Store customization 2

What this exercise is about

In this tutorial, you will make use of the WebSphere Commerce Ajax framework to call an existing command using Ajax. Your requirement is to add an OrderCancel button to the Order Status page to allow registered shoppers to cancel an order they've placed. Once the order has been canceled it should disappear from the order list without requiring a page refresh.

What you should be able to do

After completing this exercise, you should be able to:

- Define and invoke an Ajax service
- Define a refresh controller
- Use the RefreshArea widget on a page
- Create a JSP that generates the response to an Ajax data request

Introduction

The following naming conventions are used in the exercises:

Reference Variable	Description
<WCDE_INSTALL_DIR>	WebSphere Commerce Developer installation directory
<WCDE_HOST>	Host name for WebSphere Commerce Developer
<LAB_DIR>	Directory of the lab files

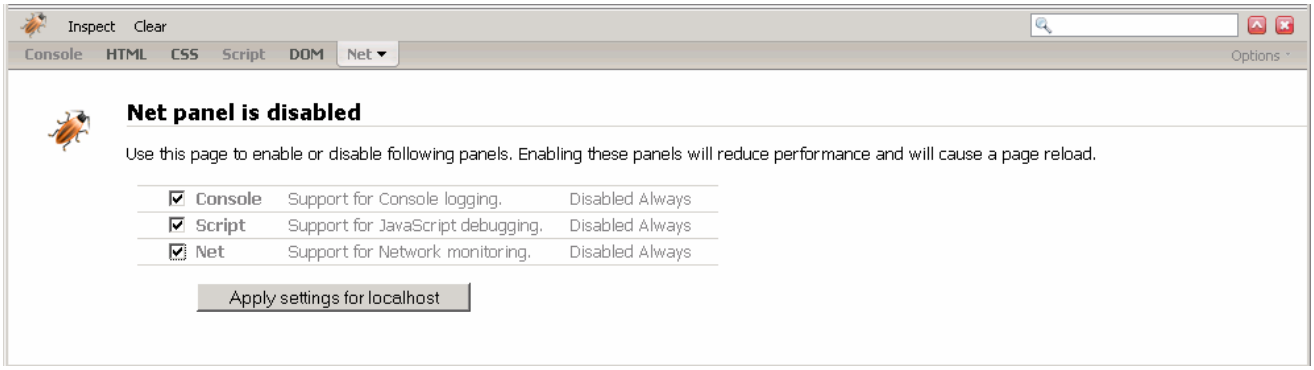
The .zip file provided with this lab contains all the new and modified files necessary to complete this lab. There is also a snippets file provided which contains just the text to be typed in. In a few cases, such as access control policies, you will be instructed to use the file provided. For the majority of the lab, you should type in the code provided. If you find this is too much typing or need assistance solving a problem, the solution files and snippets file are available.

Requirements

Before beginning this lab, ensure you have:

- Installed WebSphere Commerce Developer 6.0
- Installed fix pack 7, fix LI74128
- Installed Feature Pack 4 and fix IZ41934
- Installed Feature Pack 5
- Enabled following features :
    - management-center
    - madisons-starterstore

- Published the Madisons starter store
- Reviewed WCS6005_MadisonsStarterStoreCustomization to familiarize yourself with the WebSphere Commerce Ajax framework
- Downloaded the lab files
- Installed and enabled Firebug



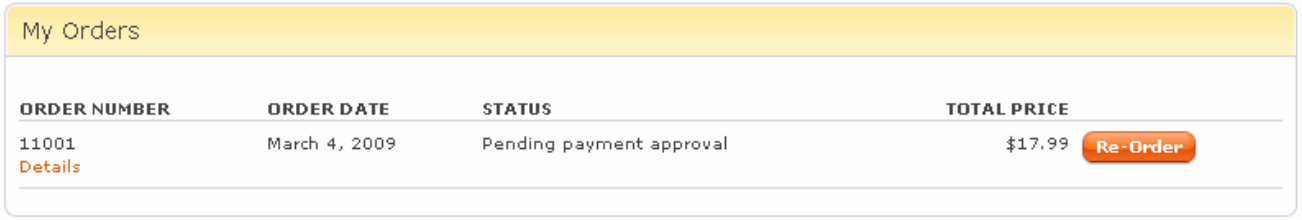
Part 1: Add the Order Cancel button

The first step in this lab is to locate the file that displays the Re-Order button and extend it to include an Order Cancel button. You will start by using Firebug to identify which file to modify.

____ 1. Launch the Madisons starter store **http://<WCDE_HOST>/webapp/wcs/stores/servlet/Madisons/index.jsp** and register a new user. The Order Status page is only available for registered customers.

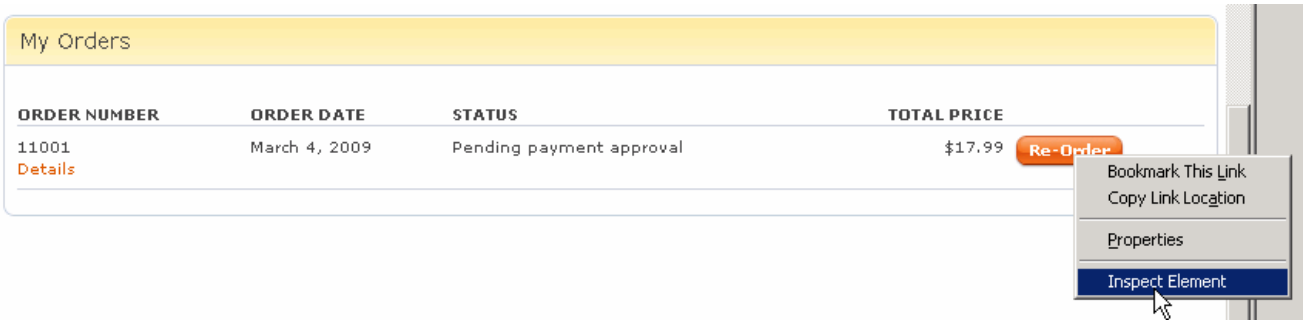
____ 2. Add any product to your cart and complete an order. This will give you some test data when you view the Order Status page.

____ 3. Click on **Order Status** in the page header. You should see a page similar to the screen capture below which shows the order you just placed. You are now ready to begin customization.

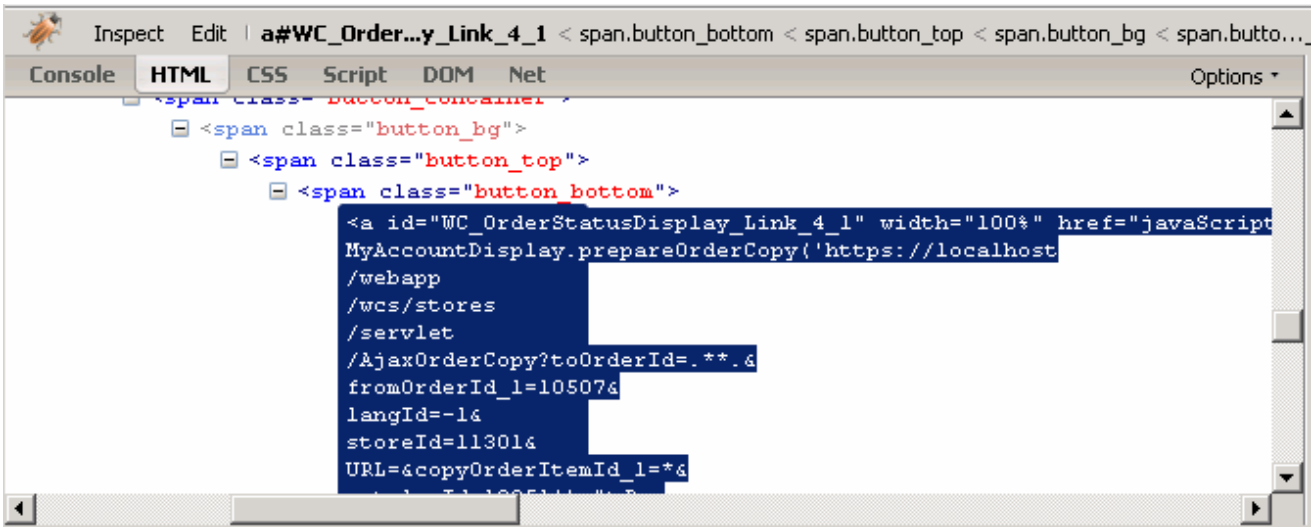


____ 4. Locate the file you need to customize.

__ a. Right click the Re-Order button and select **Inspect Element.** This will open Firebug to the location in the HTML where the button is defined.



__ b. Notice the value of the ID attribute, `WC_OrderStatusTableDisplay_Link_4_1`. This tells you that the button is defined in the file **OrderStatusTableDisplay.jsp.** You can use this technique throughout the Madisons starter store to easily locate the file where a page component is defined.



____ 5. Navigate to **Madisons > Snippets > Order > Cart** and open **OrderStatusTableDisplay.jsp**. Search for `WC_OrderStatusDisplay_Link_4` to find the section of the file that defines the Re-Order button. Do not include the `_1` in your search or you will not get any results. This second digit is automatically generated as you will see by viewing the JSP.

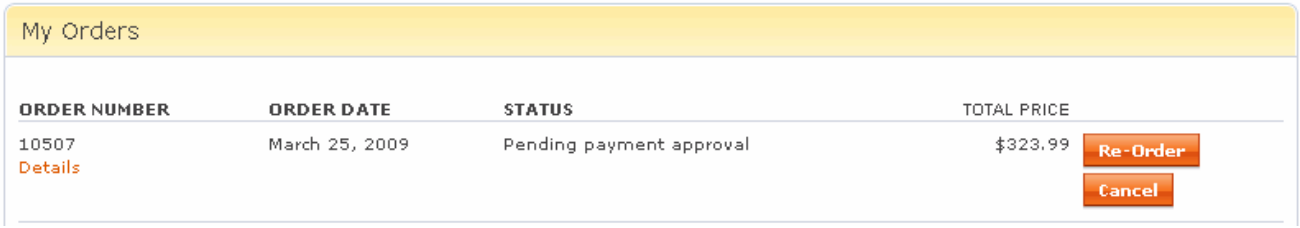
Tip: You can click an attribute name to select it in Firebug and right click to copy it.

____ 6. Add the Order Cancel button.

__ a. Add the following code under the line that defines the Re-Order button. You will notice that this button definition is a little simpler that the Re-Order button. The code that automatically generates element ID values has been removed and the id values simplified. What remains are the style class names that will provide the look and feel for the button.

```
<p>
  <span class="primary_button button_fit" >
    <span class="button_container">
      <span class="button_bg">
        <span class="button_top">
          <span class="button_bottom">
            <a href="javaScript:prepareOrderCancel('<c:out
            value='${OrderCancelUrl}'/>');" width="100%"
            id="Ext_Lab2_Link_1">Cancel</a>
          </span>
        </span>
      </span>
    </span>
  </span>
</p>
```

__ b. Save your changes and reload the Order Status page in your browser. You should see a new Cancel button below the Re-Order button.



__ c. Define the `OrderCancelURL` variable used above in the call to `prepareOrderCancel()`. Scroll up in the file slightly until you see the definition of the `OrderCopyURL` variable. Add the following code below the `OrderCopyURL` definition:

```
<wcf:url value="AjaxOrderCancel" var="OrderCancelUrl">
      <wcf:param name="orderId"
            value="${order.orderIdentifier.uniqueID}"/>
      <wcf:param name="URL" value=""/>
      <wcf:param name="storeId" value="${WCParam.storeId}"/>
      <wcf:param name="catalogId" value="${WCParam.catalogId}"/>
      <wcf:param name="langId" value="${WCParam.langId}"/>
</wcf:url>
```

The <wcf:url> tag is new in Feature Pack 5. It simplifies URL creation by taking care of which protocol to use (HTTP or HTTPS) and building SEO compliant URLs if needed.

Part 2: Define the service

In Part 1 you added a new button to the Order Status page that calls the JavaScript function prepareOrderCancel. In this part you will define that function and create the Dojo service that the function will call.

____ 1. Create a new directory structure for your custom code

__ a. Open WebSphere Commerce Developer and navigate to the **Stores** project.

__ b. Expand the Stores project directory until you reach the **Madisons** directory as shown below. Right click the **Madisons** folder and select **New > Folder.**

__ c. Create a folder named **Lab2**. All of your custom files will be stored under this directory.

__ d. Right click the **Lab2** folder and select **New > JavaScript file**. Name the file **AjaxOrderStatus.js** and click **Finish**.

____ 2. Add the prepareOrderCancel function. This function accepts the service URL as an input variable then updates and invokes the service. Notice the call to the cursor_wait() function. This function displays a visual cue to the shopper to indicate the service is running. It also prevents the shopper from sending a second request by clicking the button twice. Add the following code to your JavaScript file:

```
function prepareOrderCancel(OrderCancelURL){

      cursor_wait();

      wc.service.getServiceById("OrderCancel").url=OrderCancelURL;
      wc.service.invoke("OrderCancel");
};
```

____ 3. Now it is time to define the service that will be invoked by the `prepareOrderCancel` function you just defined. Notice that the success handler has two main responsibilities beyond logging the `serviceResponse` properties. The first is to display a message to the shopper that their order has successfully been canceled and the second is to return the cursor to its normal state. Add the following code to your JavaScript file:

```
wc.service.declare({
id: "OrderCancel",
      actionId: "OrderCancel",
      url: "AjaxOrderCancel",
      formId: ""

      ,successHandler: function(serviceResponse) {
            MessageHelper.displayStatusMessage("Your order has been
             canceled");
            cursor_clear();
      }

      ,failureHandler: function(serviceResponse) {
            if (serviceResponse.errorMessage) {
                  MessageHelper.displayErrorMessage(
                  serviceResponse.errorMessage);
            }
            else {
                  if (serviceResponse.errorMessageKey) {

                        MessageHelper.displayErrorMessage(
```

```
                                    serviceResponse.errorMessageKey);
                            }
                        }
                        cursor_clear();
                    }
                });
```

____ 4. Navigate to **Madisons > UserArea > AccountSection** and open the file **MyAccountDisplay.jsp**. This is the top level JSP that includes the OrderStatusTableDisplay.jsp snippet. All of the JavaScript includes for the page are at the top of MyAccountDisplay.

__ a. Within MyAccountDisplay.jsp, navigate to the script include area as shown in the screen capture below.

```
<script type="text/javascript" src="<c:out value="${jspStoreImgDir}javascript/CatalogArea/Ca
<script type="text/javascript" src="<c:out value="${jspStoreImgDir}javascript/CatalogArea/Co
<script type="text/javascript" src="<c:out value="${jspStoreImgDir}javascript/UserArea/MyAcc
```

__ b. Add the following JavaScript include:

```
<script type="text/javascript" src="<c:out
value="${jspStoreImgDir}Lab2/AjaxOrderStatus.js"/>"></script>
```

__ c. Save the file.

____ 5. Update the struts configuration file with the new action AjaxOrderCancel.

__ a. Navigate to **Stores > WebContent > WEB-INF** and open the file **struts-config-ext.xml**. Select the **Source** tab.

__ b. Search for **AjaxOrderCopy** to locate the action entry for the Re-Order button. Copy the AjaxOrderCopy definition and paste it at the bottom of the action-mappings section. It should look similar to this, the store id (11301) may be different:

```
<action parameter="com.ibm.commerce.order.commands.OrderCopyCmd"
path="/AjaxOrderCopy" type="com.ibm.commerce.struts.AjaxAction">
      <set-property property="authenticate" value="11301:0,11301:0"/>
      <set-property property="https" value="11301:1,11301:1"/>
</action>
```

__ c. Update the new action with the values shown in blue below. The parameter attribute is changed to call the OrderCancelCmd command and the path attribute is changed to AjaxOrderCancel. The Struts type AjaxAction causes the command to redirect to a predefined JSP when it completes. The JSP formats the response properties as JSON and that becomes the response to the browser Ajax call.

```
<action parameter="com.ibm.commerce.order.commands.OrderCancelCmd"
path="/AjaxOrderCancel" type="com.ibm.commerce.struts.AjaxAction">
      <set-property property="authenticate" value="11301:0,11301:0"/>
      <set-property property="https" value="11301:1,11301:1"/>
</action>
```

__ d. Save the file.

____ 6. Update the access control policies to allow registered shoppers to cancel orders. The current access policies for the store do not allow orders to be canceled once they have been submitted. For the purpose of this lab you will allow OrderCancel to be called on any order. For a full solution additional logic would be needed to provide some restrictions but for simplicity this lab will not provide any additional constraints.

__ a. Stop your test server.

__ b. Copy the file **<LAB_DIR>\OrderCancelAccessControlPolicies.xml** into the directory **<WCDE_INSTALL_DIR>\xml\policies\xml**

__ c. Open a DOS prompt and change to the directory <**WCDE_INSTALL_DIR>\bin**

__ d. Run the command `acpload "<WCDE_INSRALL_DIR>\xml\policies\xml\OrderCancelAccessControlPolicies.xml"` and ensure it completes correctly.

__ e. Restart your test server.

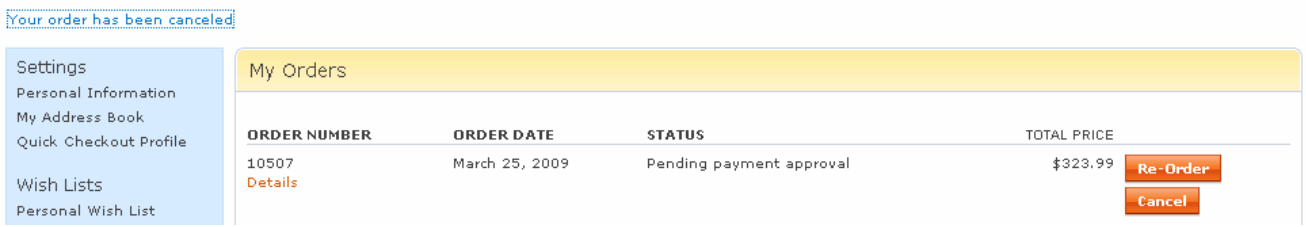
Part 3: Test the service

You should now be able to invoke your service using the Cancel button you added in Part 1. Clicking on the button will initiate an asynchronous call to the server to cancel the order and when the service returns a message will be displayed at the top of the screen. You will still be able to see the order because you have not added any code yet to update the order list.

____ 1. Ensure your test server is started and launch the Madisons starter store.

____ 2. Log into the test account you created earlier and select **Order Status** from the page header. You should see the same order you created in Part 1.

____ 3. Click the Cancel button. You should see a message appear at the top of the screen indicating the order has been canceled. The order will still be visible in the list because you have not added a refresh area yet to update the list. If you reload the page you should see the order has been removed from the list.



Part 4: Create the refresh controller

Now that the cancel service is working it is time to add a new refresh controller to listen for the event generated when an order is canceled. This event will be used to trigger an update of a refresh area. You will create the refresh area in the next part. In this part of the lab, you will define a refresh controller and the methods it will use to evaluate incoming events.

____ 1. Navigate to the Lab2 folder and open the **AjaxOrderStatus.js** file you created in Part 2.

____ 2. Declare a render context. Each refresh controller must have an associated render context when it is defined. Add the following render context definition to the bottom of your JavaScript file.

```
wc.render.declareContext("OrderStatusDisplay_Context",null,"");
```

____ 3. Declare the refresh controller. Notice how the render context is used in the definition of the refresh controller. Create the `modelChangedHandler` function as specified below. When the OrderCancel ActionId is received, the function sets some display properties specific to each refresh area. Once the properties are set, the refresh function is called to update the content in the refresh area. Add the following code to the bottom of your file:

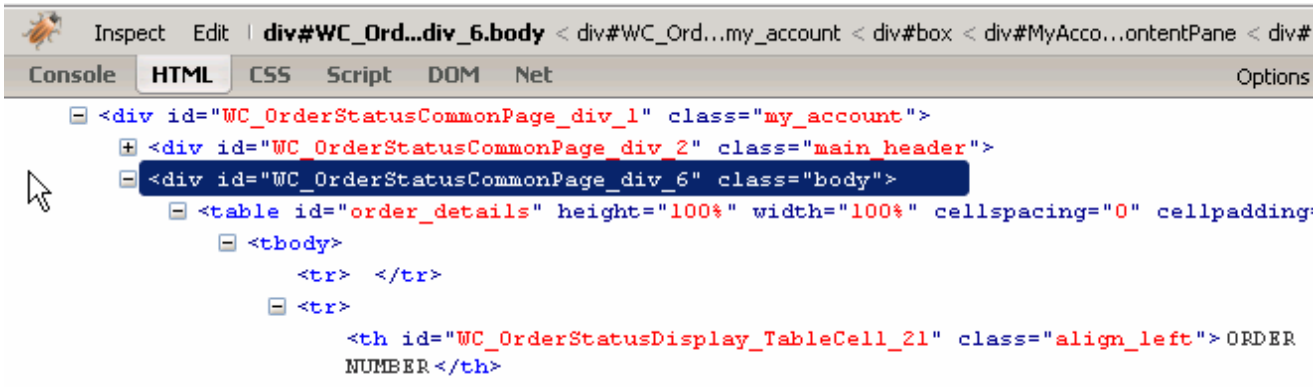
```
wc.render.declareRefreshController({
       id: "OrderStatusDisplay_Controller",
       renderContext:
            wc.render.getContextById("OrderStatusDisplay_Context"),
       url: "",
       formId: ""

       ,modelChangedHandler: function(message, widget) {
              var controller = this;
              var renderContext = this.renderContext;

              if(message.actionId == 'OrderCancel'){
                  if (widget.id ==
                        'Ext_OrderStatusCommonPage_Widget') {
                        wc.render.updateContext(
                        "OrderStatusDisplay_Context",
                        {maLandingPage:"false", allOrders:"true"});
                  }
                  else if (widget.id ==
                        'Ext_MyAccountOrderStatusDisplay_Widget') {
                        wc.render.updateContext(
                        "OrderStatusDisplay_Context",
                        {maLandingPage:"true", allOrders:"false"});
                  }
                  widget.refresh(renderContext.properties);
              }
       }

});
```

____ 4. Save the file.

Part 5: Create the refresh area

Now it is time to add a new refresh area to update the list of orders when one is canceled. A refresh area is a portion of a web page where the content displayed will be affected by a shopper's actions, in this case canceling an order.

____ 1. Create a new refresh area around the order status table. Using Firebug, you can determine that the file OrderStatusTableDisplay.jsp, where you added the cancel button, is contained within another file, **OrderStatusCommonPage.jsp**.



__ a. Navigate to **Madisons > UserArea > ServiceSection > TrackOrderStatusSubsection** and open the file **OrderStatusCommonPage.jsp**.

__ b. Search for "WC_OrderStatusCommonPage_div_6", shown in the above screen capture. This is the ID of the div element that contains the order status table.

__ c. Underneath this line, create a new refresh area to update the order status. Add the following code:

```
<div dojoType="wc.widget.RefreshArea"
id="Ext_OrderStatusCommonPage_Widget"
controllerId="OrderStatusDisplay_Controller"  role="wairole:region"
waistate:live="polite" waistate:atomic="false"
waistate:relevant="all">
```

__ d. Add a matching end tag immediately after the closing `</table>` tag. It should look like this:

```
        </table>
</div>
<br/>
```

__ e. The refresh area you just created makes use of several accessibility attributes to assist screen readers in handling portions of the page that change dynamically. This table reviews the attributes and their meanings.

Accessibility attribute	Meaning
role="wairole:region"	Declares the structure as a "live region". This informs assistive technology that the content within the region can change.
waistate:live="polite"	Specifies when to inform the user that live region content has changed. The value "polite" signals to

	wait until the user is idle.
waistate:atomic=”false”	Signals that the user needs to hear only the portion of the live region that has changed, not the whole area.
waistate:relevant=”all”	Specifies which types of changes need to be heard. This can be limited to additions, removals or text. In this example all changes will be heard.

____ 2. Create a matching refresh area around the order status portion of the My Account page.

__ a. Navigate to **Madisons > UserArea > AccountSection** and open the file **MyAccountCenterLinkDisplay.jsp.** Firebug can be used to determine this is the file to edit.

__ b. Search for "WC_MyAccountCenterLinkDisplay_div_16" (also found using Firebug). This is the ID of the div element that contains the order status table.

__ c. Underneath this line, create a new refresh area to update the order status. Add the following code noting that the id value is different than the previous step:

```
<div dojoType="wc.widget.RefreshArea"
id="Ext_MyAccountOrderStatusDisplay_Widget"
controllerId="OrderStatusDisplay_Controller"  role="wairole:region"
waistate:live="polite" waistate:atomic="false"
waistate:relevant="all">
```

__ d. Add a matching end tag just before the “WC_MyAccountCenterLinkDisplay_div_16” closing tag. It should look like this:

```
        </table>
</div>
<br/>
```

__ e. One additional step is needed for the MyAccount page. Navigate to **Madisons > UserArea > AccountSection** and open the file **MyAccountDisplay.jsp.** This is the same file you edited in Part 2 to add the JavaScript include. Locate the `<script>` block that contains the line `parseWidget("MyAccountCenterLinkDisplay_Widget");` Add the following line below the line you located:

```
parseWidget("Ext_MyAccountOrderStatusDisplay_Widget");
```

This line causes the refresh area to be parsed by the Dojo parser. This step is needed to create an instance of the RefreshArea object. You did not need to perform this step for the order status page because it is loaded as the result of another Ajax call and the HTML returned is automatically parsed by the WebSphere Commerce Ajax framework.

Part 6: Define the JSP that will generate the refresh content

In this part of the lab, you will define the JSP that will generate the HTML fragment used to update the order status list when an order is canceled. You will also register the new View in the Struts extension file and add access control for it.

____ 1. Navigate to **Madisons > Lab2** and create a new file called **AjaxOrderStatusView.jsp**.

____ 2. The HTML generated by this JSP will replace the <table> definition that is inside the two refresh areas you created in Part 5. Add the following code to the new file:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wcbase" %>
<%@ include file="../include/nocache.jspf" %>
<%@ include file="../include/JSTLEnvironmentSetup.jspf" %>

<table id="order_details" cellpadding="0" cellspacing="0" border="0"
width="100%" height="100%">
      <% out.flush(); %>

      <c:import
url="${jspStoreDir}Snippets/Order/Cart/OrderStatusTableDisplay.jsp" >
            <c:param name= "showScheduledOrders" value="false"/>
            <c:param name= "showOrdersAwaitingApproval"
value="false"/>
            <c:param name= "showPONumber" value="false"/>
            <c:param name="maLandingPage"
value="${WCParam.maLandingPage}"/>
            <c:param name="allOrders" value="${WCParam.allOrders}"/>
      </c:import>

      <% out.flush();%>
</table>
```

____ 3. Set the refresh controller URL to call your new page. Rather than hard coding the URL into the refresh controller definition it is advisable to create the URL using the `<wcf:url>` tag so that the appropriate protocol (http or https is used for the request).

__ a. Navigate to **Madisons > Snippets > Order > Cart** and open **OrderStatusTableDisplay.jsp**. This is the page you modified in Part 1 to add the cancel button. Locate the section of the page where you used the `<wcf:url>` tag to define the service URL `AjaxOrderCancel.` Add a definition for the AjaxOrderStatusView URL:

```
<wcf:url value="AjaxOrderStatusView" var="AjaxOrderStatusURL"
type="Ajax">
      <wcf:param name="storeId"   value="${WCParam.storeId}"  />
      <wcf:param name="catalogId" value="${WCParam.catalogId}"/>
      <wcf:param name="langId" value="${langId}" />
</wcf:url>
```

__ b. Update the refresh controller with the new URL when the Cancel button is clicked. Add the following JavaScript statement ahead of the call to prepareOrderCancel() in the Cancel button declaration:

```
CommonControllersDeclarationJS.setControllerURL('OrderStatusDisplay_C
ontroller','<c:out value='${AjaxOrderStatusURL}'/>');
```

__ c. When you are done the link definition should look like this:

```
<a
href="javaScript:CommonControllersDeclarationJS.setControllerURL('Ord
```

```
erStatusDisplay_Controller','<c:out
value='${AjaxOrderStatusURL}'/>');prepareOrderCancel('<c:out
value='${OrderCancelUrl}'/>');" width="100%"
id="Ext_Lab2_Link_1">Cancel</a>
```

____ 4. Update the struts configuration file with the new view AjaxOrderStatusView.

__ a. Navigate to **Stores > WebContent > WEB-INF** and open the file **struts-config-ext.xml**. Select the **Source** tab.

__ b. Within the `<global-forwards>` tag add the following definition:

```
<forward className="com.ibm.commerce.struts.ECActionForward"
name="AjaxOrderStatusView" path="/Lab2/AjaxOrderStatusView.jsp"/>
```

__ c. Within the `<action-mappings>` tag add the following definition:

```
<action path="/AjaxOrderStatusView"
type="com.ibm.commerce.struts.BaseAction"/>
```

__ d. Save the file.

____ 5. Update the access control policies to allow registered shoppers to call the new View.

__ a. Stop your test server.

__ b. Copy the file **<LAB_DIR>\OrderStatusPolicies.xml** into the directory **<WCDE_INSTALL_DIR>\xml\policies\xml**.

__ c. Open a DOS prompt and change to the directory **<WCDE_INSTALL_DIR>\bin**.

__ d. Run the command `acpload "<WCDE_INSTALL_DIR>\xml\policies\xml\OrderStatusPolicies.xml"` and ensure it completes correctly.

__ e. Restart your test server.

Part 7: Test the refresh area

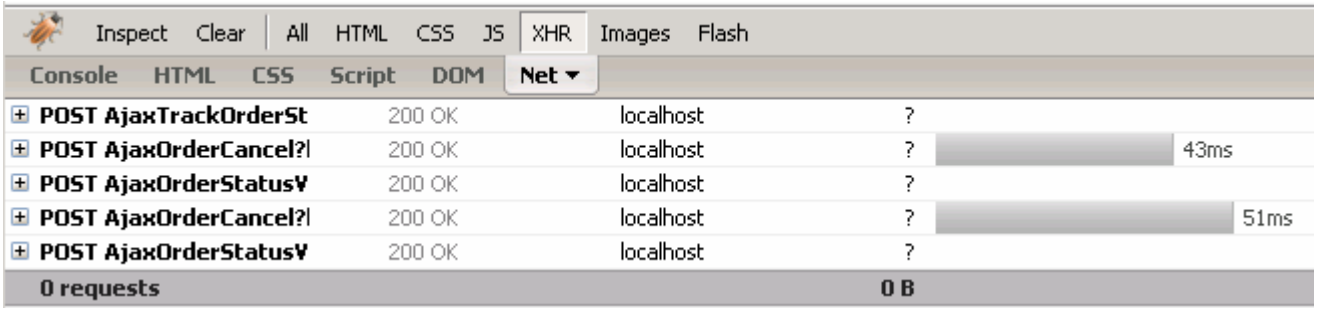
Now that you have defined a refresh area and created the JSP that will generate the refresh content you should see the canceled order disappear from the order list.

____ 1. Launch the Madisons starter store **http://<WCDE_HOST>/webapp/wcs/stores/servlet/Madisons/index.jsp** and log into the account you created earlier. If your **My Account** page shows you have no orders, create additional orders for testing purposes.

Optional: To speed up testing you can retrieve canceled orders by resetting their status in the database. Launch **http://<WCDE_HOST>/webapp/wcs/admin/servlet/db.jsp** and enter the SQL statement `update orders set locked='1',status='M' where status='X';` Refresh the store page and you will see your previously canceled orders.

____ 2. Click the Cancel button. You should see a message appear at the top of the screen indicating the order has been canceled. The canceled order should also disappear from the list of orders.

____ 3. If the order does not disappear from the list, open Firebug to the Console tab and check for any error messages. Messages output by the WebSphere Commerce Ajax framework do not trigger the red x symbol in Firebug so you need to read through the console messages. You can also use the Net tab and XHR filter to look at the Ajax requests. You should see one for the order being canceled (AjaxOrderCancel) and another one for refreshing the order list on the page (AjaxOrderStatusView)



This screen capture shows the XHR view after two orders were canceled.

Part 8: What you did in this exercise

In this tutorial you created an end to end Ajax update using the WebSphere Commerce Ajax framework.

The four main tasks covered in the exercise were:

- Defining and invoking a new Ajax service
- Defining a refresh controller
- Defining a portion of a page as a refresh area
- Creating JSP to generate the HTML for the refresh area