing small to moderate-scale auctioning as part of your e-commerce solution, and for conducting auctions simultaneously.

## Summary

- WebSphere Commerce view layer
- Struts framework in WebSphere Commerce
- Working with JavaServer Pages and data beans
- Customization strategies




This presentation introduced you to developing and customizing WebSphere Commerce storefront pages. The view layer overview was followed by an introduction to the use and configuration of Struts in WebSphere Commerce. Next was a summary of working with JavaServer Pages and data beans including best practices for JSP development. The presentation concluded with a discussion of storefront customization strategies including tool-supported customization such as style modification and site flow options in addition to WebSphere Commerce features that can be added to your store pages.

# 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:

mailto:iea@us.ibm.com?subject=Feedback_about_wcs60_DevelopingAndCustomizingStorefrontPages.ppt

This module is also available in PDF format at:
../wcs60_DevelopingAndCustomizingStorefrontPages.pdf




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 WebSphere

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

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

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 or changes in the products or programs 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 (for example, 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.