This presentation introduces store front customization in feature pack 5 using the new Madisons starter store as a starting point.

Goals

- Understand different customization options available and when to use each
- Understand Web 2.0 customization with Ajax and Dojo
- Understand how to make customizations accessible using Accessible Rich Internet Applications (ARIA) standards
- Understand customization best practices

There are four main goals for this presentation. The first is to understand the different ways you can customize a store front and when to use each. The second goal is to understand the overall Web 2.0 customization method for WebSphere® Commerce and how to work with Ajax and Dojo. The third is to understand how the ARIA standards can be used to make Web 2.0 customizations accessible. The final goal is to understand the best practices for store customization.

Agenda

- Types of customization
- Web 2.0 Programming model
- Adding Web 2.0 to an existing store
- Migration
- Best practices

The first half of this presentation focuses on the types of customization available in the Madisons starter store. Changes specific to feature pack 5 are highlighted in each section. The second half of the presentation addresses several customization topics. These include the Web 2.0 programming model, tips for adding Web 2.0 function to an existing store, migrating from the feature pack 2 Web 2.0 sample store and reviewing best practices for customization.

Section

Types of customization

This section covers types of customization

Types of customization

- Page design
  - Colors, fonts, page layout
- Content
  - Business or personal information displayed on page load
- Web 2.0 content
  - Business or personal information displayed in partial screen refresh
- User interaction
  - Data selection / entry widgets, accessibility

The first section of this presentation reviews the common methods of storefront customization and highlight feature pack 5 specific changes in each.

Page design customization includes changes to colors, fonts, and the overall organization of your page.

Content customization involves adding or changing the information a shopper can access. This includes both business and personal information.

Web 2.0 content customization addresses content customization scenarios where the information is being loaded into a portion of an existing page rather than on initial page load.

User interaction customization covers the shoppers interaction with the Web site including data entry widgets and Web site accessibility.

Page design

- Changes in feature pack 5
  - Page layout using tags and styling using CSS
- Previously
  - <table cellpadding="0" cellspacing="0" width="768" …
- Now
  - or
- Impact
  - Easier to make quick changes
  - Easier to reuse sample code

One of the significant changes in the Madisons starter store that addresses total cost of implementation is the introduction of page layout using div tags. Stylesheets have been used in the past put primarily in conjunction with table tags for layout. The new store has been brought up to date in terms of Web design by leaving all styling decisions, including page layout to the stylesheet.

This makes pages easier to change for demonstration purposes and makes code reuse easier.

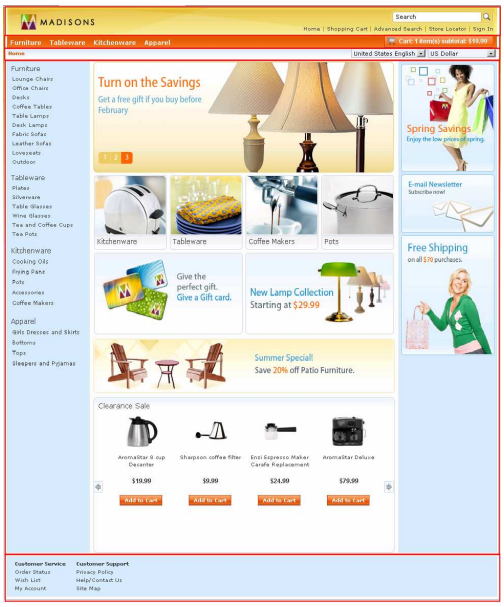
Madisons starter store layout

div id="header"

div id="header_nav"

div id="breadcrumb"

div id="page"



div id="main_content _wrapper"

div id="footer"

This page shows the high-level div structure of the Madisons starter store home page. On the left side, the page div represents the entire area shown. The red boxes around portions of the page show how it is further divided into smaller div areas. These areas are listed on the right side. Each area is again divided into multiple smaller div areas making it easy to move or replace a section of the page.

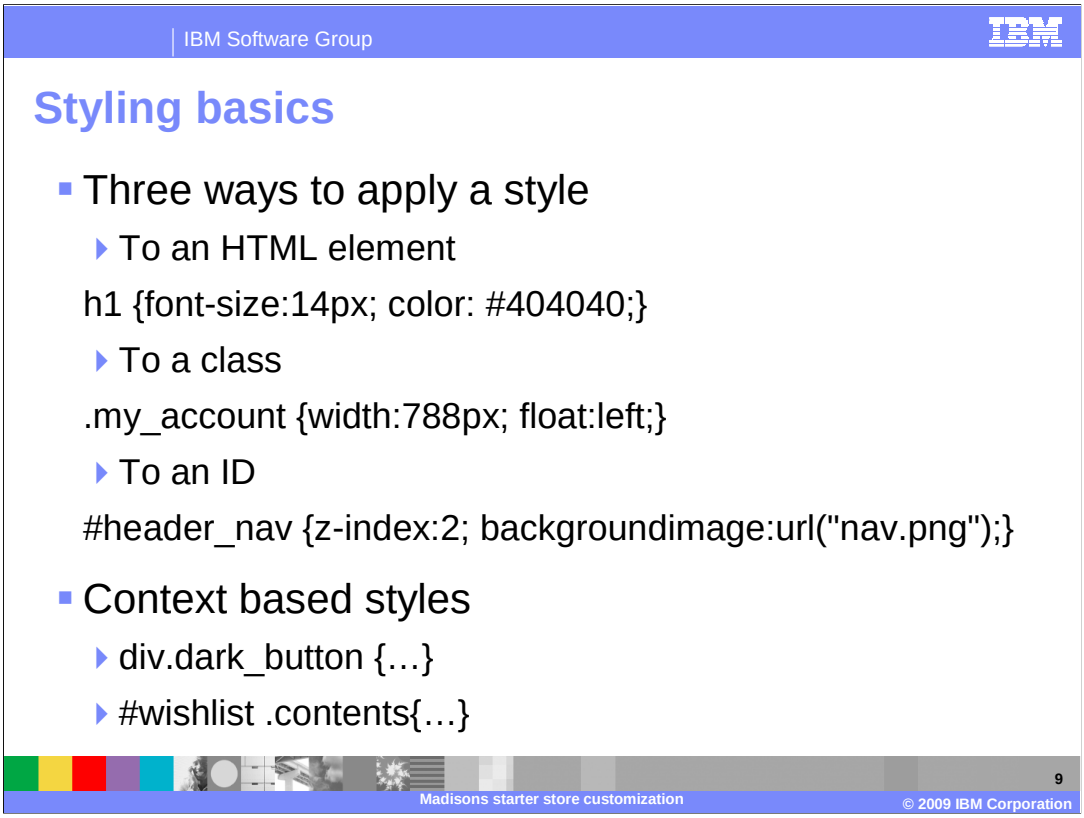
basics

- div
  - A container element representing a portion of a page
- id
  - Uniquely identifies the element within the document
  - Assists in code readability and maintenance
  - Can be used for styling
- class
  - Defines the 'type' of the element
  - An element can belong to more than one class

The underlying structure of the Madisons starter store pages is now a series of nested div tags where it used to be nested tables. This structure is a lot easier to read and maintain than a table based layout. Div tags have no default HTML styling, all styling is applied through CSS. There are two attributes that you will see a lot throughout the Madisons starter store code but which are particularly important when used with div tags.

The first is the ID attribute. An ID uniquely identifies an element within an HTML document. This identification both improves the readability of the code and makes the element easily accessible in JavaScript™ where it can be looked up by ID. IDs can also be used to apply styling to a page. In the Madisons starter store the div IDs contain the name of the file they are defined in. When you look at the source for a page composed of several fragments it is easy to see which file each portion of the page is defined in.

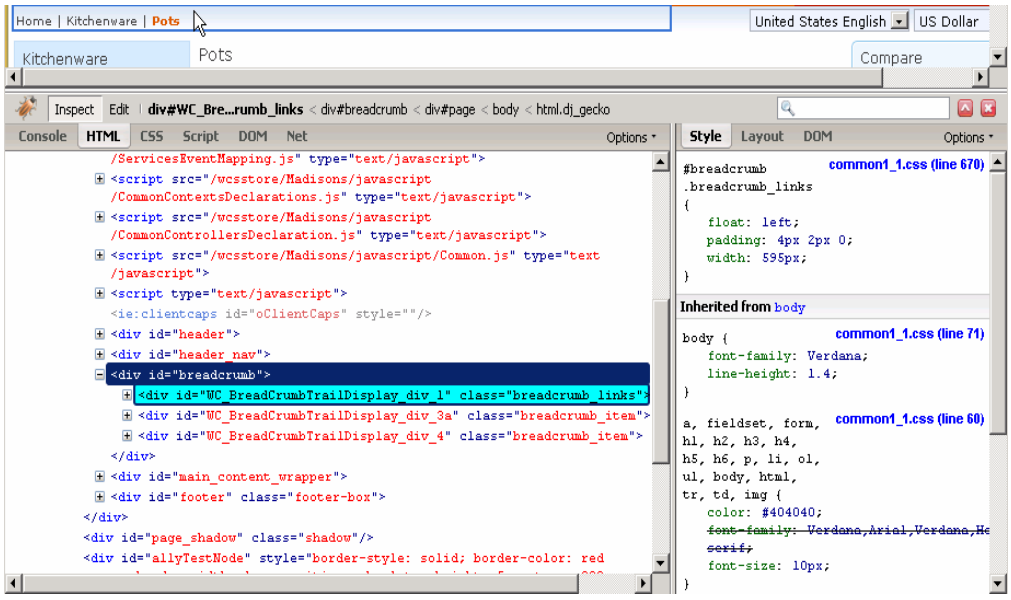
The second attribute to pay attention to is the class attribute. When looking at something as generic as a div tag, the class attribute can give you a hint about the actual usage of the element. For example a class value of 'header' tells you that the div is defining the top of a section and you can expect to find a content area below it. Class names are defined by the page developer and, except for a few restrictions, can be any name. Having meaningful class names can greatly improve the readability of your code. The primary use for classes is styling. The class name is defined in a stylesheet and specifies the styling and layout properties to be applied to the portion of the page contained within the div. More than one class can be applied to a single div.



Each style defined in a style sheet can be thought of as a CSS rule. A single rule applies to specific subset of HTML elements on a page and describes how those elements should look and behave. The way a style is defined determines how it is applied to the page. The first way to apply a style is to a specific HTML element. In this case you add the style declaration following the HTML tag name. The second way to apply a style is to a class. Class names are prefixed by a period. Any time the class name is used by and HTML element the style is applied. The third way to apply a style is to an ID. IDs are prefixed by the pound sign. If the specified ID occurs in the current page, the style is applied.

It is also possible to create context based styling rules by combining HTML element names, class names and IDs. Two examples are shown on the slide. The first style is applied to instances of the dark_button class that occur within a div tag. The second style is applied to instances of the contents class that occurs within an element that has the ID value wishlist.

Working with styles



This screen capture is a picture of the Firebug tool, an add-on for the Firefox browser. Here you see an example of the styling applied to the breadcrumb area of the page. The breadcrumb_links class is applied within the context of the breadcrumb ID. Note that Firebug also shows you which other styles are being applied and which have been overwritten.

Content

- Changes in feature pack 5
  - Services used to retrieve data instead of data beans
- Previously

```
<wcbase:useBean id="orderBean1"
   classname="com.ibm.commerce.order.beans.OrderDataBean" scope="page">
   <c:set property="orderId" value="${WCParam.orderId}" target="${orderBean1}" />
</wcbase:useBean>
```


The Madisons starter store now includes many examples of using the getData tag to invoke a get service rather than the previous useBean tag. The sample shown here is loading data using the order data bean.

Content - continued

- Now

```
<wcf:getData type="com.ibm.commerce.order.facade.datatypes.OrderType" var="order"
   expressionBuilder="findByOrderId">
      <wcf:param name="accessProfile" value="IBM_Details" />
      <wcf:param name="orderId" value="${WCParam.orderId}" />
</wcf:getData>
```

  - Databeans still used where services not available
- Impact
  - Continued adoption of new SOA architecture
  - Access to newer tables that do not have data bean support

On this page, you see a corresponding example of loading data using the get service and findOrderById query.

Data beans are still used where Web services don't exist or where the data beans provide convenience methods the services don't have. The move towards using more services is a continuation of the SOA architecture adoption and in some cases is required to access data from new tables that do not have an associated data bean. Using the getData tag is recommended where possible.

Storefront services

- Available
  - Order
  - Member
  - E-Marketing spot
- Not yet available
  - Catalog
  - Wish list
  - Content spot
  - Promotion

The Madisons starter store makes use of services for accessing order, member and e-marketing spot data. Catalog, wish list, content spot and promotion storefront services are not available in feature pack 5.

Web 2.0 content

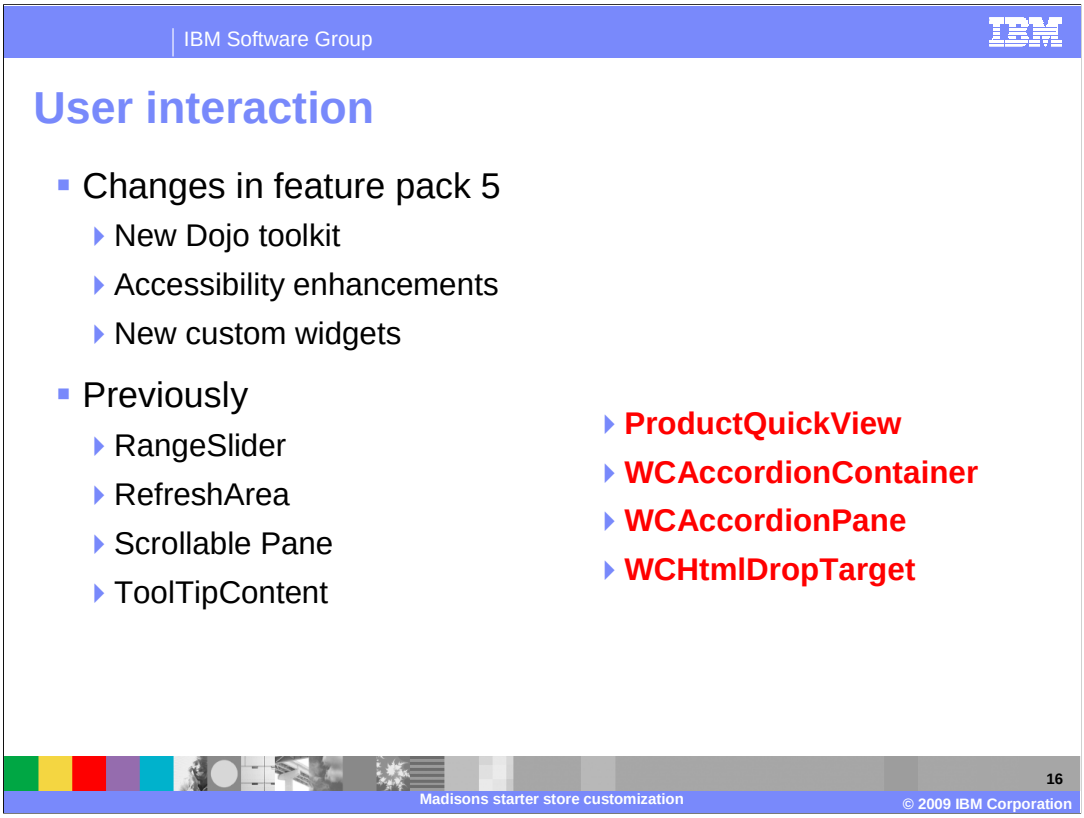
- Changes in feature pack 5
  - Direct JavaScript declarations of Ajax Framework Dojo classes
- Previously
  - wcf:declareService
  - wcf:declareRenderContext
  - wcf:declareRefreshController

Before feature pack 5, the WebSphere Commerce Ajax framework JavaScript classes were instantiated using taglib tags. The wcf tag library still contains the tags for declaring the service, render context and refresh controller classes but they generate JavaScript code compatible with Dojo 0.4.

Web 2.0 content

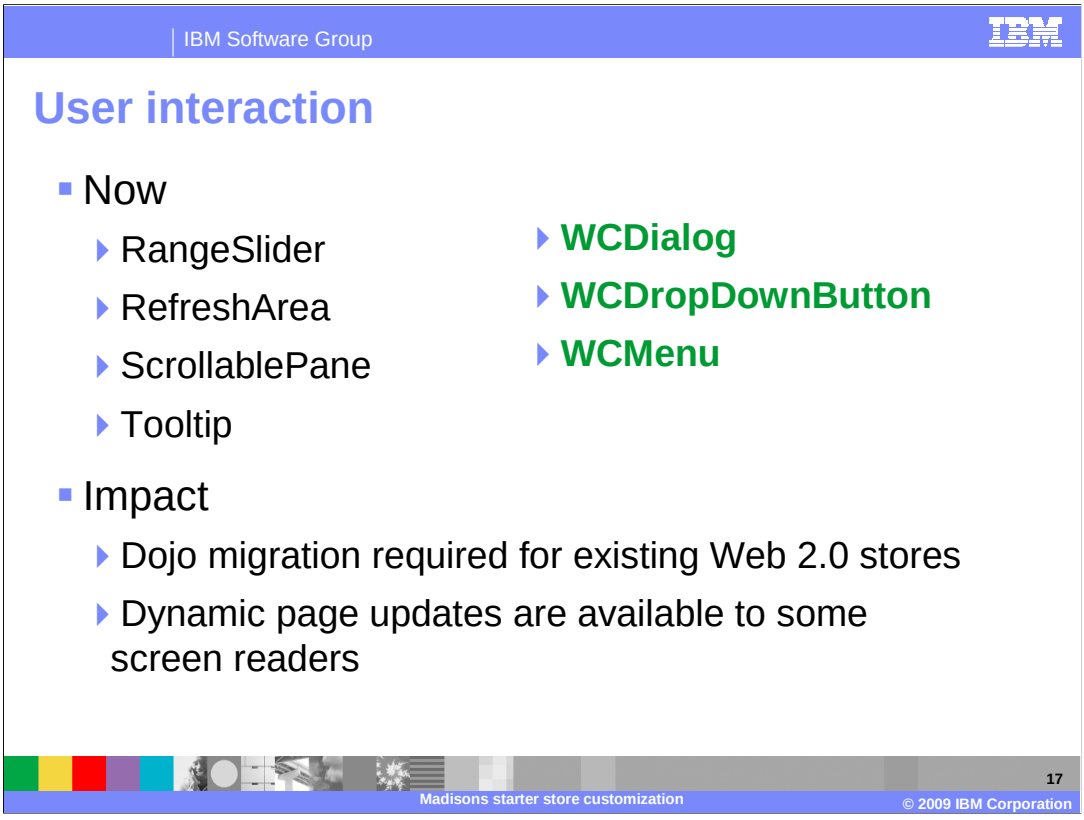
- Now
  - wc.service.declare
  - wc.render.declareContext
  - wc.render.declareRefreshController
- Impact
  - Promotes better understanding of the framework
  - JavaScript can be kept in separate files

Services, render contexts and refresh controllers are now instantiated using JavaScript. This promotes a better understanding of the framework and facilitates separation of JavaScript into separate files.



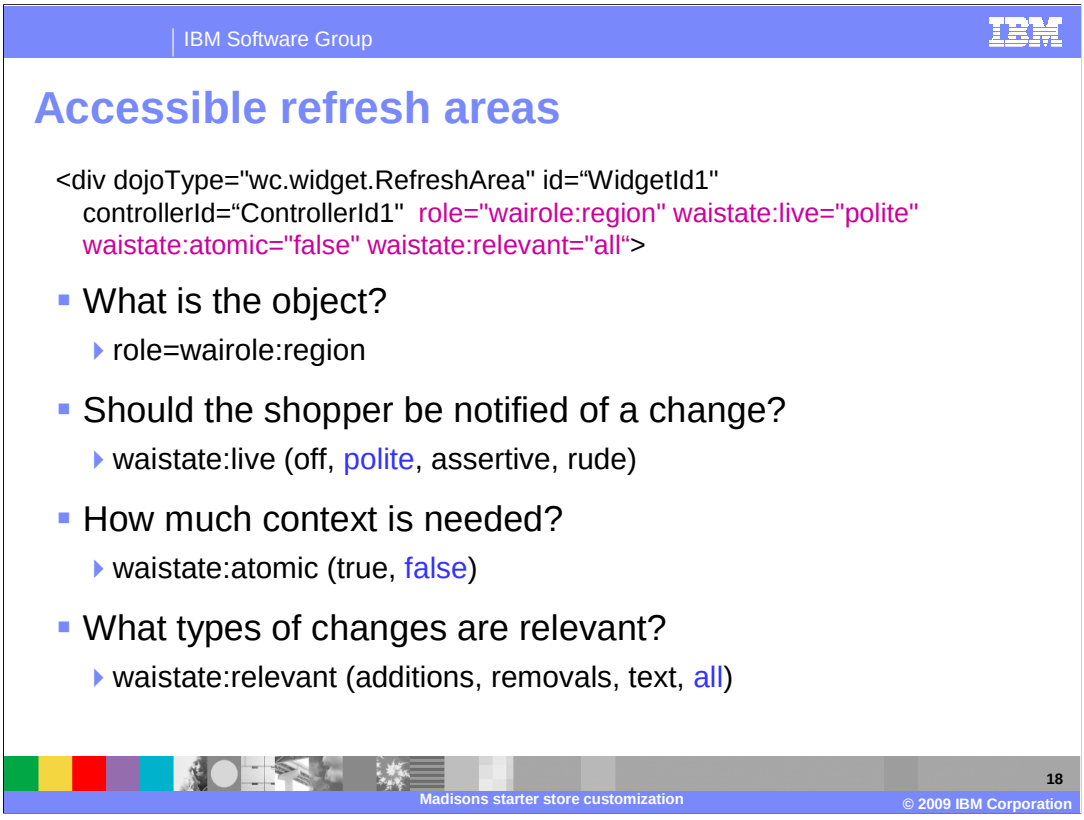
The Dojo toolkit, introduced in feature pack 2, is still used to provide richer JavaScript widgets however it has been upgraded to a newer version. Dojo 1.0.2, included in feature pack 5, provides improved accessibility support for both widgets and dynamically updated page areas. Along with the new version of Dojo, WebSphere Commerce has introduced some new custom widgets that are demonstrated in the Madisons starter store.

Some of the custom WebSphere Commerce widgets introduced in feature pack 2 have been migrated to the new version of Dojo. Others, highlighted in red on the slide, are not included in feature pack 5. These widgets are not used in the Madisons starter store. If you are using one of these widgets and want to migrate to Dojo 1.0.2 the recommendation is to switch to using the corresponding Dojo toolkit widget that the custom widget is based on.



Three new custom Dojo widgets have been added in feature pack 5. They are highlighted in green on the slide. These new widgets are demonstrated in the Madisons starter store.

The most significant impact of these changes to the existing Web 2.0 sample store is the need to migrate Dojo widgets to the new version. Significant changes were made ahead of the 1.0 release of Dojo that impact widget packaging and APIs. You will find that both require statements and widget declarations need to change. Some more migration information is provided near the end of this presentation. The other significant impact of the feature pack 5 changes is the progress towards making rich internet applications accessible to screen readers. With traditional Web applications a page does change much after it is loaded by the browser. This means a screen reader can read the loaded page and the shopper can understand how to interact with it. The introduction of dynamic page updates and partial page refreshes makes Web pages unusable to screen readers since they are unable to detect any changes after initial page load. By implementing the Accessible Rich Internet Applications (ARIA) specification, Dojo 1.0.2 provides accessibility support for partial page updates in newer screen readers.



At the top of this slide is a declaration of a refresh area. The highlighted attributes are those associated with providing accessibility support for the refresh area. The attributes provide information to the screen reader about how it should handle dynamically updated content.

The first attribute, role, answers the question “what is the object?”. The ARIA specification calls dynamically updated areas **live regions** so the value of role is **wairole:region**. The second attribute, waistate:live, answers the question “how should a user be notified of a change to a live region?”. There are four possible values that can be used to answer this question. The Madisons starter store uses the value **polite.** This means the shopper is notified of the change as soon as they have completed any in progress UI tasks. The third attribute, waistate:atomic, answers the question “how much context is needed when reading the change?”. A value of true means the entire live region is read and a value of false means just the changes are read. Finally, the waistate:relevant attribute answers the question “what types of changes are relevant to the shopper?”. There are four possible values for this attribute but the Madisons starter store chooses to declare all changes are relevant.

Accessible widgets

- Add accessibility attributes to your template
  - What is the object?
    - wairole (dialog, alert, button, …)
  - What meaningful properties does the object have?
    - waistate:<property> = “value”
    - waistate:disabled = “true”
    - waistate:pressed = “false”
    - waistate:labelledby = “myLabel”
- Update states as they change

Dojo 1.0.2 defines accessibility attributes for all widgets. If you are defining custom Dojo widgets for your store you need to consider what accessibility attributes to include. The attributes are added to your widget template. You can begin by considering two high level questions. The first is “what is the object?”. The purpose of the widget, or its role, is defined by the wairole attribute. You can use one of the many predefined roles or define one of your own. Dialog, alert and button are some common widget roles.

The second question to consider is “what meaningful properties does the object have?”. You define widget state using the attribute format waistate:<propertyName> = “propertyValue”. Some examples of widget states include disabled, pressed and labelledBy. As the shopper interacts with your custom widget you can use JavaScript to update the values of the states as they change.

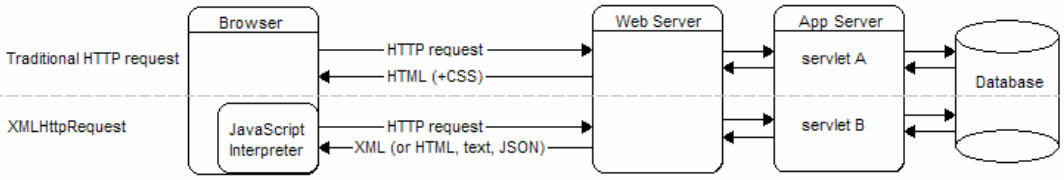
Section

Web 2.0 programming model

This section covers the Web 2.0 programming model for WebSphere Commerce.

Anatomy of an Ajax request

- Differences from HTTP request
  - How the call is initiated: from the JavaScript interpreter rather than the browser
  - How the response is built: XML, HTML, text or JSON.
  - How response is handled: callback to JavaScript function



You might notice that this does not look like a typical WebSphere Commerce diagram and it is not. There is nothing WebSphere Commerce specific about this picture. Before looking at how the WebSphere Commerce Ajax framework works, it is important to understand what is going on when you make an Ajax request. This diagram highlights three key differences to keep in mind as you add Ajax functionality to a Web site.

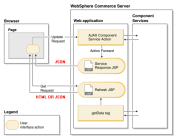
The first is that an Ajax request is initiated by the JavaScript interpreter within the browser rather than by the browser itself as a regular page request is. A JavaScript object named XMLHttpRequest is instantiated and initialized with the properties for a given request. The request itself is a standard HTTP request. It looks the same to the Web server as a request coming directly from the browser.

This is where you see the second main difference. You do not want the application server to regenerate the entire page so you need a way to differentiate an Ajax request from a full page request. You can do this by specifying the return type of the request to be something other than HTML, such as JSON. Or, you can direct the request to a servlet that knows what response is expected. The response to an Ajax request can be a variety of data formats including XML, HTML, JSON or even plain text. There needs to be an understanding between the client and the server about what format is being returned.

The final key difference between browser-initiated HTTP requests and Ajax requests is how the response from the server is handled. Since the request was initiated by the JavaScript interpreter, that is where the response is received using a callback function that was specified when the XMLHttpRequest object was initialized. The callback function is responsible for performing post-processing on the response, such as updating an area of the current page.

One very important thing to remember when designing Ajax requests for your Web site: JavaScript interpreters are single threaded. If you add large amounts of client side JavaScript processing, it will block the shopper from performing other UI actions that require JavaScript until the processing completes.

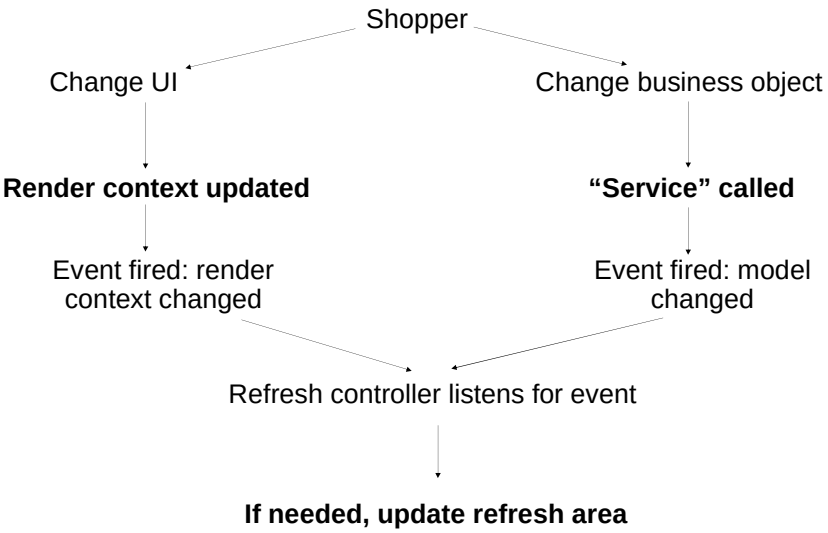
WebSphere Commerce Ajax framework



Now that you know how Ajax requests are structured, it is time to look at the WebSphere Commerce Ajax framework. Frameworks can make working with Ajax requests easier, but they do not change the underlying implementation. The WebSphere Commerce Ajax framework is not the only way to make an Ajax request, but it does provide some advantages that might be useful depending on your requirements.

The flow in this diagram begins with an update request from the browser. Although not shown explicitly here, this request comes from the JavaScript interpreter. Using the Struts configuration file, the Ajax request is mapped to an Ajax-specific Struts action. The Struts action is able to invoke an existing WebSphere Commerce command or service to update the server. Instead of the update forwarding on a view that re-generates the full page, the Struts action forwards to a generic response Java Server Pages (JSP) that generates a JSON response from the command or service response properties. This JSON response typically does not provide enough data for the browser to be updated in response to the change made on the server. In order to update the browser, a second Ajax request is made known as a Get request. The Get request calls a JSP that is specifically designed to return exactly the data the client expects. This can be an HTML fragment representing the portion of the page that changed, or it can be JSON formatted data that the browser will then process.

Ajax framework flow



This diagram shows how the Ajax framework manages page updates. There are two different scenarios. On the left side, the shopper requests a change to the UI, such as paging through a list of items, or clicking the browser forward or back button. This action causes the render context to be updated. A render context update might include an Ajax call to update the server. It also causes a Dojo “render context changed” event to be fired. The event is heard by all refresh controllers that specify the same render context. Each refresh controller will notify the refresh areas it manages and they will be updated, if needed. A refresh area update involves another Ajax call.

If the shopper updates a business object, such as adding an item to their cart, the right path is taken. The change to the business object causes the JavaScript service object’s invoke method to be called. The service launches an Ajax update to the server and then fires a model changed event. Model changed events are received by all refresh controllers. Once again the refresh controllers notify their registered refresh areas and any affected areas are updated by making another Ajax call to get updated data to display.

Ajax requests using Dojo

- Reduces server traffic
- Requires a deeper understanding of Dojo and JavaScript

```
dojo.xhrPost({
    url: this.url,
    handleAs: "json-comment-filtered",
    form: formNode,
    content: parameters,
    load: function(serviceResponse, ioArgs) {
        // Process the response here
    },
    error: function(errObj,ioArgs) {
        var messages = dojo.i18n.getLocalization("wc", "common");
        console.debug("Warning: communication error while making the service call");
    }
});
```


The WebSphere Commerce Ajax framework provides a very modular way of managing Ajax requests. The one side effect of this is an increase in server traffic when making updates. If you recall from the last slide, two Ajax calls are required to refresh a page as a result of a business object update. If the required page updates are returned as a response to the initial update request, only one Ajax call is required. It is possible to bypass the WebSphere Commerce Ajax framework and use the Dojo API directly to make Ajax calls. Using this method requires a deeper understanding of both Dojo and JavaScript. You will need to process the response data yourself to make any necessary page updates. You will also need to keep track of which page elements need to be updated, since you will not have the benefit of the Dojo event system notifying interested refresh areas.

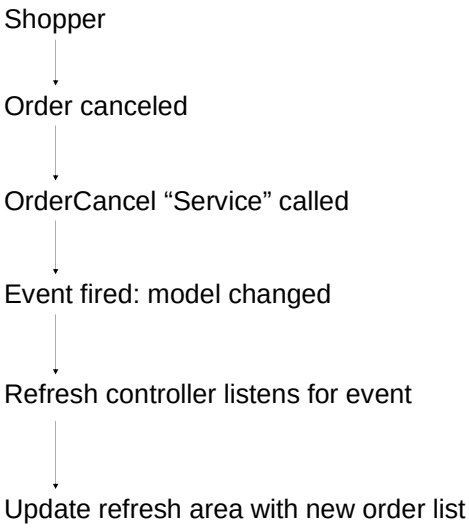
On the bottom half of the screen you can see the Dojo API for making an Ajax call using POST. There is a similar method for making GET requests.

Customization example

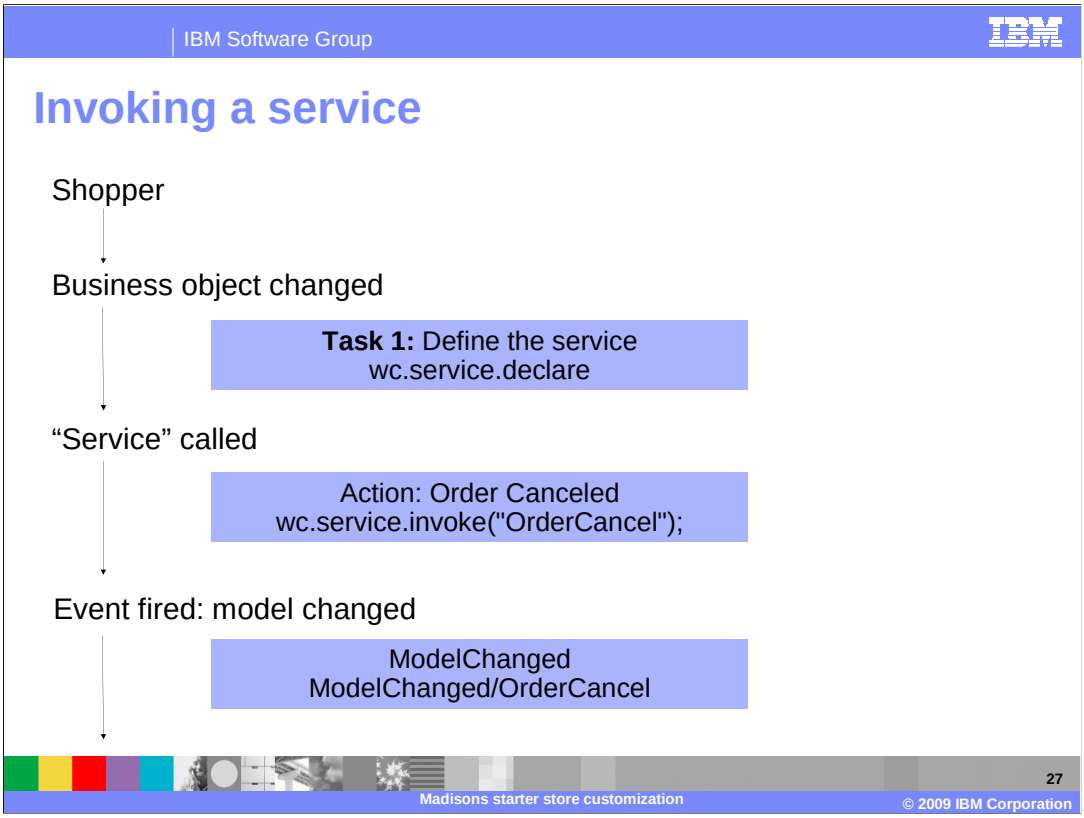
- New requirement: A shopper can cancel an order
- Page design: When an order is canceled it disappears from the order history and a confirmation message is displayed on the screen

Now that you have seen the basics of how the WebSphere Commerce Ajax framework works, it is time to look at an example of how to use it to add a new Ajax function to a page. The new requirement is to support registered shoppers cancelling an order. The cancellation is processed as an Ajax request so that the order disappears from the shopper's list of orders and a confirmation message is displayed on the screen.

Ajax framework in action



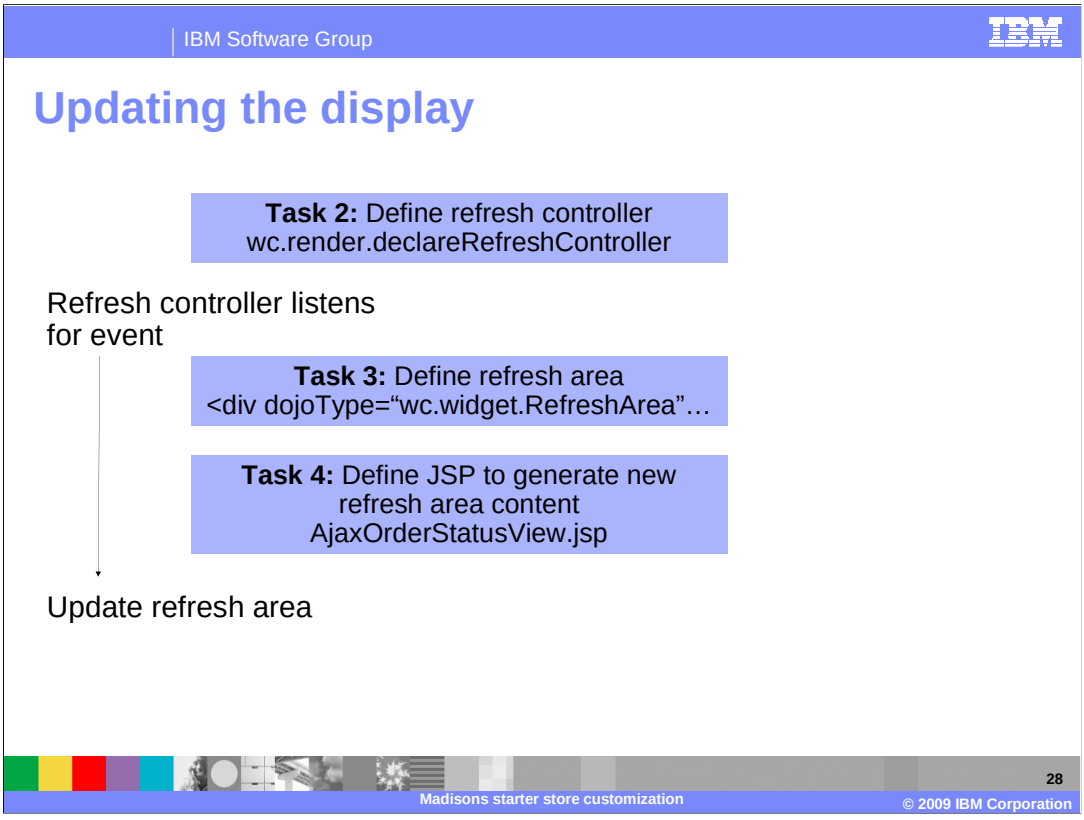
The customization example follows the right side of the Ajax framework flow discussed earlier. When the shopper cancels the order, a service is called to invoke the existing OrderCancel command using Ajax. Once the update has completed, a model changed event is fired and is received by the refresh controller. The refresh controller then notifies the refresh area responsible for updating the order list, and another Ajax call is made to get the new order list.



The next two slides break the customization down into more detail and look at the main tasks required to complete the change. This slide focuses on the first step, which is calling the service to update the business object on the server. When a business object is changed by a shopper, and you want to update the server using Ajax, you set up the UI to call the JavaScript function wc.service.invoke() with the name of a predefined service. The first task in the customization is to define the service using the wc.service.declare API. As part of defining the service, you also need to update the Struts configuration extension file to map the service name to an actual WebSphere Commerce command of service by using the Struts AjaxAction.

When the shopper clicks on the order cancel button, the OrderCancel service you define is invoked. Once it returns, two Dojo events are fired. The first is a general model changed event and the second specifies the action ID representing the specific change. In this case, the action ID is OrderCancel, which you define when you declare the service.

At this point, the server is being updated when the shopper clicks on the order cancel button, but the change is not reflected in the UI.



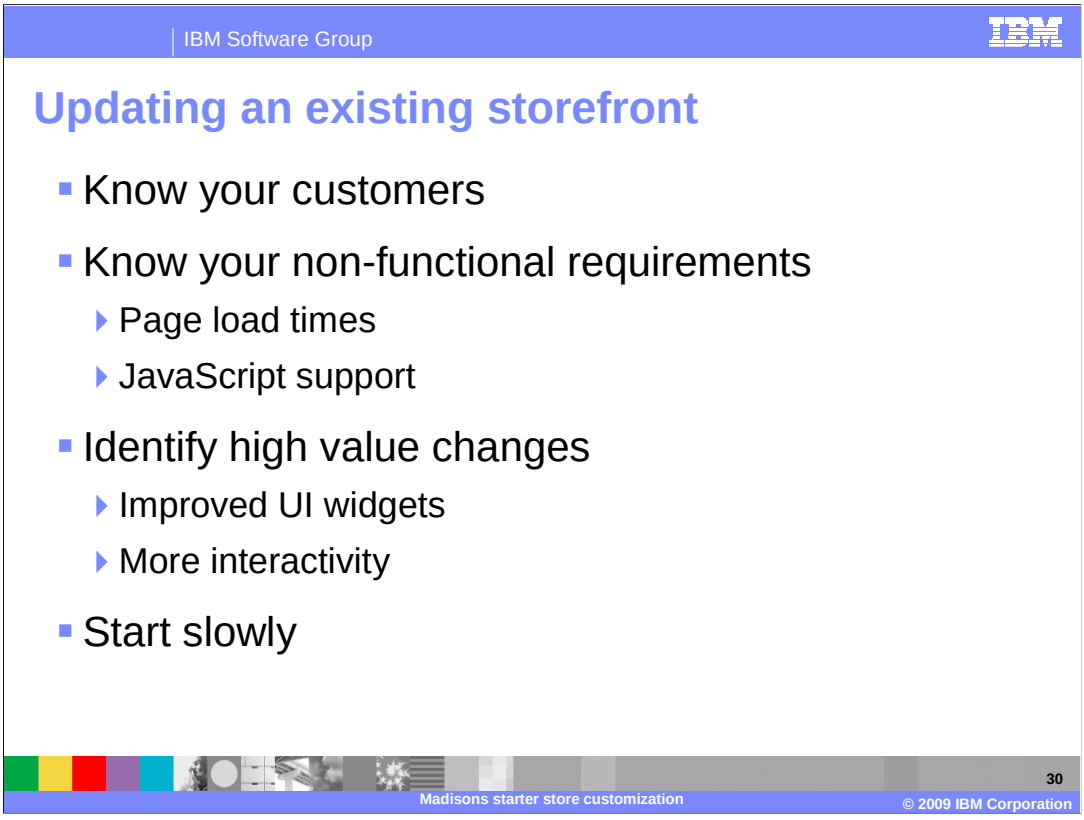
The last slide left off with the OrderCancel service firing a model changed event. In order to listen for this event and take action, you need to define a refresh controller. You do this using the wc.render.declareRefreshController API. As part of declaring a refresh controller, you can specify what happens when model changed events are received. In your new refresh controller, you want to update the refresh area containing the order list when an OrderCancel event is received. This brings you to the next task, creating a refresh area. The refresh area is the portion of the screen you want to redraw when an order is canceled. You do this by wrapping the area in a tag and specifying the attribute dojoType=wc.widget.RefreshArea. When the refresh controller receives the OrderCancel event, it will tell the refresh area to update, triggering an Ajax call to get the modified contents of the area. Your final task is to create the JSP that will render the changed portion of the page. You also need to add this new view to the Struts configuration extension file.

There are a few other sub tasks, such as adding access control, needed to complete this customization. You can find these outlined in the lab StoreCustomization2 or in the Information Center .

Section

Adding Web 2.0 to an existing store

This section covers some tips for adding Web 2.0 functionality to an existing store.



An important consideration when adding Web 2.0 features to an existing store is how your customers will react. Once they have learned how to use your Web site, they will have acquired a comfort level with using it. How much are you asking them to re-learn with the new features? Make sure adequate information is provided and consider providing an option to disable the new features and use the traditional version of your site.

Understand what your non-functional requirements are. Adding Web 2.0 functionality is not free. It increases page download time and can put more load on the client machine. Are the new features worth the extra wait? Web 2.0 functions are also useless without JavaScript enabled. Do you need to support shoppers who have JavaScript disabled?

Examine your existing page flow and identify areas that will most benefit from Web 2.0 capability. Start there. It might be that replacing a standard HTML widget with a Dojo widget makes it easier for the shopper to enter valid information. If you want to improve interactivity, look for a page that is reloaded multiple times as the shopper interacts with it, such as paging or adding information to a form. Concentrating Web 2.0 functions in high value areas provides a better tradeoff between cost and functionality.

If you are working with Ajax and Dojo for the first time, start slowly. Make sure you understand how each new Ajax call affects the page before you add the next one. This can simplify debugging and help to avoid unexpected side effects. Debugging JavaScript is a challenge. Plan accordingly.

Tips for adding Web 2.0 function

- Reuse existing assets where possible
  - Plan Ajax updates to use existing commands or services
  - Use JSP fragments to contain page sections that change
- Configure Dojo using djConfig
  - Parser settings
    - parseOnLoad
  - Debug settings
    - isDebug
    - debugAtAllCosts

The WebSphere Commerce Ajax framework is designed to make it easy to call your existing commands and services using Ajax. Where possible, replace an existing update call with an Ajax update rather than splitting or combining functions so that a new command or service needs to be created. Also, consider creating a JSP fragment for the portion of you page that is going to change. This can avoid code duplication if you are providing both a Web 2.0 and Web 1.0 flow option.

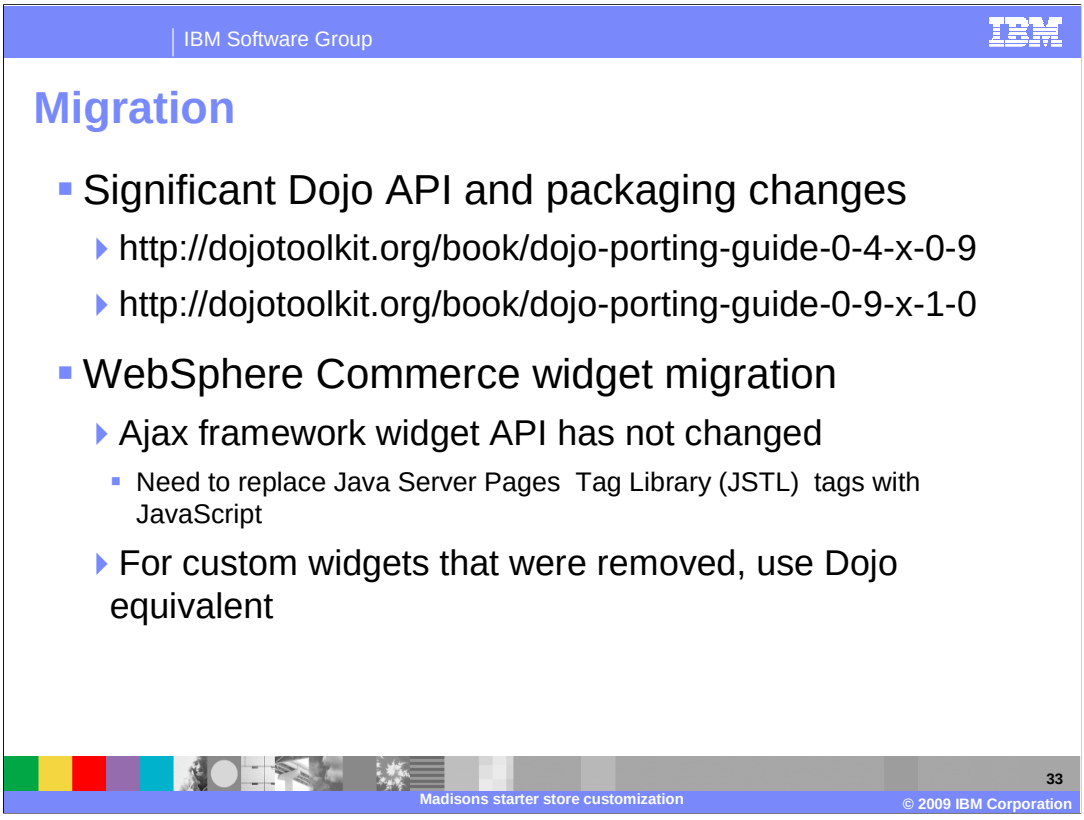
When you are including the Dojo library for the first time you will need to decide what values you want to use for the configuration settings. The first configuration parameters to consider is parseOnLoad which tells Dojo whether it should parse each page it loads looking for the DojoType attribute in an HTML tag. In the Madisons starter store, parseOnLoad is set to false. This results in better performance but it means if you forget to manually invoke the parser for a widget Dojo will not recognize it.

The next thing to consider is the debug settings. The isDebug parameter, when set to true, displays a debug area at the bottom of your browser if you are using Firebug Lite. This gives you access to the Firebug console messages and provides an area where you can type console commands. The debugAtAllCosts parameter provides greater accuracy in locating errors within Dojo widgets. If you are extending or creating new Dojo widgets you might want to turn this on during development. The Web 2.0 sample store sets both debug parameters to false by default.

Section

Migration

This section covers some tips for migrating a store based on the feature pack 2 Web 2.0 sample store to feature pack 5.



There is no official migration path between the Web 2.0 sample store in feature pack 2 and the Madisons starter store in feature pack 5. If you have made use of Dojo functionality beyond that used in the Web 2.0 sample store, you should read the Dojo porting guides available on the Dojo Web site. There are significant changes between the 0.4 and 0.9 releases.

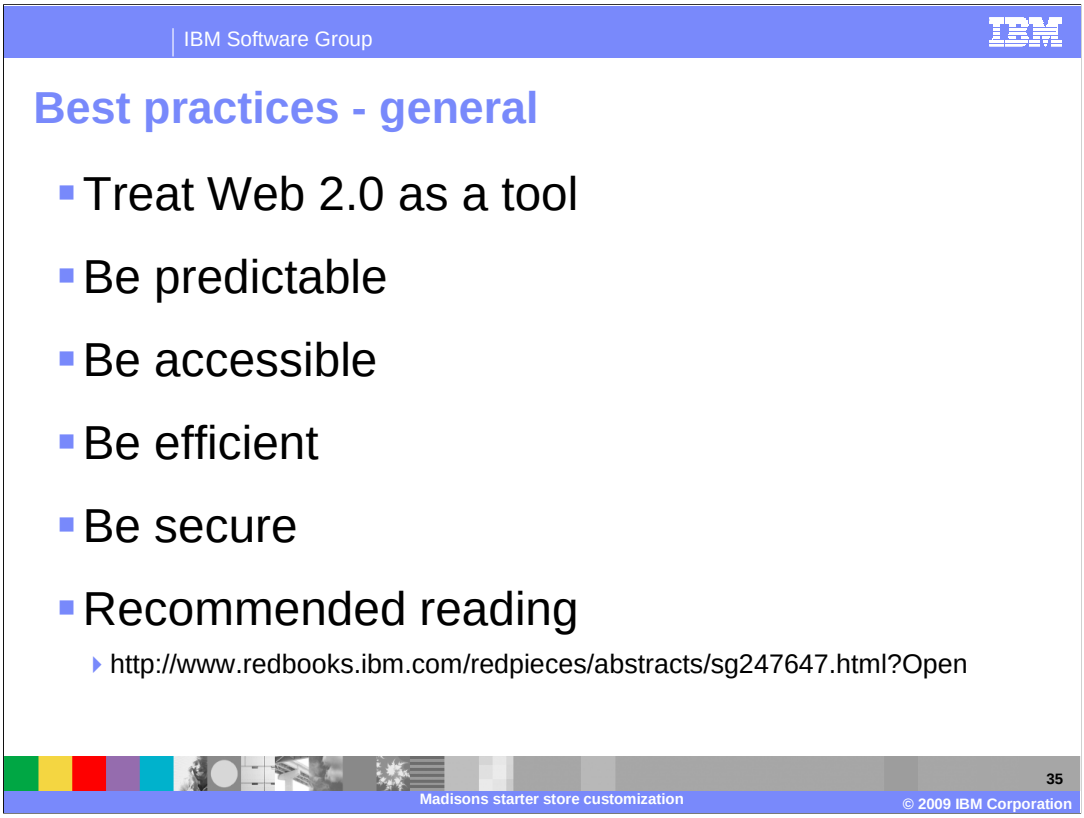
Custom WebSphere Commerce widgets have been migrated to the new version of Dojo, but some APIs have changed. You will need to update your storefront pages to work with the new APIs. Some custom widgets from feature pack 2 have been discontinued and are not used in the Madisons starter store. If you want to continue using the discontinued widgets, you can switch to the Dojo version of the widget that the custom widget is based on.

The WebSphere Commerce Ajax framework APIs have not changed from feature pack 2, so less effort is need to migrate refresh areas. The one change you do need to make is to switch from using the JSTL tags for object creation to the corresponding JavaScript APIs. To simplify this process, you can copy the generated JavaScript out of your storefront page and replace the tag with the JavaScript code. To take the migration one step further, you can use this opportunity to create a separate JavaScript file for the definition of services, render contexts and refresh controllers.

Section

Best practices

This section covers some best practices for Web 2.0 application development in WebSphere Commerce.



There is little doubt that Ajax and Web 2.0 have changed the face of the internet in the last five years. If you are introducing Web 2.0 technologies into your store for the first time, it is very important to resist the temptation to add every type of feature all at once. Take the time to understand where your site will most benefit from improved interactivity or more usable interface widgets. Used as one tool among many, Web 2.0 functionality can provide some very powerful benefits for your users. Consider that some of the highest impact applications of Ajax might barely be noticeable to the shopper except to reduce the interruptions to their shopping experience.

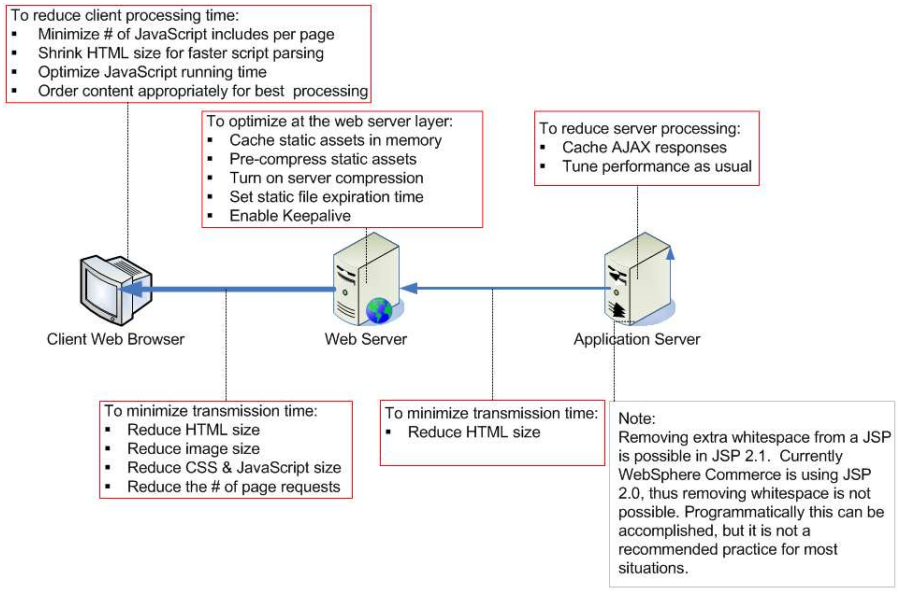
Rather than looking at a detailed list of rules for creating a Web 2.0 applications, it is helpful to consider best practices as principles that should guide your design, development and testing choices.

The first is be predictable. This might sound boring, but there is nothing more frustrating to a user than a Web site that does not behave the way they expect. This includes providing support for browser forward and back buttons on pages updated by Ajax requests. It also means designing interactive page updates that do not disorient the shopper.

Next is be accessible. Many Web accessibility best practices have been around for years and apply to Web 2.0 applications just as much as they did to Web 1.0 applications. On top of basics like keyboard accessibility and alt tags come the new Web accessibility initiatives from the W3C. The ARIA specification provides a way of specifying context on a page beyond what HTML tag names can provide. It also introduces a method of making dynamic page updates visible to assistive technologies. Adding accessibility support as you go is a lot less work than retrofitting an application once it is complete.

Next, be efficient. Some specific performance guidelines are given on the next slide, but in general, you need to consider the cost of Web 2.0 functions and spend your network and JavaScript processing time wisely. The fewer times you contact the server, the smaller the amount of data downloaded to the browser, the faster your application will run. The more dynamic functionality you want to provide, the more you have to be efficient in how you do

Best practices - performance



This picture comes from the RedBook on best practices for Web 2.0 store development. There is an excellent chapter on performance optimizations that contains more detail and tips than are covered here. The purpose of including this picture is to motivate you to create a Web application that performs well by considering many performance aspects, from the client's Web browser through to the application server.

Summary

- Madisons starter store has implemented several best practices
- WebSphere Commerce Ajax framework provides modular approach to asynchronous reads and updates

The Madisons starter store has implemented many best practices for store development that were outlined in the first half of this presentation. The second half of the presentation focused on a review of the WebSphere Commerce Ajax framework and adding Web 2.0 functionality to a store.

References

- Madisons starter store
  - https://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/index.jsp?topic=/com.ibm.commerce.madisons-starterstore.doc/concepts/csmmadisonintro.htm
- ARIA
  - http://esw.w3.org/topic/PF/ARIA/BestPractices/LiveRegion
  - http://www.w3.org/TR/2007/WD-aria-state-20070601/
    - Section 4.1

This slide contains some useful references for store development in feature pack 5.

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_WCS6005_StoreCustomization.ppt

This module is also available in PDF format at: ../WCS6005_StoreCustomization.pdf

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

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

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

Other company, product, or 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 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.

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.

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.