Marketing Customization 2

What this exercise is about

In this tutorial, you will implement a new marketing activity action. Adding a new action is one of many customization options available for the Marketing tool in version 7. This is the second in a series of three labs designed to demonstrate marketing customization. The labs can be completed individually but each will build on concepts and techniques introduced in the previous lab. The recommended starting point for this lab is the lab MarketingCustomization1_newTarget.

The new action you will create allows marketing managers to recommend physical stores to a shopper based on the city they live in. Business users can select which types of stores: regular, expresss or outlet to recommend. This action might be used to cross promote local stores when the shopper is browsing the Web site. In this lab you will also learn how to pass new data types to an e-Marketing spot.

What you should be able to do

After completing this exercise, you should be able to:

- Define and register a marketing element implementation template that includes variables
- Implement a task command to perform an action
- Pass a new data type to an e-Marketing spot
- Add a new action to the Management Center Web activity palette and create a properties view for it
- Display a new e-Marketing spot data type in the Madisons store

Introduction

The following naming conventions are used in the exercises:

Reference Variable	Description
<WCDE_INSTALL_DIR>	WebSphere Commerce Developer installation directory
<WCDE_HOST>	Hostname for WebSphere Commerce Developer
<LAB_FILE_DIR>	Location lab files were extracted to

The MarketingCustomizationLabs.zip file that is provided with this lab contains all the new files necessary to complete this lab. There is also a snippets file provided which contains just the text to be typed in. Unless otherwise stated in the instructions, 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 7.0

Part 1: Define the new action

In this part of the lab, you will plan your new action and collect all the information you need to proceed with the customization. This information includes parameters specified by the business user when setting up the action and the command that provides the business logic to perform the action. Once you have all the information, you create an implementation template. This template links all the information together.

____ 1. Identify what parameters the business user needs to specify to create the action. In this example, the action is to return physical stores that match the selected type: regular, express or outlet. The business user will specify which store type to recommend. This means one variable, `Type`, is needed in the element template. The value of the variable will be stored in the DMELEMENTNVP table when the marketing activity is created.

____ 2. Identify the name of the task command that will perform the action. In this example, you will use the task command `com.mycompany.DisplayStoreTaskCmd`. You will create this task command later in this lab.

____ 3. Define the implementation template for the action using the information gathered in steps 1 and 2.

__ a. The completed implementation template for this action is shown below**.** Notice the format of the Argument element. The value `MARKETING_storeType` indicates that the actual value for the `Type` variable can be found in the DMELEMENTNVP table. `Type` is the parameter name that will be passed to the task command. You will see the significance of the variable name in the next part.

```
<FlowElementImplementation type="Display Store">
      <Implementation invocationType="TaskCommand">
            <Class name="com.mycompany.DisplayStoreTaskCmd">
                  <Argument name="Type" value="MARKETING_storeType"/>
            </Class>
      </Implementation>
</FlowElementImplementation>
```

____ 4. Insert the implementation template in the marketing element template table, DMELETEMPLATE.

__ a. Choose a name for the action. This name will be used to identify the action in Management Center. For this lab, use the name **customDisplayStore.**

__ b. Access the development database using the URL: http://localhost/webapp/wcs/admin/servlet/db.jsp

__ c. Run the following command to insert the implementation template into the DMELETEMPLATE table. The value **1001** was chosen as a unique primary key and the value **3** signifies that this is an action template.

```
insert into dmeletemplate (dmeletemplate_id, dmelementtype_id, name,
implxml) values (1001, 3, 'customDisplayStore',
'<FlowElementImplementation type="Display Store"><Implementation
invocationType="TaskCommand"><Class
name="com.mycompany.DisplayStoreTaskCmd"><Argument name="Type"
value="MARKETING_storeType"/></Class></Implementation></FlowElementIm
plementation>');
```

Part 2: Implement the action task command

The action task command provides the business logic to perform the action. In this example, it will determine which physical stores to return to the customer. An action task command must implement the MarketingCampaignElementTaskCmd interface. In this part of the lab, you will create the new task command specified in the action implementation template.

____ 1. Create the command interface.

__ a. Open WebSphere Commerce Developer and navigate to the **com.mycompany** package.

Note: If you did not do the previous lab you will need to create this package under the **WebSphereCommerceServerExtensionsLogic** project.

__ b. Right click the package and create a new interface called **DisplayStoreTaskCmd**. This interface should extend **MarketingCampaingElementTaskCmd.**

__ c. Click **Finish** to generate the interface.

____ 2. Update the generated interface.

__ a. Add the following member variable. Do not worry about the error. You will fix that in the next step.

```
public final static String defaultCommandClassName =
DisplayStoreTaskCmdImpl.class.getName();
```

__ b. When you are done, your interface should look like the one shown below.

```
package com.mycompany;

import com.ibm.commerce.marketing.commands.elements.MarketingCampaignElementTaskCmd;

public interface DisplayStoreTaskCmd extends
        MarketingCampaignElementTaskCmd {
    public final static String defaultCommandClassName = DisplayStoreTaskCmdImpl.class.getName();
}
```

__ c. Save the file.

____ 3. Create the task command implementation.

__ a. Right click on the **com.mycompany** package and use the pop-up menu to create a new class file**.**

__ b. Set the class name to **DisplayStoreTaskCmdImpl,** add **MarketingCampaignElementTaskCmdImpl** as the superclass and **DisplayStoreTaskCmd** as the interface. Click the **Finish** button to create the class.

__ c. Add code to retrieve a list of physical stores based on the shopper's city of residence. The list will be filtered to display only the type of stores selected when the action was created. Notice in the code below that the element parameters are being passed directly into the Get request for the physical store. This was made possible by the initial implementation template setup. The variable name was set to `Type` which matches the name of the attribute in STLOCATTR that you want to filter on. To simplify the code for this example, exception handling has been skipped. A screen capture of the completed class is shown below. Review the bottom section of code to see how the physical store IDs are returned to the e-Marketing spot. You can copy the code from the snippets or solution file in <LAB_DIR>/MarketingCustomization2.

```
package com.mycompany;

import java.util.List;
import java.util.ListIterator;
import java.util.Map;
import java.util.logging.Logger;

import com.ibm.commerce.foundation.common.util.logging.LoggingHelper;
import com.ibm.commerce.marketing.commands.elements.MarketingCampaignElementTaskCmdImpl;
import com.ibm.commerce.user.objects.AddressAccessBean;
import com.ibm.commerce.store.facade.client.StoreFacadeClient;
import com.ibm.commerce.store.facade.datatypes.*;

import com.ibm.commerce.marketing.runtime.engine.Activity;
import com.ibm.commerce.marketing.runtime.util.EMarketingSpotDataBean;


public class DisplayStoreTaskCmdImpl extends
        MarketingCampaignElementTaskCmdImpl implements
        DisplayStoreTaskCmd {

    private final static Logger LOGGER = LoggingHelper.getLogger(DisplayStoreTaskCmdImpl.class);
    private final static String CLASSNAME = DisplayStoreTaskCmdImpl.class.getName();

    public DisplayStoreTaskCmdImpl() {}

    public void performExecute() {
        final String METHOD_NAME = "performExecute";
        if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
            LOGGER.entering(CLASSNAME, METHOD_NAME);
        }

        try {
            // Set up the geo node search criteria
            Long memberId = getRegisteredMemberIdForPersonalizationId();
            AddressAccessBean abAddress = new AddressAccessBean();
            abAddress = abAddress.findSelfAddressByMember(memberId);

            String geoNodeType = "CITY";
            String geoNodeName = abAddress.getCity();

            //Get the geo node for the shopper
            StoreFacadeClient storeClient = new StoreFacadeClient();
            ShowGeoNodeType showGeoNode =
                storeClient.getGeoNodesByGeoNodeTypeAndName("IBM_Store_All", geoNodeType, geoNodeName);
            GeoNodeType geoNode = (GeoNodeType)showGeoNode.getDataArea().getGeoNode().get(0);
            String geoNodeId = geoNode.getGeoNodeIdentifier().getUniqueID();

            // Get the physical stores for the geo node and search attributes
            Map parameters = getElementParameters();
            ShowPhysicalStoreType showPhysStoreType =
                storeClient.getPhysicalStoresByGeoNodeUniqueID("IBM_Store_Details",geoNodeId,parameters);
            List physicalStores = showPhysStoreType.getDataArea().getPhysicalStore();
            ListIterator iterator = physicalStores.listIterator();

            // Return matching store ids to the e-Marketing spot
            PhysicalStoreType store;
            String storeId;
            Activity activity = getActivity();
            while (iterator.hasNext()) {
```

```
                store = (PhysicalStoreType)iterator.next();
                storeId = store.getPhysicalStoreIdentifier().getUniqueID();
                EMarketingSpotDataBean emsDataBean =
                    new EMarketingSpotDataBean("PhysicalStoreType", storeId, activity, getElementId(),
                            getExperimentTestElements());
                addEMarketingSpotDataBean(emsDataBean);
            }
        }
        catch (Exception e) {/*Exception handling omitted for this example*/}

        //For an action return true
        setReturnValue(true);

        if (LoggingHelper.isEntryExitTraceEnabled(LOGGER)) {
            LOGGER.exiting(CLASSNAME, METHOD_NAME);
        }
    }
}
```

__ d. Save your work.

Part 3: Add the new action to Management Center

In this part of the lab, you will update the Management Center activity builder to support the new action.

____ 1. Create a new OpenLaszlo file for the action element object definition.

__ a. In the LOBTools project, navigate to the folder **…/lzx/mycompany/marketing/objectDefinitions/activityBuilder**. Right click and select **New > File.**

Note: If you did not do the previous lab you will need to create the directory structures **mycompany/marketing/objectDefinitions/activityBuilder** and **mycompany/marketing/resources**.

__ b. Name your new file **DisplayStoreFlowElementObjectDefinition.lzx**. Click **Finish** to create the file.

____ 2. Create the object definition. Triggers, targets and actions are all activity builder flow elements. When you add a new flow element it must extend the class `mktFlowElementObjectDefinition.`

__ a. Add the following code to the file DisplayStoreFlowElementObjectDefinition.lzx. There are two new attributes that were not included in the first lab. `propertiesClass` is the class name for the properties panel and `summaryClass` is the class that creates the summary text displayed in the activity flow diagram when properties are set for the action element. Notice the `elemTemplateType` attribute is set to Action. The `wcfPropertyDefinition` section defines the variable included in the implementation template and specifies the choices available to the business user. The `value` for each option matches the possible values in the STLOCATTR table. This ensures the correct search value is sent to the action task command you defined in the Part 2.

```
<library>
      <class name="extDisplayStoreElementObject"
      extends="mktFlowElementObjectDefinition"
      objectType="customDisplayStore"
      displayName="Recommend physical store"
      headerIcon="customDisplayStoreHeaderIcon"
      flowIcon="customDisplayStoreIcon"
      paletteIcon="customDisplayStorePaletteIcon"
      propertiesClass="extDisplayStoreProperties"
      summaryClass="extDisplayStoreSummary"
      elemTemplateType="Action">

            <mktFlowElementCreateService/>
            <mktFlowElementUpdateService/>

            <dataset name="template">
            <elemTemplateName>customDisplayStore</elemTemplateName>
            </dataset>

            <wcfPropertyDefinition propertyName="storeType"
            displayName="Recommend the following store type">
                  <wcfPropertyValue displayName="Regular"
                  value="Regular Store"/>
                  <wcfPropertyValue displayName="Express"
                  value="Express Store"/>
                  <wcfPropertyValue displayName="Outlet"
                  value="Outlet Store"/>
            </wcfPropertyDefinition>

      </class>
```

```
</library>
```

__ b. Save the file.

____ 3. Define the display icons. In the object definition above, three icon resources are used. You will define those resources now.

__ a. Navigate to the **resources** directory open the file called **ExtMarketingResources.lzx**

Note: You need to create this file if you did not do the first lab.

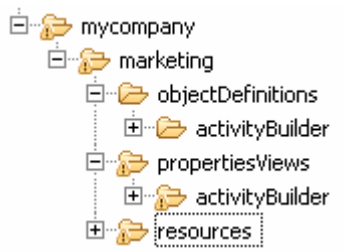
__ b. Add the following code to the file between the `<library>` tags. To simplify this example, you will reuse the recommend category action icon for your new action.

```
<resource name="customDisplayStorePaletteIcon"
src="../../../commerce/marketing/resources/pal/category.png"/>
<resource name="customDisplayStoreIcon"
src="../../../commerce/marketing/resources/dgm/category.png"/>
<resource name="customDisplayStoreHeaderIcon"
src="../../../commerce/marketing/resources/hdr/category.png"/>
```

__ c. Save the file.

____ 4. Create the properties view for the action.

__ a. Create the directories **propertiesViews** and **activityBuilder** under your custom marketing directory as shown below.



__ b. Right click the new **activityBuilder** folder and create a new file **DisplayStorePropertiesView.lzx**.

__ c. Add the code shown below to the file. This is a very simple properties view that displays a single drop down for the business user to select which type of store to recommend. The options were defined in the object definition in Step 2 above.

```
<library>
      <class name="extDisplayStoreProperties"
      extends="mktFlowElementProperties">
            <wcfPropertyGroup name="group" collapsable="false">
                  <wcfPropertyCombobox promptText="Store type to
                  recommend" propertyName="storeType"/>
            </wcfPropertyGroup>
      </class>
</library>
```

__ d. Save the file.

____ 5. Create the summary class for the action. This class generates the text that is shown in the activity flow once the business user has selected a type of store to recommend.

__ a. Right click the **propertiesView/activityBuilder** folder and create a new file called **DisplayStoreSummary.lzx**

__ b. Add the code shown below to the file. The summary is a simple text string that contains the type of store selected.

```
<library>
      <class name="extDisplayStoreSummary"
extends="mktFlowElementSummary">
            <mktFlowSummaryParam name="storeType"
            propertyName="storeType"/>
            <method name="updateSummary" args="e">
                  <![CDATA[
                        var summary = "";
                        if (this.resolvedParams["storeType"] != "") {
                              summary = "Refer " +
                              this.resolvedParams["storeType"];
                        }
                        this.summary.setText(summary);
                  ]]>
            </method>
      </class>
</library>
```

__ c. Save the file.

____ 6. Register your custom OpenLaszlo files with the Management Center Marketing tool**.**

__ a. Navigate to **…/lzx/commerce/marketing** and open the file **MarketingExtenstionsLibrary.lzx**

__ b. Add the names of the files you just created. If the resources file is not already included from the first lab, add it as well.

```
<include
href="../../mycompany/marketing/objectDefinitions/activityBuilder/Dis
playStoreFlowElementObjectDefinition.lzx"/>
<include
href="../../mycompany/marketing/propertiesViews/activityBuilder/Displ
ayStorePropertiesView.lzx"/>
<include
href="../../mycompany/marketing/propertiesViews/activityBuilder/Displ
ayStoreSummary.lzx"/>
```

__ c. Save the file.

____ 7. Create an instance of the action object**.**

__ a. Navigate to **…/lzx/commerce/marketing/objectDefinitions/activityBuilder** and open the file **FlowPathElementObjectDefinition.lzx**.

__ b. Scroll to the bottom of the file and add an instance of the `extDisplayStoreElementObject` class to the bottom of the list of element objects. Enter the following line of code.

```
<extDisplayStoreElementObject/>
```

__ c. Save the file.

____ 8. Add the action to the activity builder palette.

__ a. Navigate to **…/lxz/commerce/marketing/propertiesViews** and open the file **WebActivityBuilder.lzx.**

__ b. Scroll through the file until you locate the actions group.

```
<Group resourceBundle="mktMarketingResources" name="actions"
    <Element objectType="displayProduct"/>
    <Element objectType="displayContent"/>
    <Element objectType="displayCategory"/>
    <Element objectType="displayAssociation"/>
    <Element objectType="displayPromotion"/>
    <Element objectType="recentlyViewed"/>
    <Element objectType="addToRemoveFromCustomerSegment"/>
```

__ c. At the bottom of the actions group, add a new element with `objectType` **customDisplayStore**. The updated group should look like the following screen capture.

```
<Group resourceBundle="mktMarketingResources" name="actions"
    <Element objectType="displayProduct"/>
    <Element objectType="displayContent"/>
    <Element objectType="displayCategory"/>
    <Element objectType="displayAssociation"/>
    <Element objectType="displayPromotion"/>
    <Element objectType="recentlyViewed"/>
    <Element objectType="addToRemoveFromCustomerSegment"/>
    <!-- New action -->
    <Element objectType="customDisplayStore"/>
    <!-- This prebuilt element is available for customizatio
         DisplayExternalRecommendationActionTaskCmd task com
    <Element objectType="displayExternalRecommendation"/>
    -->
</Group>
```

__ d. Save the file. You have now updated the palette for the Web activity builder. You can also add this action to the Web activity template builder. Each builder definition is in a separate file in this same directory. This action cannot be used in a Dialog activity because it returns content to an e-Marketing Spot which does not exist in a Dialog activity.

____ 9. Compile the Management Center project. Right click the **LOBTools** project and select **Build OpenLaszlo Project.**

____ 10. Update the Struts extension file. In order to read the new action information from the database and correctly display it in Management Center, you need to add a new action to the Struts configuration file.

__ a. Navigate to **LOBTools/WebContent/WEB-INF** and open the file **struts-extension.xml.**

__ b. Within the `<action-mappings>` block, add the following new action.

```
<action path="/SerializeActivityElement-customDisplayStore"
include="/jsp/commerce/marketing/restricted/SerializeActivitygenericE
lement.jsp" />
```

__ c. Save the file.

__ d. Start your test server. If it is already running, restart it.

Part 4: Update the store front

In this part of the lab, you will update the Madisons store front to support the physical store data being returned to an e-Marketing spot. In this lab you will display the store IDs to test that your new action works. You can examine the rest of the ESpotNavDisplay snippet to see how existing marketing content is formatted.

____ 1. Update the e-Marketing spot snippet.

__ a. Open the Stores project and navigate to **WebContent/Madisons/Snippets/Marketing/ESpot** and open the file **ESpotNavDisplay.jsp.** This snippet file displays an e-Marketing Spot in the right sidebar of Madisons store pages.

__ b. Navigate to **<LAB_FILE_DIR>/MarketingCustomization2/MktCust2Snippets.txt** and select the snippet code for this step. Scroll to the bottom of **ESpotNavDisplay.jsp** and add the snippet code just before the closing `</div>` tag. The code to add is shown below. Note the `PhysicalStoreType` data type is being matched. This is the data type name you defined in Part 2. The line of code that prints out the physical store IDs is also circled below.

```
<c:set var="currentStoreCount" value="0" />
<c:set var="headerDisplayed" value="false" />
<c:forEach var="marketingSpotData" items="${marketingSpotDatas.baseMarketingSpotActivityData}" varStatus="extStatus">
    <c:if test='${marketingSpotData.dataType eq "PhysicalStoreType"}'>
        <c:if test="${!headerDisplayed}">
            <div id="recommend">
                <div class="header" id="Ext_RightSidebarESpotDisplay_div_1_<c:out value='${extStatus.count}'/>">
                    <h2 class="sidebar_header">Physical Stores</h2>
                    <p>The following stores are in your area:</p>
                </div>
                <div class="content" id="Ext_RightSidebarESpotDisplay_div_2_<c:out value='${extStatus.count}'/>">
                    <c:set var="headerDisplayed" value="true" />
        </c:if>

                    <c:set var="currentStoreCount" value="${currentStoreCount+1}" />

                    <!--  Write out the physical store id -->
                    <div class="itemWrapper">
                        <p>${marketingSpotData.uniqueID}</p>
                    </div>
    </c:if>
</c:forEach>
<c:if test="${currentStoreCount != 0}">
                </div>
                <div class="footer" id="WC_RightSidebarESpotDisplay_div_5">
                    <img src="${jspStoreImgDir}${vfileColor}recommend_bot.png" alt="" />
                </div>
            </div>
</c:if>
```

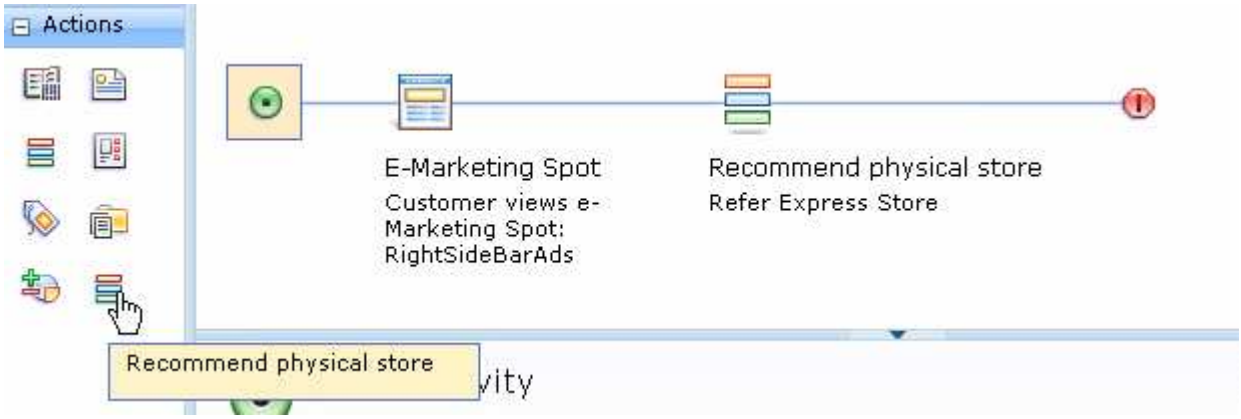
__ c. Save the file.

Part 5: Test your new action

In this part of the lab, you will test the changes you have made.

____ 1. Launch Management Center using the URL **https://<WCDE_HOST>:8000/lobtools**.

____ 2. Create a Web activity using your new action. Save and activate the activity.



____ 3. Launch the Madisons starter store using the URL **http://<WCDE_HOST>/webapp/wcs/stores/servlet/Madisons/index.jsp**

____ 4. Register a new shopper or log in to an existing account. Make sure there are physical stores defined for the city specified in the shopper's account information. You can confirm this using the Store Locator feature in Madisons.

____ 5. Navigate to the page where your test Web activity is running. Confirm the shopper sees a list of recommended physical store IDs.

Part 6: What you did in this exercise

In this tutorial you learned how to create and test a new marketing action element.

You should now understand how to complete the following tasks:

- Define and register a marketing element implementation template that includes variables
- Implement a task command to perform an action
- Pass a new data type to an e-Marketing spot
- Add a new action to the Management Center Web activity palette and create a properties view for it
- Display a new e-Marketing spot data type in the Madisons store