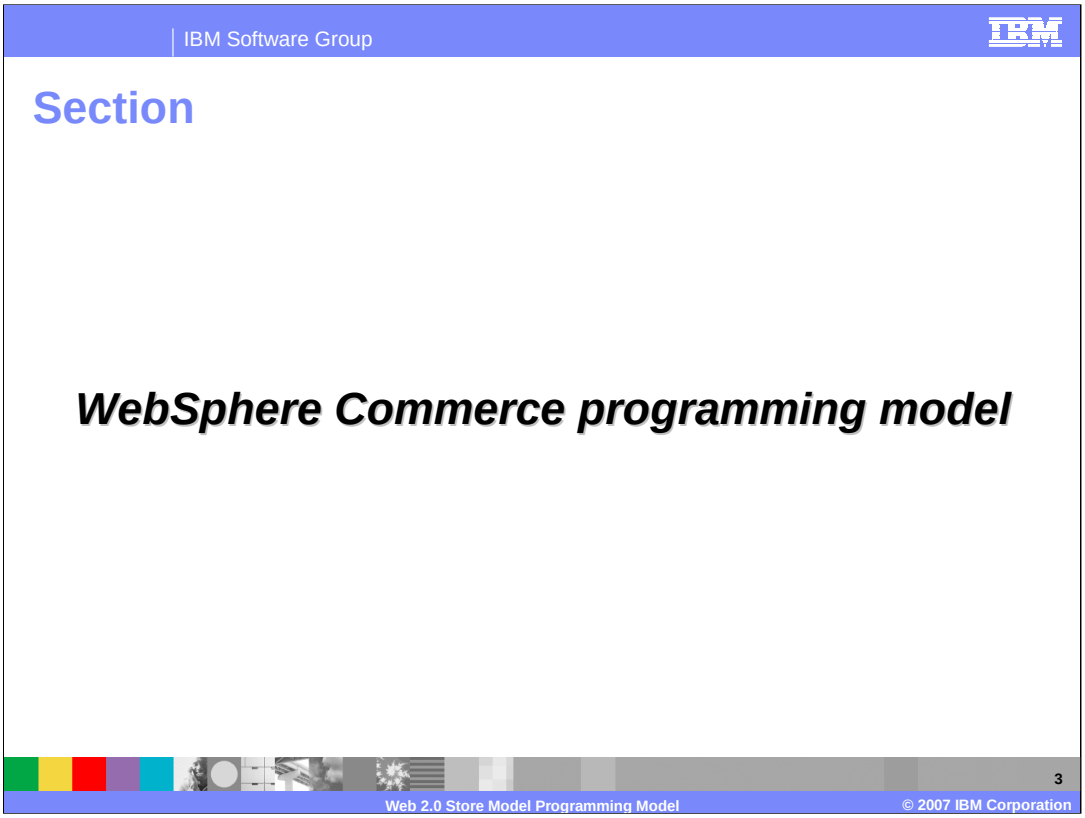
Welcome to the WebSphere Commerce Feature Pack 2 Web 2.0 Store programming model presentation.

Agenda

- WebSphere Commerce programming model
- Web 2.0 Store programming model
- Web 2.0 scenario example

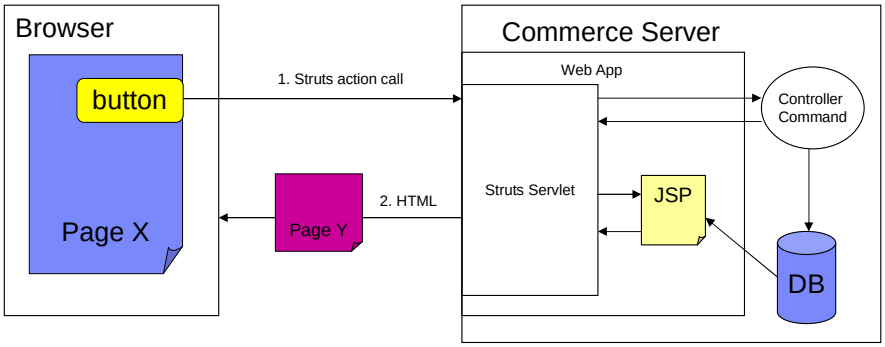
This presentation discusses the WebSphere Commerce programming model, the Web 2.0 Store programming model and the Web 2.0 scenario example.



This section discusses the WebSphere Commerce programming model.

WebSphere Commerce programming model

- Typical storefront interaction with WebSphere Commerce server



This diagram shows the WebSphere Commerce programming model used in all existing starter stores.

WebSphere Commerce programming model (cont.)

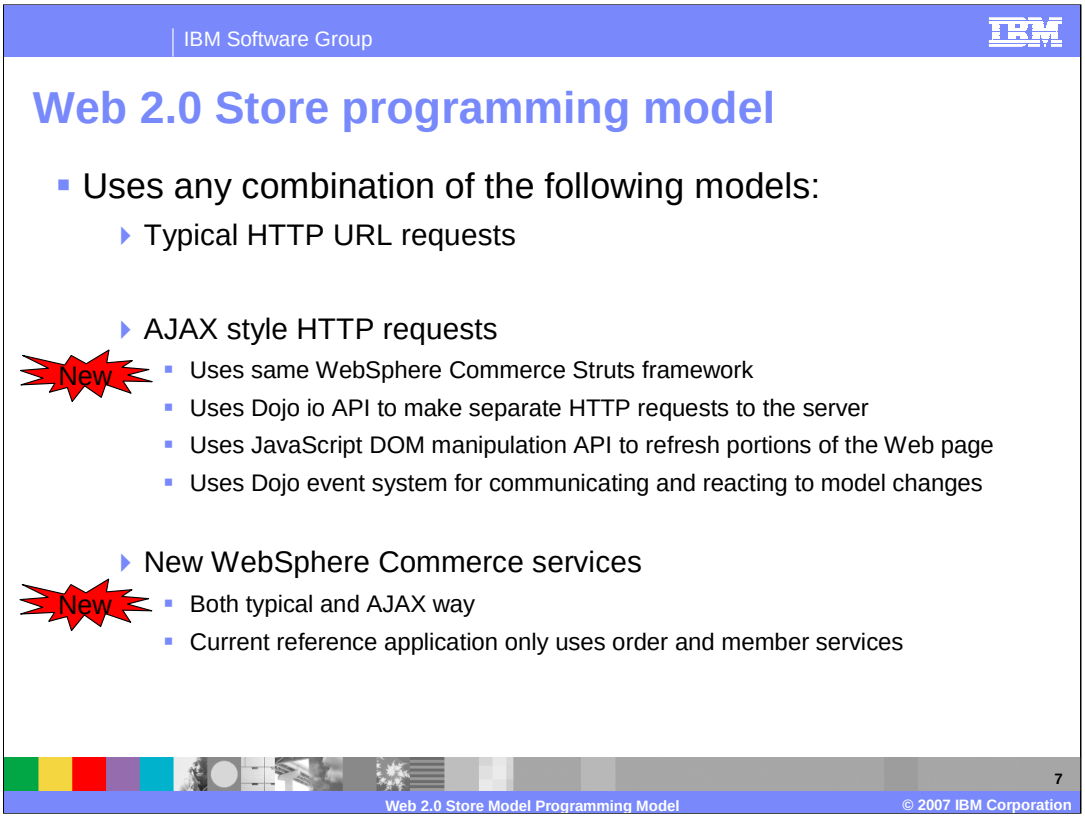
- Make request using HTTP URL
- Run Struts action command
- Re-direct the request to Struts action view
- Compose response using JSP™
- Response HTML replaces current browser request

All browser requests are made from the current browser HTTP URL. The response coming back from that request refreshes the entire browser with the new HTML page contents.

Section

Web 2.0 Store programming model

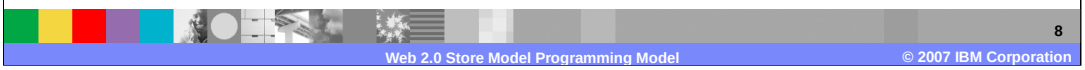
This section discusses the Web 2.0 Store programming model.



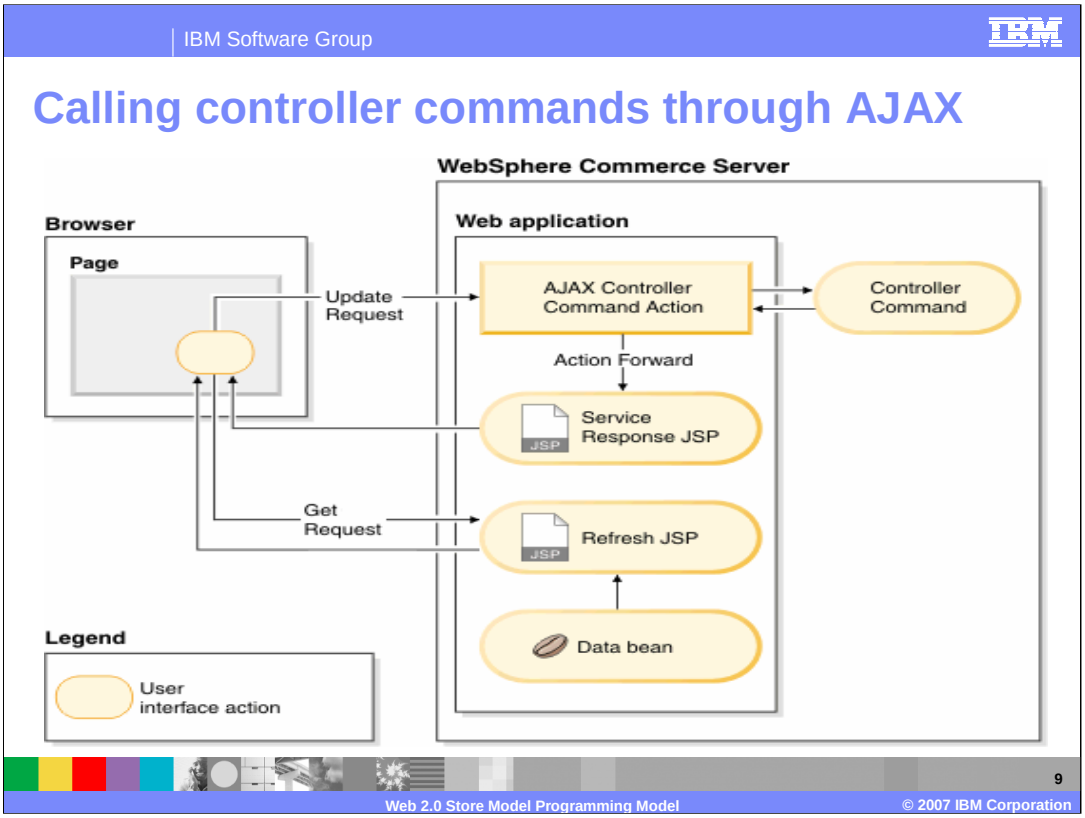
The Web 2.0 Store has several ways of communicating with the server. It can use the traditional model of making HTTP URL requests to the server. It can make AJAX style HTTP requests to the server, or it can call existing controller commands with the new WebSphere Commerce services. WebSphere Commerce services called through the storefront do not go through the overhead of Web services. Instead, they are used in a local binding.

Web 2.0 Store programming model

- JSPs access WebSphere Commerce object model by using:
  - Data beans
  - New: Or services through the new WebSphere Commerce foundation tag library <wcf:getData> tag



JSPs have two ways of retrieving data from the WebSphere Commerce server: using existing data beans and using services through the use of tags in the WebSphere Commerce foundation tag library.



This diagram shows the interactions for making an AJAX call for a controller command. The response is a JSON object containing all the values in the response properties placed by the controller command. JSON stands for JavaScript Object Notation. It is a light weight data interchange format.

Calling controller commands through AJAX (cont.)

- Updating object model requires the Dojo io API to make an AJAX request to call controller commands
- Use new Struts action handler for AJAX update requests
- Define new Struts actions for WebSphere Commerce controller commands called using AJAX
- Calling OrderCopy requires updates to struts-config-ext.xml

- For example,

```
<action parameter="com.ibm.commerce.order.commands.OrderCopyCmd" path="/AjaxOrderCopy" type="com.ibm.commerce.struts.AjaxAction">
        <set-property property="authenticate" value="0:0"/>
        <set-property property="https" value="0:1"/>

</action>
```


When you generate activity within the Web 2.0 Store that requires an update to the object model, the browser's JavaScript code uses the Dojo API to make an AJAX update request to call a controller command. WebSphere Commerce Feature Pack 2 comes with a new Struts action handler for AJAX update requests. You must define new Struts actions for any existing WebSphere Commerce controller command that you want to call with AJAX.

For example, if you want to call the OrderCopy command, add the code in the slide to the struts-config-ext.xml file.

Calling controller commands through AJAX (cont.)

- **AjaxAction** forwards control to JSP files
  - Success
    - AjaxActionResponse
  - Failure
    - AjaxActionErrorResponse
- JSPs generate JSON objects containing response properties information

When the command completes, the new Struts action **AjaxAction** forwards successful transactions to AjaxActionResponse.jsp or failed transactions to AjaxActionErrorResponse.jsp. These JSPs generate JSON objects containing all the information in the response properties.

Calling controller commands through AJAX (cont.)

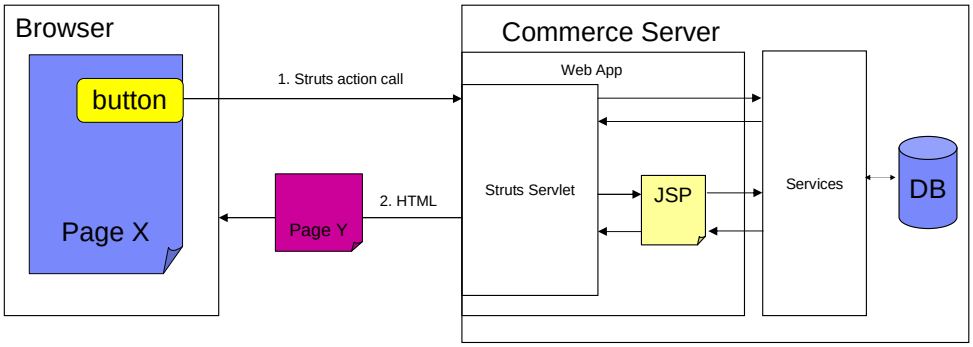
- Dojo event API used to communicate model changes
- Events have listeners registered for portions of JSP page needing to be refreshed
- Listeners make AJAX get requests to update portions of Web page on successful model change
- Get requests are for JSPs that respond with HTML or JSON

The Dojo event API is used to communicate model changes. At page initialization time, any portions of the page that need to be refreshed when the model changes will register listeners for well known events. The listeners make AJAX get requests to retrieve the portions that should be updated on the Web page as a result of a successful model change.

The get requests are for JSPs that may respond with HTML or JSON, whichever is more convenient for the scenario.

Calling services

- Call new WebSphere Commerce services and refresh the whole page.



This diagram shows how to call to the new WebSphere Commerce services. It then shows how to re-direct to a JSP to build the HTML response for the browser request.

Calling services (cont.)

- Client interacts with services using name-value pair paradigm
- Order and member services defined in the struts-config-order-services.xml and struts-config-member-services.xml
- ComponentServiceAction used to create the XML message that interacts with services
- For example,

```
<action parameter="order.addOrderItem" path="/OrderChangeServiceItemAdd"  type="com.ibm.commerce.struts.ComponentServiceAction">
        <set-property property="authenticate" value="0:0"/>
        <set-property property="https" value="0:1"/>
</action>
```

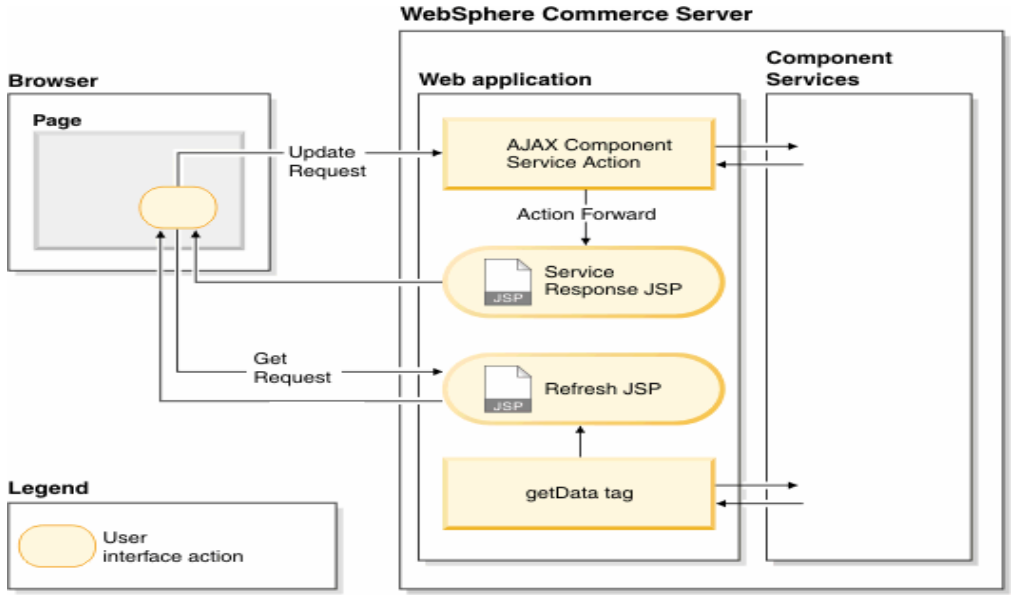

The browser client interacts with WebSphere Commerce services using the name-value pair paradigm. All order and member services are defined in the struts-config-order-services.xml and struts-config-member-services.xml. They are configured to use the ComponentServiceAction which handles the complexity of creating the XML message that interacts with the services. The slide displays an example of one of the Order services defined in the Struts configuration file.

Calling services (cont.)

- Request re-directed to view JSP to construct resulting HTML
- JSP uses <**wcf:getData>** from the WebSphere Commerce foundation tag library with services to retrieve data from the database

Upon success, the request is re-directed to the view JSP to construct the resulting HTML. The JSP uses a tag from the WebSphere Commerce foundation tag library and services to retrieve data from the database.

Calling services through AJAX



This diagram shows the interactions for making an AJAX call for a service. The response is a JSON object containing the response Business Object Document (BOD) message.

Calling services through AJAX (cont.)

- Updating object model requires Dojo io API to make an AJAX request to call the services
- AJAX update requests for services use new AjaxComponentServiceAction Struts action handler
- For example:

```
<action parameter="order.addOrderItem" path="/AjaxOrderChangeServiceItemAdd" type="com.ibm.commerce.struts.AjaxComponentServiceAction">
        <set-property property="authenticate" value="0:0"/>
        <set-property property="https" value="0:1"/>
</action>
```


When you generate activity within the Web 2.0 Store that requires an update to the object model, the browser's JavaScript code uses the Dojo API to make an AJAX update request to call a service. WebSphere Commerce Feature Pack 2 introduces a new Struts action handler for AJAX update requests that use services. An example of configuring the new Struts action is shown in the slide.

Calling services through AJAX (cont.)

- **AjaxComponentServiceAction** forwards control to JSP files
  - Success
    - AjaxActionResponse
  - Failure
    - AjaxActionErrorResponse
- JSPs generate JSON objects containing response properties information

When the command completes, the new Struts action **AjaxComponentServiceAction** forwards successful transactions to AjaxActionResponse.jsp or failed transactions to AjaxActionErrorResponse.jsp. These JSPs generate JSON objects containing all the information in the response properties.

Calling services through AJAX (cont.)

- Dojo event API used to communicate model changes
- Events have listeners registered for portions of JSP page needing to be refreshed
- Listeners make AJAX get requests to update portions of Web page on successful model change
- Get requests are for JSPs that respond with HTML or JSON

Subsequently, the Dojo event API is used to communicate model changes. At page initialization time, any portions of the page that need to be refreshed on model changes will register listeners for well known events. The listeners will make AJAX get requests to retrieve the portions that should be updated on the Web page as a result of a successful model change.

The get requests are for JSPs that may respond with HTML or JSON, whichever is more convenient for the scenario.

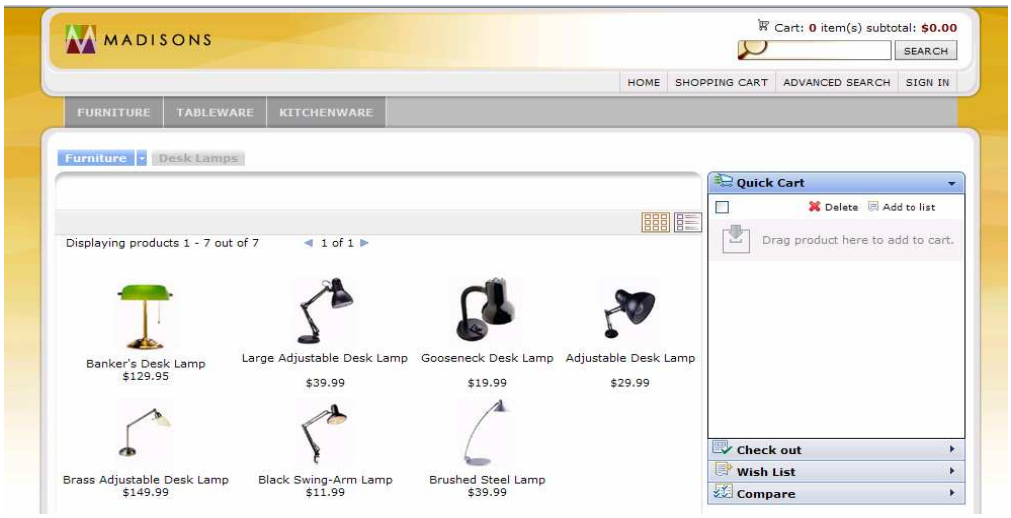
Section

Web 2.0 scenario example

This section discusses the Web 2.0 scenario example.

Web 2.0 scenario example

- Category display page



This screen capture shows the Web 2.0 Store category display page. The next slide discusses the interactions on this page.

Web 2.0 scenario example (cont.)

- Quick cart and mini shop cart register listeners for “add to cart” event
- Dropping item into quick cart forces AJAX call to AjaxOrderChangeServiceItemAdd service
- “add to cart” event triggered upon success
- Registered listeners issue AJAX get request to obtain updated data or HTML

At page load time, the quick cart and mini shop cart register listeners for the “add to cart” event. Dropping an item into the quick cart means that you must call the AjaxOrderChangeServiceItemAdd service using AJAX because you do not want to reload the whole page. Upon success, an “add to cart” event is triggered. At that time, all registered listeners issue an AJAX get request to obtain the updated data or HTML as required.

Summary

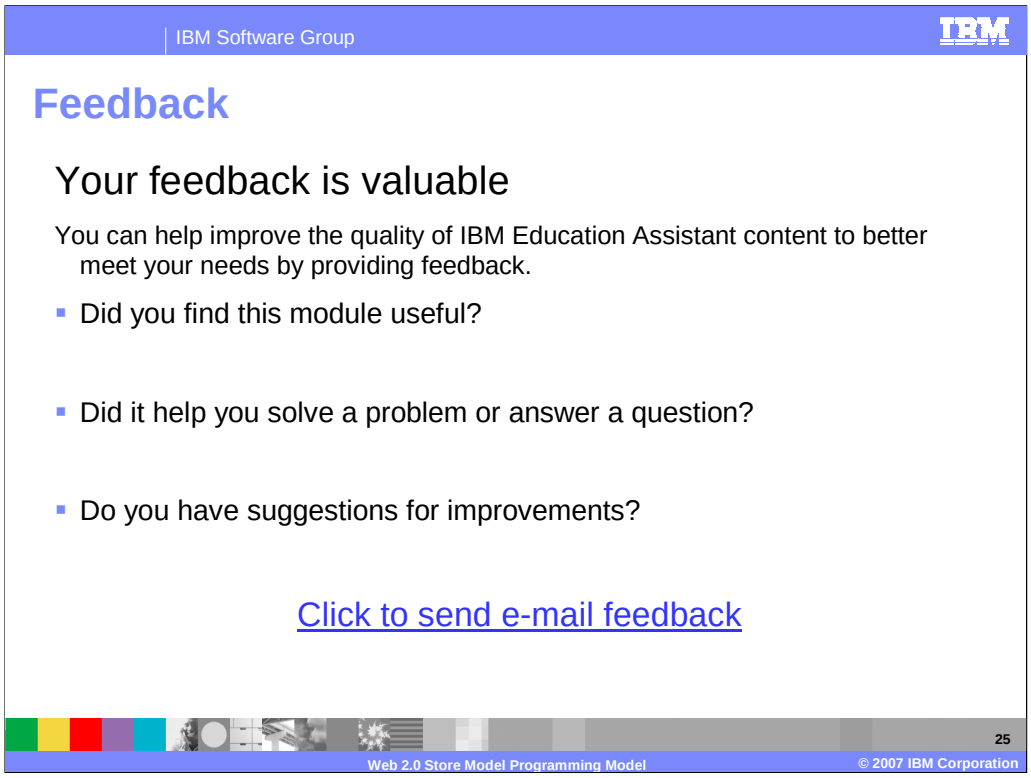
- WebSphere Commerce programming model
- Web 2.0 Store programming model
- Web 2.0 scenario example

This presentation discussed details of the WebSphere Commerce programming model, the Web 2.0 Store programming model and the Web 2.0 scenario example.

References

- What is JSON
  - http://www.json.org/
- WebSphere Commerce services and Web 2.0
  - http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/index.jsp?topic=/com.ibm.commerce.web20storesolution.refapp.doc/tasks/tsm_web20_design.htm

For more information regarding JSON or WebSphere Commerce services and Web 2.0, visit the sites indicated in the presentation.



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 Sametime WebSphere

JavaScript, JSP, Sun, Sun, 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.

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.

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.