IBM WebSphere Commerce

WebSphere Commerce Accelerator Customization Guide

Version 5.4

IBM WebSphere Commerce

WebSphere Commerce Accelerator Customization Guide

Version 5.4

Note:
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 93.

First Edition (January 2002).

This edition applies to version 5.4 of IBM® WebSphere Commerce, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

IBM welcomes your comments. You can send your comments by any one of the following methods:

1. Electronically to either of the network IDs listed below. Be sure to include your entire network address if you wish a reply.

   ```
   Internet: torrcf@ca.ibm.com
   IBMLink: toribm(torrcf)
   ```

2. By FAX, use the following numbers:

   ```
   United States and Canada:  416-448-6161
   Other countries: (+1)-416-448-6161
   ```

3. By mail to the following address:

   IBM Canada Ltd. Laboratory
   B3/KB7/8200/MKM
   8200 Warden Avenue
   Markham, Ontario,
   L6G 1C7
   Canada

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2001. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2001

Before you begin

Conventions used in this book

This book uses the following highlighting conventions:

Boldface type indicates commands or graphical user interface (GUI) controls such as names of fields, buttons, or menu choices.

`Monospaced type` indicates examples of text you enter exactly as shown, as well as directory paths.

Italic type is used for emphasis and variables for which you substitute your own values.

This icon marks a Tip — additional information that can help you complete a task.

Knowledge requirements

To customize the WebSphere Commerce Accelerator, you require knowledge of the following:

- HTML and XML
- Structured Query Language (SQL)
- Java Programming

Please refer to the WebSphere Commerce Programmer's Guide for more information on customizing WebSphere Commerce. This book is available from the following Web site:

`www.ibm.com/software/webservers/commerce/wcs_pro/lit-tech-general.html`

How this book is organized

The following table outline's the book's organization:

Table 1.

Component	Description	Location
Business Relationship Management	This section details the pieces that you must consider when customizing the Contracts tools	Chapter 1, "Business relationship management" on page 3
Customer Service Representative	This section details the pieces that you must consider when customizing the Customer Service Representative's tools	Chapter 2, "Customer Service Representative tools" on page 9
Customer Profiles	This section details the pieces that you must consider when customizing the Customer Profile tools	Chapter 4, "Customer profiles" on page 21

© Copyright IBM Corp. 2001

Table 1. (continued)

Component	Description	Location
Campaigns	This section details the pieces that you must consider when customizing the Campaigns tools	Chapter 5, “Campaigns” on page 29
Coupons	This section details the pieces that you must consider when customizing the Coupons tools	Chapter 7, “Coupons” on page 53
Discounts	This section details the pieces that you must consider when customizing the Discounts tools	Chapter 8, “Discounts” on page 55
RFQ	This section details the pieces that you must consider when customizing the RFQ tools	Chapter 9, “RFQ response” on page 57
Inventory	This section details the pieces that you must consider when customizing the Inventory tools	Chapter 10, “Expected Inventory” on page 69
Business Intelligence	This section details the pieces that you must consider when customizing the reporting tools	Chapter 11, “Business intelligence” on page 71
Reporting Framework	This section details the pieces that you must consider when customizing the reporting tools	Chapter 12, “Overview of the Reporting framework” on page 73
Search	This section details the pieces that you must consider when customizing the catalog search tools	Chapter 6, “Catalog search” on page 37
Blaze Rules Advisor	This section details the pieces that you must consider when customizing the Brokat Advisor rule projects.	Chapter 14, “Rule Projects” on page 89
Product Advisor	This section details the pieces that you must consider when customizing the Product Advisor tools	Chapter 13, “Product Advisor” on page 87

Part 1. Customizing the WebSphere Commerce Accelerator

This book describes how to customize the WebSphere Commerce Accelerator. By providing background knowledge about the design decisions for the WebSphere Commerce Accelerator, this book teaches you how to approach the customization, and details the steps required for customization.

While this book is intended to help guide you through customizing the WebSphere Commerce Accelerator, it is not designed to walk you through an exhaustive list of all of the customizations that are possible. Instead, the book's structure introduces the different generic pieces that make up the WebSphere Commerce Accelerator and explains how to customize these pieces.

The WebSphere Commerce Accelerator is a collection of tools designed to facilitate the day-to-day business operations of your site. That is, it is intended as an interface for business people to create and modify any number of aspects of the site's operations without having to continually contact the IT staff responsible for the site's operation. As such, there are numerous components that comprise the WebSphere Commerce Accelerator, each targeting a key aspect of the business. While every effort has been made to make these tools universal, this same goal means that some users may find that certain requirements are not met. If you extended the database during your site's creation, and these extensions relate to any of the tools listed in this document, the data will be invisible to the Accelerator until you customize the relevant tool.

This book describes how to customize the following components in the WebSphere Commerce Accelerator.

© Copyright IBM Corp. 2001

Chapter 1. Business relationship management

The elements discussed in this chapter represent the user interface to the Accounts and Contracts components in the WebSphere Commerce Accelerator. They facilitate the creation and maintenance of accounts, contracts, and other actions that a Sales Manager or Account Representative need to perform their daily tasks associated with the business. The business relationship management component includes the following elements:

- Accounts notebook
- Contracts notebook

Customization examples

The following examples outline how to customize this part of the WebSphere Commerce Accelerator.

Scenario 1: Add a new term and condition to the Contract user interface

Implementation overview

In an attempt to make the WebSphere Commerce Accelerator generic and as useful to as many people as possible, some terms and conditions were omitted. Therefore, you may find it necessary to add terms and conditions to your site to make it more applicable to your business. This scenario guides you through the steps required to add additional terms and conditions to your site.

Customization steps

1. Define the term and condition in the server. Before you can perform the user interface customization, you must have already completed the following:
   - Define a new term and condition in the `B2BTrading.dtd` file. For full details, refer to Chapter 7 in the *IBM WebSphere Commerce Programmer's Guide*.
   - Create an Enterprise Bean or Access Bean for the new term and condition.
2. Add the term and condition to the user interface. To add the required elements to the user interface, do the following:
   a. Create a JSP file for the user interface page for the term and condition. This page must include a dynamic JavaScript object to capture the data from the input fields on the page. Create a data bean that encapsulates loading the data from the Access bean. Optionally, include the Access bean directly on the page to load the term and condition data from the database. The `contractId` parameter is passed to all pages to help reduce data load performance. A sample of this JSP file follows:

```
<!--=========================================================================
//*-------------------------------------------------------------------------
//* The sample contained herein is provided to you "AS IS".
//*
//* It is furnished by IBM as a simple example and has not been thoroughly tested
//* under all conditions.  IBM, therefore, cannot guarantee its reliability,
//* serviceability or functionality.
//*
//* This sample may include the names of individuals, companies, brands and
//* products in order to illustrate concepts as completely as possible.  All of
//* these names are fictitious and any similarity to the names and addresses used
//* by actual persons or business enterprises is entirely coincidental.
//*-------------------------------------------------------------------------
//*
  ==========================================================================-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<%@page language="JAVA"
import="com.ibm.commerce.tools.util.UIUtil,
```

© Copyright IBM Corp. 2001

```
com.ibm.commerce.beans.DataBeanManager,
com.ibm.commerce.tools.contract.beans.MemberDataBean,
com.ibm.commerce.tools.contract.beans.MyTCDataBean,
com.ibm.commerce.tools.contract.beans.PolicyDataBean,
com.ibm.commerce.tools.contract.beans.PolicyListDataBean"
%>

<%@include file="../common/common.jsp" %>
<%@include file="ContractCommon.jsp" %>

<HTML>

<HEAD>
 <%= fHeader %>
 <LINK rel="stylesheet" href="<%= UIUtil.getCSSFile(fLocale) %>" type="text/css">

 <TITLE><%= contractsRB.get("MyTCHeading") %></TITLE>
 <SCRIPT LANGUAGE="JavaScript" SRC="/wcs/javascript/tools/common/Util.js"></SCRIPT>
 <SCRIPT LANGUAGE="JavaScript" SRC="/wcs/javascript/tools/contract/ContractUtil.js"></SCRIPT>
 <SCRIPT LANGUAGE="JavaScript" SRC="/wcs/javascript/tools/contract/Extensions.js"></SCRIPT>


<SCRIPT LANGUAGE="JavaScript">

var MyTCModel;

////////////////////////////////////////
// LOAD-SAVE-VALIDATE SCRIPTS
////////////////////////////////////////
function onLoad() {

   if (parent.setContentFrameLoaded) {
       parent.setContentFrameLoaded(true);
   }

   // Check to see if the model has already been loaded
   var isModelLoaded = parent.get("ContractMyTCModelLoaded", null);

   if (isModelLoaded) {
       // Returning to this page, reload from the model
       // Get the model
       MyTCModel = parent.get("ContractMyTCModel", null);
   }
   else {
       // First visit to this page, create the model

       // Create the model to store the MyTC data
       MyTCModel = new ContractMyTCModel();

       // Persist the model
       parent.put("ContractMyTCModel", MyTCModel);
       parent.put("ContractMyTCModelLoaded", true);

var myTCPolicyList = new Array();
<%
try {
// Load all the policies from the database
PolicyListDataBean policyList = new PolicyListDataBean();
PolicyDataBean policy[] = null;

policyList.setPolicyType(policyList.TYPE_PRODUCT_SET);
DataBeanManager.activate(policyList, request);
policy = policyList.getPolicyList();

for (int i = 0; i < policy.length; i++) {
MemberDataBean mdb = new MemberDataBean();
mdb.setId(policy[i].getStoreMemberId());
DataBeanManager.activate(mdb, request);
%>
myTCPolicyList[myTCPolicyList.length] =
new PolicyObject('<%=UIUtil.toJavaScript(policy[i].getShortDescription())%>',
'<%= policy[i].getPolicyName() %>',
'<%= policy[i].getId() %>',
'<%= policy[i].getStoreIdentity() %>',
new Member('<%= mdb.getMemberType() %>',
'<%= mdb.getMemberDN() %>',
'<%= mdb.getMemberGroupName() %>',
'<%= mdb.getMemberGroupOwnerMemberType() %>',
'<%= mdb.getMemberGroupOwnerMemberDN() %>')
);
<%
}
} catch (Exception e) {}
%>
MyTCModel.policyList = myTCPolicyList;

// Check if this is an update of the contract
if (<%= foundContractId %> == true) {
// Load the data from the databean
<%
if (foundContractId) {
MyTCDataBean tc = new MyTCDataBean(new Long(contractId));
DataBeanManager.activate(tc, request);
if (tc.getHasMyTC()) {
%>
MyTCModel.attr1 = '<%= UIUtil.toJavaScript((String)tc.getAttr1()) %>';
MyTCModel.attr2 = '<%= UIUtil.toJavaScript((String)tc.getAttr2()) %>';
MyTCModel.tcReferenceNumber = '<%= tc.getReferenceNumber() %>';
MyTCModel.policyReferenceNumber = '<%= tc.getPolicyReferenceNumber() %>';
for (var i = 0; i < myTCPolicyList.length; i++) {
if (myTCPolicyList[i].policyId == '<%= tc.getPolicyReferenceNumber() %>') {
MyTCModel.selectedPolicyIndex = i;
}
}
<%
```

```
}
}
%>
}

   }

   loadPanelData();

    // handle error messages back from the validate page
    if (parent.get("attr1Empty", false))
     {
      parent.remove("attr1Empty");
      alertDialog("<%= UIUtil.toJavaScript((String)contractsRB.get("attr1Empty"))%>");
     }
    else if (parent.get("attr2Empty", false))
     {
      parent.remove("attr2Empty");
      alertDialog("<%= UIUtil.toJavaScript((String)contractsRB.get("attr2Empty"))%>");
     }
    else if (parent.get("attr1TooLong", false))
     {
      parent.remove("attr1TooLong");
      alertDialog("<%= UIUtil.toJavaScript((String)contractsRB.get("attr1TooLong"))%>");
     }
    else if (parent.get("attr2TooLong", false))
     {
      parent.remove("attr2TooLong");
      alertDialog("<%= UIUtil.toJavaScript((String)contractsRB.get("attr2TooLong"))%>");
     }

   return;
}

function loadPanelData() {

// Set the input fields
document.MyTCForm.Attr1.value = MyTCModel.attr1;
document.MyTCForm.Attr2.value = MyTCModel.attr2;

// Load the policies
for (var i = 0; i < MyTCModel.policyList.length; i++) {
if (MyTCModel.selectedPolicyIndex == i) {
document.MyTCForm.PolicyList.options[i] = new Option(MyTCModel.policyList[i].displayText,
i, true, true);
} else {
   document.MyTCForm.PolicyList.options[i] = new Option(MyTCModel.policyList[i].displayText,
i, false, false);
}
}
}

function savePanelData() {
    MyTCModel.attr1 = document.MyTCForm.Attr1.value;
    MyTCModel.attr2 = document.MyTCForm.Attr2.value;
    MyTCModel.selectedPolicyIndex = document.MyTCForm.PolicyList.selectedIndex;
}
</SCRIPT>

</HEAD>

<!--
////////////////////////////////////////
// HTML SECTION
////////////////////////////////////////
-->

<BODY onLoad="onLoad()" class="content">

   <H1>
   <%= contractsRB.get("MyTCHeading") %>
   </H1>

<FORM NAME="MyTCForm">

   <%= contractsRB.get("MyTCAttr1Label") %>
   <BR>
   <INPUT type=text name=Attr1 value="" size=10 maxlength=10>
   <BR>

   <%= contractsRB.get("MyTCAttr2Label") %>
   <BR>
   <INPUT type=text name=Attr2 value="" size=10 maxlength=10>
   <BR>

   <%= contractsRB.get("MyTCPolicyLabel") %>
   <BR>
   <SELECT NAME="PolicyList" SIZE="1">
   </SELECT>

</FORM>

</BODY>
</HTML>
```

b. Create a JavaScript file for the user interface page for the term and condition. Create functions to validate and submit the JavaScript data.

When you submit the data, you need to create a new JavaScript object that matches the XML format for the new term and condition. A sample of the JavaScript file follows:

```
//*-------------------------------------------------------------------------------------
//* The sample contained herein is provided to you "AS IS".
//*
//* It is furnished by IBM as a simple example and has not been thoroughly tested
//* under all conditions.  IBM, therefore, cannot guarantee its reliability,
//* serviceability or functionality.
//*
//* This sample may include the names of individuals, companies, brands and products
//* in order to illustrate concepts as completely as possible.  All of these names
//* are fictitious and any similarity to the names and addresses used by actual persons
//* or business enterprises is entirely coincidental.
//*-------------------------------------------------------------------------------------
//*

function ContractMyTCModel() {

  this.tcReferenceNumber = "";
  this.policyReferenceNumber = "";

  this.attr1 = "";
  this.attr2 = "";

  this.policyList = new Array();
  this.selectedPolicyIndex = "0";
}

function validateMyTCPanel() {

  var tcModel = get("ContractMyTCModel");
  if (tcModel != null) {
  // Check if attr1 is empty or too long
  if (!tcModel.attr1)
   {
    put("attr1Empty", true);
    gotoPanel("MyTCHeading");
    return false;
   }

  if (!isValidUTF8length(tcModel.attr1, 10))
   {
    put("attr1TooLong", true);
    gotoPanel("MyTCHeading");
    return false;
   }
  // Check if attr2 is empty or too long
  if (!tcModel.attr2)
   {
    put("attr2Empty", true);
    gotoPanel("MyTCHeading");
    return false;
   }

  if (!isValidUTF8length(tcModel.attr2, 10))
   {
    put("attr2TooLong", true);
    gotoPanel("MyTCHeading");
    return false;
   }
  }

}

function submitMyTC(termsAndConditions) {

  var tcModel = get("ContractMyTCModel");

  if (tcModel != null) {

var myTC = new Object();
myTC.MyTC = new Object();
myTC.MyTC.MySubTC = new Object();
myTC.MyTC.MySubTC.attr1 = tcModel.attr1;
myTC.MyTC.MySubTC.attr2 = tcModel.attr2;

myTC.MyTC.PolicyReference = new Object();
myTC.MyTC.PolicyReference.policyName = tcModel.policyList[tcModel.selectedPolicyIndex].policyName;
myTC.MyTC.PolicyReference.policyType = "ProductSet";
myTC.MyTC.PolicyReference.storeIdentity = tcModel.policyList[tcModel.selectedPolicyIndex].storeIdentity;
myTC.MyTC.PolicyReference.Member = tcModel.policyList[tcModel.selectedPolicyIndex].member;

if (tcModel.tcReferenceNumber != "") {
// Change the term and condition
myTC.action = "update";
myTC.referenceNumber = tcModel.tcReferenceNumber;
}
else {
// Create a new term and condition
myTC.action = "new";
}

termsAndConditions[termsAndConditions.length] = myTC;
  }

  return true;
}

function validateContractExtensions() {
   if (this.validateMyTCPanel) {
     if (validateMyTCPanel() == false) {
```

```
return false;
    }
  }
  return true;
}

function submitContractExtensions(termsAndConditions) {
  if (this.submitMyTC) {
    if (submitMyTC(termsAndConditions) == false) {
return false;
    }
  }
}
```

c. Modify the `Contract.js` file to call the new `submit` and `validate` functions created in step 2b.

d. Modify the resource bundle to add the new strings that you require.

e. Modify the `ContractNotebook.xml` to add the new panel.

Scenario 2: Remove a term and condition from the Contract user interface

Implementation overview

Just as you may find that some terms and conditions are missing, you may also find that some are unnecessary for your business. This scenario guides you through the steps required to remove extraneous terms and conditions from your site.

Customization steps

Modify the `ContractNotebook.xml` file and remove the section that contains the panel element for the term and condition you want to remove. Locate this file in the following directory:

AIX `/usr/WebSphere/CommerceServer/xml/tools/contract/`

400 `/QIBM/UserData/WebSphere/CommerceServer/CommerceServer/xml/tools/contract/`

Linux `/opt/WebSphere/CommerceServer/xml/tools/contract/`

Solaris `/opt/WebSphere/CommerceServer/xml/tools/contract/`

Windows *drive*`:\WebSphere\CommerceServer\xml\tools\contract\`

Chapter 2. Customer Service Representative tools

The elements discussed in this chapter represent the user interface for the Customer Service Representative (CSR) tools in the WebSphere Commerce Accelerator. They facilitate the creation and maintenance of customers, orders, and other actions that a CSR needs to perform daily tasks associated with the business. The CSR tools component includes the following elements:

- Customer wizard
- Customer notebook
- Order wizard
- Order notebook

Customization examples

The following examples outline how to customize the CSR tools within the WebSphere Commerce Accelerator.

Scenario 1: Add additional customer data to the Business Customer summary dialog

Implementation overview

This example shows how to add a Business Customer's employee number to the Business Customer summary dialog.

Customization step

Extend the `ShopperSummaryB2BDialog.jsp`. Locate this file in the following directory:

AIX `/usr/WebSphere/CommerceServer/web/tools/csr/`

400

`/QIBM/UserData/WebSphere/CommerceServer/CommerceServer/web/tools/csr/`

Linux `/opt/WebSphere/CommerceServer/web/tools/csr/`

Solaris `/opt/WebSphere/CommerceServer/web/tools/csr/`

Windows *drive*`:\WebSphere\CommerceServer\web\tools\csr\`

See the code sample below on how to change the JSP file to display the employee number in the summary dialog. The employee number is a property of the `OptoolsRegisterDataBean`. The sample did not use properties files to store the label `Employee Number`. Sample output which has the employee number displayed in the Customer Summary dialog:

```
<%--
//*************************************************************************
//*-------------------------------------------------------------------------
//* Licensed Materials - Property of IBM
//*
//* 5724-A18
//*
//* (c) Copyright IBM Corp. 2001
//*
//* US Government Users Restricted Rights - Use, duplication or
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//*
//*-------------------------------------------------------------------------
//*
--%>
```

© Copyright IBM Corp. 2001

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
:
:
: Some code is omitted here
:
:
<HTML>

<HEAD>
<LINK rel=stylesheet
        href="<%= UIUtil.getCSSFile(cmdContext.getLocale()) %>"
        type="text/css">
<SCRIPT SRC="/wcs/javascript/tools/common/Util.js"></SCRIPT>

<script>
<%@ include file = "SummaryDisplay.jsp" %>
:
:
: Some code is omitted here
:
:
</script>

</HEAD>
<BODY CLASS=content onLoad = "initializeState();">
<p>
</p>
<FORM NAME="profile" action="" Method="POST">

   <INPUT type="hidden" name="logonId" value="">
   <INPUT type="hidden" name="XML" value="">
   <INPUT type="hidden" name="URL" value="">

<h1><%= userNLS.get("customerSummaryTitle") %></h1>
<P><B><%=userNLS.get("generalHeader")%></B>
<BR>
   <%=userNLS.get("logonid")%>
   <I><%=UIUtil.toHTML(registerDataBean.getLogonId()) %></I>
<BR>
   <%=userNLS.get("custName")%>:
   <I><script>displayNameSummary()</script></I>
<BR>
   Employee Number:
   <I><%=UIUtil.toHTML(registerDataBean.getEmployeeId()) %></I>
<BR>
   <%=userNLS.get("orgAccount")%>:
<% if (accountDBean != null) { %>
   <I><%=UIUtil.toHTML(accountDBean.getAccountName()) %></I>
<% } %>
<BR>
   <%=userNLS.get("challengeQuestion")%>:
   <I><%=UIUtil.toHTML(registerDataBean.getChallengeQuestion())%></I>
<BR>
   <%=userNLS.get("challengeAnswer")%>:
   <I><%=UIUtil.toJavaScript(registerDataBean.getChallengeAnswer())%></I>
<BR>
   <%=userNLS.get("clientCertificate")%>:
   <I>
<% if (certStatus != "") { %>
   <% if (certStatus == "V") {%><%=userNLS.get("valid")%><%}%>
   <% if (certStatus == "E") {%><%=userNLS.get("expired")%><%}%>
   <% if (certStatus == "R") {%><%=userNLS.get("revoked")%><%}%>
<% } else { %>
   <%=userNLS.get("noCertificate")%>
<% } %>
   </I>
<BR>
   <%=userNLS.get("status")%>:
   <I>
<% if (userRegistry.getStatus().equals("1")) {%>
   <%=userNLS.get("accountStatusEnabled")%>
<% } else { %>
   <%=userNLS.get("accountStatusDisabled")%>
<% } %>
   </I>
<BR>
<P><B><%=userNLS.get("contactHeader")%></B>
   <TABLE border="0" CELLPADDING=0 CELLSPACING=0 >
      <TR valign="top">
```

```
          <TD><%=userNLS.get("address")%>:</TD>
          <TD><I><script>displayAddrSummary(0)</script></I></TD>
      </TR>
      <TR valign="top">
         <TD></TD>
         <TD><I><script>displayAddrSummary(1)</script></I></TD>
      </TR>
      <TR valign="top">
         <TD></TD>
         <TD><I><script>displayAddrSummary(2)</script></I></TD>
      </TR>
      <TR valign="top">
         <TD></TD>
         <TD><I><script>displayAddrSummary(3)</script></I></TD>
      </TR>
      <TR valign="top">
         <TD></TD>
         <TD><I><script>displayAddrSummary(4)</script></I></TD>
      </TR>
   </TABLE>
<%=userNLS.get("phone")%>:
   <I><%=UIUtil.toHTML(address.getPhone1())%></I>
<BR>
   <%=userNLS.get("fax")%>:
   <I><%=UIUtil.toHTML(address.getFax1())%></I>
<BR>
   <%=userNLS.get("email")%>:
   <I><%=UIUtil.toHTML(address.getEmail1())%></I>
<BR>
<P><B><%=userNLS.get("orgHeader")%></B>
<BR>
   <TABLE border="0" CELLPADDING=0 CELLSPACING=0 >
      <TR valign="top">
         <TD><%=orgEntityNLS.get("OrgEntityDeliveryDescription")%>:</TD>
         <TD>
<% if (orgEntity != null) { %>
   <I><%=UIUtil.toHTML(orgEntity.getDescription())%></I>
<% } %>
         </TD>
      </TR>
   </TABLE>
   <TABLE border="0" CELLPADDING=0 CELLSPACING=0 >
      <TR valign="top">
         <TD><%=orgEntityNLS.get("OrgEntityGeneralBusCat")%>:</TD>
         <TD>
<% if (orgEntity != null) { %>
      <I><%=UIUtil.toHTML(orgEntity.getBusinessCategory())%></I>
<% } %>
         </TD>
      </TR>
   </TABLE>

</FORM>
<%
}
catch (Exception e)
{
   e.printStackTrace();
}

%>

</BODY>
</HTML>
```

Scenario 2: Enable ShopCart in Customer Service - Order management interface

Implementation overview

If the store designer wants to manage shopper's orders in state `P` using the CSR tools, they must modify the**CSROrderSearchB2B** and **CSROrderSearchB2C** views.

Customization step

Extend the CSROrderSearchB2B.jsp and CSROrderSearchB2B.jsp JSP files. By adding the Pending option in the order state criteria, the CSR will be able to search the customer's pending orders and manage them. Pending orders are those orders in the shopping cart. Both of these files are in the following directory:

AIX /usr/WebSphere/CommerceServer/web/tools/order/

400 /QIBM/UserData/WebSphere/CommerceServer/CommerceServer/web/tools/order/

Linux /opt/WebSphere/CommerceServer/web/tools/order/

Solaris /opt/WebSphere/CommerceServer/web/tools/order/

Windows *drive*:\WebSphere\CommerceServer\web\tools\order\

See the following code sample:

```
<%--
//*************************************************************************
//*-------------------------------------------------------------------------
//* Licensed Materials - Property of IBM
//*
//* 5724-A18
//*
//* (c) Copyright IBM Corp. 2001
//*
//* US Government Users Restricted Rights - Use, duplication or
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//*
//*-------------------------------------------------------------------------
//* --%>
<%@ page language="java" %>
<%@ page import="java.util.*" %>
<%@ page import="com.ibm.commerce.tools.util.*" %>
<%@ page import="com.ibm.commerce.command.CommandContext" %>
<%@ page import="com.ibm.commerce.server.*" %>
<%@ page import="com.ibm.commerce.tools.optools.user.beans.*" %>
<%@ page import="com.ibm.commerce.tools.optools.order.commands.*" %>
<%@ page import="com.ibm.commerce.tools.contract.beans.*" %>
<%@ page import="com.ibm.commerce.beans.*" %>
<%@ page import="com.ibm.commerce.tools.util.UIUtil" %>
<%@include file="../common/common.jsp" %>

<%! public String getUserLogon(String customerId, HttpServletRequest request) {
   try {
      if (customerId != null && !customerId.equals("")) {
         OptoolsRegisterDataBean userBean = new OptoolsRegisterDataBean();
         userBean.setUserId(customerId);
         DataBeanManager.activate(userBean, request);

         if (userBean.getLogonId() != null)
            return userBean.getLogonId();
      }
   } catch (Exception ex) {
      return "";
   }
   return "";
}
%>

<HTML>
<HEAD>
<%
   // obtain the resource bundle for display
   CommandContext cmdContextLocale =
     (CommandContext)request.getAttribute(com.ibm.commerce.server.ECConstants.EC_COMMANDCONTEXT);
   Locale jLocale = cmdContextLocale.getLocale();
   Hashtable orderLabels =
     (Hashtable)ResourceDirectory.lookup("order.orderLabels", jLocale);

      // retrieve request parameters
      JSPHelper jspHelp = new JSPHelper(request);
      String customerId =
        jspHelp.getParameter(ECOptoolsConstants.EC_OPTOOL_CUSTOMER_ID);
```

```
      if (customerId == null) {
         customerId = "";
      }

      //compose entire list of account names
      AccountListDataBean acctListDB = new AccountListDataBean();
      DataBeanManager.activate(acctListDB, request);
      AccountDataBean[] acctList = acctListDB.getAccountList();

%>
<link rel=stylesheet
   href="<%= UIUtil.getCSSFile(jLocale) %>" type="text/css">
<TITLE></TITLE>
<SCRIPT SRC="/wcs/javascript/tools/common/Util.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
<!---- hide script from old browsers

function initializeState()
{
   parent.setContentFrameLoaded(true);
}

function savePanelData()
{
}

function onLoad() {
   initializeState()
}

function isEmpty(id) {
   return !id.match(/[^\s]/);
}
function isNumber(word)
{
   var numbers="0123456789";
   for (var i=0; i < word.length; i++)
   {
      if (numbers.indexOf(word.charAt(i)) == -1)
      return false;
   }
   return true;
}
function formValid() {
   var invalidChars = /[,;&#%|"'\\/]/
   if (document.orderFindForm.orderId.value.match(invalidChars) ||
      document.orderFindForm.userLogon.value.match(invalidChars))
   {
      // alert(",;is not valid");
      alertDialog("<%=UIUtil.toJavaScript(orderLabels.get
               ("invalidChars").toString()) %>");
      return false;
   }
   return true;
}

function validateEntries()
{
   if (isEmpty(document.orderFindForm.orderId.value)
    && isEmpty(document.orderFindForm.userLogon.value)
    && (document.orderFindForm.orderState.value == "all")
    && isEmpty(document.orderFindForm.accountId.value)) {
      alertDialog('<%=UIUtil.toJavaScript((String)orderLabels.get("findDialogNoCriteria"))%>');
      return false;
   } else

   if (!isEmpty(document.orderFindForm.orderId.value)) {
      if (!isNumber(document.orderFindForm.orderId.value)) {
         alertDialog ('<%=UIUtil.toJavaScript((String)orderLabels.get
                  ("findDialogInvalidNumber"))%>');
         return false;
      }
   } else if (!formValid()) {
      return false;
   }

   return true;
}
function findAction() {
   if (validateEntries() == true) {
```

```
      url = '/webapp/wcs/tools/servlet/NewDynamicListView';
      var urlPara = new Object();
      urlPara.listsize='22';
      urlPara.startindex='0';
      if ("<%=customerId%>" != "") {
         urlPara.ActionXMLFile='order.csadminOrderListB2B';
      } else {
         urlPara.ActionXMLFile='order.csOrderListB2B';
      }
      urlPara.cmd='OrderListViewB2B';
      urlPara.orderId=document.orderFindForm.orderId.value;
      urlPara.userLogon=document.orderFindForm.userLogon.value;
      urlPara.accountId=document.orderFindForm.accountId.value;
      urlPara.orderType=document.orderFindForm.orderState.value;
      urlPara.orderby='orderid';

      top.setContent("<%= UIUtil.toJavaScript((String)orderLabels.get
                     ("findResultBCT")) %>",url,true, urlPara);
      return true;
   }
   return false;
}
function cancelAction() {
   top.goBack();
}
// -->
</SCRIPT>
</HEAD>
<BODY CLASS=content ONLOAD="initializeState();">
<H1><%=orderLabels.get("findDialog")%></H1>
<P><%=orderLabels.get("findCSOrderInst")%>
<FORM NAME="orderFindForm">
<TABLE>
   <TBODY>
   <TR>
      <TD><%=orderLabels.get("orderNumber")%></TD>
   </TR>
   <TR>
      <TD><INPUT size="9" type="text" maxlength="9" name="orderId"></TD>
   </TR>
   <TR>
      <TD></TD>
   </TR>
   <TR>
      <TD><%=orderLabels.get("customerName")%></TD>
   </TR>
   <TR>
      <TD><INPUT size="31" type="text" maxlength="31" name="userLogon"
                 value="<%=getUserLogon(customerId, request)%>"></TD>
   </TR>
   <TR>
       <TD></TD>
   </TR>
   <TR>
      <TD><%= orderLabels.get("orderStatus") %></TD>
   </TR>
   <TR>
      <TD>
         <SELECT name="orderState">
            <OPTION value="all"></OPTION>
            <OPTION value="P"><%= orderLabels.get("P") %></OPTION>
            <OPTION value="I"><%= orderLabels.get("I") %></OPTION>
            <OPTION value="W"><%= orderLabels.get("W") %></OPTION>
            <OPTION value="N"><%= orderLabels.get("N") %></OPTION>
            <OPTION value="M"><%= orderLabels.get("M") %></OPTION>
            <OPTION value="B"><%= orderLabels.get("B") %></OPTION>
            <OPTION value="C"><%= orderLabels.get("C") %></OPTION>
            <OPTION value="E"><%= orderLabels.get("E") %></OPTION>
            <OPTION value="R"><%= orderLabels.get("R") %></OPTION>
            <OPTION value="S"><%= orderLabels.get("S") %></OPTION>
            <OPTION value="D"><%= orderLabels.get("D") %></OPTION>
            <OPTION value="L"><%= orderLabels.get("L") %></OPTION>
            <OPTION value="T"><%= orderLabels.get("T") %></OPTION>
            <OPTION value="A"><%= orderLabels.get("A") %></OPTION>
            <OPTION value="F"><%= orderLabels.get("F") %></OPTION>
            <OPTION value="G"><%= orderLabels.get("G") %></OPTION>
            <OPTION value="X"><%= orderLabels.get("X") %></OPTION>
         </SELECT>
      </TD>
   </TR>
```

```
   <TR>
      <TD></TD>
   </TR>
   <TR>
      <TD><%= orderLabels.get("accountName") %></TD>
   </TR>
   <TR>
      <TD>
         <SELECT name="accountId">
            <OPTION value=""></OPTION>
            <%   if (acctList != null) {
                    for (int i=0; i<acctList.length; i++) { %>
                    <OPTION value="<%=acctList[i].getAccountId()%>">
                          <%=acctList[i].getAccountName()%></OPTION>
            <%      }
                 } %>
         </SELECT>
      </TD>
   </TR>
   <TR>
      <TD></TD>
   </TR>
   </TBODY>
</TABLE>
</FORM>
<SCRIPT LANGUAGE="JavaScript">
<!--
//For IE if (document.all) {
   onLoad();
}
//-->
</SCRIPT>

</BODY>
</HTML>
```

Chapter 3. Collaborative Workspaces

The elements discussed in this chapter present a case study that details the steps required to customize a QuickPlace, and create the ToolTech template shipped with WebSphere Commerce. It also outlines the steps required to customize the content of electronic mail (e-mail) sent when a user has been invited to join a collaborative workspace.

Customization Examples

The following examples outline how to customize this part of the WebSphere Commerce Accelerator.

Note: Scenarios 1 and 2 are closely related. If you perform scenario 1, you most often will want to follow through scenario 2 as well.

Scenario 1: Customize the workspace look and feel

Implemention overview

The first thing you might want to customize for the Collaborative Workspaces feature is the ″look and feel″ of the collaborative workspaces created. To customize the look and feel, you must create a template called a PlaceType. For more details about customization refer to the *Customizing QuickPlace* Redbook.

Customization steps

This example uses the ToolTech sample store shipped in WebSphere Commerce 5.4. This example illustrates how to create the QuickPlace:

1. Create a default QuickPlace by doing the following:
   a. Make sure that the QuickPlace server is running. Open the browser to `http://qp_server/quickplace` and log on as the QuickPlace administrator.
   b. Click **Create A QuickPlace**.
   c. Choose **Standard QuickPlace for Teams**. Fill in the fields as applicable, and click **NEXT**.
   d. Log on as the creator.
2. Update the Welcome page for the QuickPlace. Click **Edit**. Make the desired changes and click **PUBLISH**. This updates the Welcome page.
3. Change the Logo for the QuickPlace by doing the following:
   a. Click **Customize** in the table of contents.
   b. Click **Basics**.
   c. Click **Change Basics**.
   d. Set up the logo using either **Simple Text**, **Logo Maker**, or **Upload Logo Artwork**. The ToolTech sample used **Upload Logo Artwork**. The logo is now updated

Scenario 2: Create a custom theme

Implementation overview

A theme controls the ″look and feel″ of a QuickPlace. This example creates an Hypertext Markup Language (HTML) page and a cascading stylesheet (CSS) for

© Copyright IBM Corp. 2001

the layout and style of the QuickPlace. For more information about how to write those files refer to the *Customizing QuickPlace* Redbook.

Note:
Since member administration is managed through WebSphere Commerce, remove the **Members** link in the table of contents. To remove the link, use the following code in the Page Layout file when inserting the table of contents:

```
<QuickPlaceSkinComponent
   name=TOC
  Format={
  <tr>
  <td class=h-toc-text><br></td>
  <td class=h-toc-text><Item class=h-toc-text></td>
  <td class=h-toc-text><br></td>
  </tr>
  <tr>
  <td colspan=3><img src="../../../$resources.nsf/h_ResourcesByName/
                transparent.gif/$FILE/transparent.gif?OpenElement"
                width=1 height=3 alt="" border="0"></td>
  }
  SelectedFormat={
  <tr>
  <td class=h-tocSelected-text><br></td>
  <td class=h-tocSelected-text><Item class=h-tocSelected-text></td>
  <td class=h-tocSelected-text><br></td>
  </tr>
  <tr><td colspan=3><img src="../../../$resources.nsf/h_ResourcesByName/
                transparent.gif/$FILE/transparent.gif?OpenElement"
                width=1 height=3 alt="" border="0">
  </td>
  }
  EmptyFormat={}
  ReplaceString={Members=}>
```

Customization steps

1. Click **Customize** in the table of contents, **Custom Themes**, and then **New Theme**. Fill in the title, and upload the **Style Sheet** and **Page Layout** files.
2. Click **Next**.
3. Click **Customize** in the table of contents and then **Decorate**
4. Click **Choose a theme**.
5. Select the theme you just created and click **Next**. This should updates the look and feel of the QuickPlace.
6. Note that the **Members** link on the table of contents has disappeared.
7. Creating a PlaceType from a QuickPlace. You now can create a PlaceType based on the QuickPlace that you just customized. Click **Customize** on the table of contents and then **PlaceType Options**.
8. Click **Edit**. Choose **Yes** for **Allow PlaceTypes to be created from this QuickPlace?** and **No** for **Include current members of this QuickPlace in future QuickPlaces created from the PlaceType?**
9. Click **next**.
10. Open the browser to `http://`*`qp_server`*`/quickplace` and log on as the QuickPlace administrator.
11. Click **PlaceTypes**.
12. Click **CREATE PLACETYPE...**. Enter the name of the PlaceType and choose the QuickPlace upon which it you want it to be based.

13. Click **next**. You should see your new PlaceType.

Scenario 3: Customize the e-mail notification

Implementation overview

To change the default e-mail notification to a custom page for all stores, you need to modify the `CollabEmailContent.jsp` file so that it includes your desired content.

Customization steps

1. Modify the `CollabEmailContent.jsp` file. Locate the file in the following directory:

   AIX

   `/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wcstores.war`

   400

   `/QIBM/ProdData/WebAsAdv4/installedApps/WC_Enterprise_App_demo.ear/wcstores.war`

   Linux

   `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wcstores.war`

   Solaris

   `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wcstores.war`

   Windows

   drive`:\WebSphere\AppServerinstalledApps\WC_Enterprise_App_demo.ear\wcstores.war`

2. Customize the e-mail for a specific store by doing the following:
   a. Open a DB2 command window.
   b. Connect to your WebSphere Commerce database. To connect to the default database, use the following command:

      ```
      db2 connect to mall
      ```

   c. Execute the following DB2 command:

      ```
      db2 update viewreg set properties = 'docname=CollabEmailContent.jsp'
          where viewname = 'CollabEmailContentView'
      ```

   d. Place your customized template file `CollabEmailContent.jsp` into your store directory:

      AIX

      `/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wcstores.war/`*store_name*

      400

      `/QIBM/ProdData/WebAsAdv4/installedApps/WC_Enterprise_App_demo.ear/wcstores.war/`*store_name*

      Linux

      `/opt/WebSphere/AppServerinstalledApps/WC_Enterprise_App_demo.ear/wcstores.war/`*store_name*

      Solaris

      `/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wcstores.war/`*store_name*

      Windows

      drive`:\WebSphere\AppServer\installedApps\WC_Enterprise_App_demo.ear\`

`wcstores.war\`*`store_name`*
where*store_name* is the store directory name you choose when you published the store.

e. Restart your WebSphere Administration Server.

Chapter 4. Customer profiles

The elements discussed in this chapter represent the user interface to the Customer Profile components in the WebSphere Commerce Accelerator. They facilitate the creation and maintenance of customer profiles and all of the actions that a Seller or Merchant needs to perform their daily tasks associated with the business. The customer profiles component includes the following elements:

- Customer profile wizard
- Customer profile notebook

Sample code

This section refers to code samples contained in the zip file located at the following URL:

`ftp://ftp.software.ibm.com/software/websphere/commerce/54/profcust.zip`

To complete some sections in this chapter, you must download and install this sample code on a development machine running WebSphere Application Server 4.

Once downloaded, unzip this ZIP file into the following directory:

AIX `/usr/WebSphere/CommerceServer`

400 `/QIBM/UserData/WebSphere/CommerceServer`

Linux `/opt/WebSphere/CommerceServer`

Solaris `/opt/WebSphere/CommerceServer`

Windows *drive*`:\WebSphere\CommerceServer`

Customization examples

The following examples outline how to customize this part of the WebSphere Commerce Accelerator.

Scenario 1: Add an attribute to the customer profile

Implementation overview

In this example, the `FIELD1` column of the `USERS` table stores the beverage preference of the user. The following values will be valid:

- beer
- wine
- coffee
- tea
- water

Note that the changes only apply to a B2C store, and not to a B2B store.

An example demonstrates each of these steps. Locate this example in the `profcust` directory created when you installed the sample code.

© Copyright IBM Corp. 2001

Customization steps

1. Add a new page to the customer profiling notebook by creating a new JSP file. In our example, this page will display two options:
   - Ignore beverage preference
   - Target customers with one or more of the following beverage preferences:

   The preferences will appear as check boxes under the second option. Please inspect the accompanying JSP page, `BeveragePanel.jsp`, in the following directory, for additional details:

   AIX `/usr/WebSphere/CommerceServer/profcust/web/tools/segmentation/`

   400 `/QIBM/UserData/WebSphere/CommerceServer/profcust/web/tools/segmentation/`

   Linux `/opt/WebSphere/CommerceServer/profcust/web/tools/segmentation/`

   Solaris `/opt/WebSphere/CommerceServer/profcust/web/tools/segmentation/`

   Windows *drive*`:\WebSphere\CommerceServer\profcust\web\tools\segmentation\`
2. After you create the JSP file, you must add it to the `VIEWREG` table. Please inspect the accompanying SQL script, `beverage.sql`, in the following directory, for additional details:

   AIX `/usr/WebSphere/CommerceServer/profcust/schema/`

   400 `/QIBM/UserData/WebSphere/CommerceServer/profcust/schema/`

   Linux `/opt/WebSphere/CommerceServer/profcust/schema/`

   Solaris `/opt/WebSphere/CommerceServer/profcust/schema/`

   Windows *drive*`:\WebSphere\CommerceServer\profcust\schema\`
3. The new page must be recognized by the customer profile notebook, found in the following directory:

   AIX `/usr/WebSphere/CommerceServer/xml/tools/segmentation/`

   400 `/QIBM/UserData/WebSphere/CommerceServer/xml/tools/segmentation/`

   Linux `/opt/WebSphere/CommerceServer/xml/tools/segmentation/`

   Solaris `/opt/WebSphere/CommerceServer/xml/tools/segmentation/`

   Windows *drive*`:\WebSphere\CommerceServer\xml\tools\segmentation\` Make a copy of the `SegmentNotebook.xml`. In the sample, this new file's name is `MySegmentNotebook.xml`. The following element was added to the document:

   ```
   <panel name="segmentNotebookBeveragePanel"
          url="SegmentNotebookBeveragePanelView"
          group="segmentNotebookMiscPanelGroup" />
   ```

   You must include the new panel name in the segmentation properties file, located in the following directory:

   AIX `/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/properties/com/ibm/commerce/tools/segmentation`

   400 `/QIBM/ProdData/WebAsAdv4/installedApps/WC_Enterprise_App_demo.ear/properties/com/ibm/commerce/tools/segmentation`

Linux

/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/properties/com/ibm/commerce/tools/segmentation/

Solaris

/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/properties/com/ibm/commerce/tools/segmentation/

Windows

drive:\WebSphere\AppServer\installedApps/WC_Enterprise_App_demo.ear\properties\com\ibm\commerce\tools\segmentation\

In order for this to work, you will need to add the following line to Resources.properties:

```
segmentNotebookBeveragePanel=Preferred beverage
```

Optionally, copy MySegmentNotebook.xml from:

AIX

/usr/WebSphere/CommerceServer/profcust/xml/tools/segmentation/MySegmentNotebook.xml

400 /QIBM/UserData/WebSphere/CommerceServer/profcust/xml/tools/segmentation/MySegmentNotebook.xml

Linux /opt/WebSphere/CommerceServer/profcust/xml/tools/segmentation/MySegmentNotebook.xml

Solaris /opt/WebSphere/CommerceServer/profcust/xml/tools/segmentation/MySegmentNotebook.xml

Windows

drive:\WebSphere\CommerceServer\profcust\xml\tools\segmentation\MySegmentNotebook.xml

to:

AIX /usr/WebSphere/CommerceServer/xml/tools/segmentation/MySegmentNotebook.xml

400

/QIBM/UserData/WebSphere/CommerceServer/xml/tools/segmentation/MySegmentNotebook.xml

Linux /opt/WebSphere/CommerceServer/xml/tools/segmentation/MySegmentNotebook.xml

Solaris /opt/WebSphere/CommerceServer/xml/tools/segmentation/MySegmentNotebook.xml

Windows *drive*:\WebSphere\CommerceServer\xml\tools\segmentation\MySegmentNotebook.xml

4. Now you need to ensure that the new XML document is recognized. To do this, modify the Resources.xml file in the following directory:

   AIX /usr/WebSphere/CommerceServer/xml/tools/segmentation/

   400

   /QIBM/UserData/WebSphere/CommerceServer/xml/tools/segmentation/

   Linux /opt/WebSphere/CommerceServer/xml/tools/segmentation/

   Solaris /opt/WebSphere/CommerceServer/xml/tools/segmentation/

   Windows *drive*:\WebSphere\CommerceServer\xml\tools\segmentation\

   Modify the following element:

```
<XML name="SegmentNotebook"
     file="segmentation/SegmentNotebook.xml" />
```

Change the element to:

```
<XML name="SegmentNotebook"
     file="segmentation/MySegmentNotebook.xml" />
```

Save the file as MyResources.xml

Optionally, copy MyResources.xml from:

AIX

/usr/WebSphere/CommerceServer/profcust/xml/tools/segmentation/MyResources.xml

400 /QIBM/UserData/WebSphere/CommerceServer/profcust/xml/tools/segmentation/MyResources.xml

Linux /opt/WebSphere/CommerceServer/profcust/xml/tools/segmentation/ MyResources.xml

Solaris /opt/WebSphere/CommerceServer/profcust/xml/tools/segmentation/MyResources.xml

Windows

drive:\WebSphere\CommerceServer\profcust\xml\tools\segmentation\MyResources.xml

to:

AIX /usr/WebSphere/CommerceServer/xml/tools/segmentation

400 /QIBM/UserData/WebSphere/CommerceServer/xml/tools/segmentation

Linux /opt/WebSphere/CommerceServer/xml/tools/segmentation

Solaris /opt/WebSphere/CommerceServer/xml/tools/segmentation

Windows *drive*:\WebSphere\CommerceServer\xml\tools\segmentation

5. Point the application to MyResources.xml. Modify the demo.xml file in the following directory:

   AIX /usr/WebSphere/CommerceServer/instances/demo/xml/

   400 /QIBM/UserData/WebSphere/CommerceServer/instances/demo/xml/

   Linux /opt/WebSphere/CommerceServer/instances/demo/xml/

   Solaris /opt/WebSphere/CommerceServer/instances/demo/xml/

   Windows *drive*:\WebSphere\CommerceServer\instances\demo\xml\

   Look for:

   ```
   <resourceConfig file="segmentation/Resources.xml" />
   ```

   Change the element to:

   ```
   <resourceConfig file="segmentation/MyResources.xml" />
   ```

6. Extend the data bean that describes customer profiles, com.ibm.commerce.tools.segmentation.SegmentNotebookDataBean. In the sample, com.mycompany.tools.segmentation.MySegmentNotebookDataBean, extends the data bean to provide properties for the beverage attribute. Locate this sample, MySegmentNotebookDataBean.java, in the following directory:

   AIX

   /usr/WebSphere/CommerceServer/profcust/lib/com/mycompany/tools/segmentation/

400

/QIBM/UserData/WebSphere/CommerceServer/profcust/lib/com/mycompany/tools/segmentation/

Linux

/opt/WebSphere/CommerceServer/profcust/lib/com/mycompany/tools/segmentation/

Solaris

/opt/WebSphere/CommerceServer/profcust/lib/com/mycompany/tools/segmentation/

Windows

drive:\WebSphere\CommerceServer\profcust\lib\com\mycompany\tools\segmentation\

7. In order to get the segment notebook to recognize the new data bean, you will need to modify the XML that describes the customer profile notebook. Return to `MySegmentNotebook.xml` and notice the following change:

```
<databean name="segmentDetails"
    class="com.mycompany.tools.segmentation.SegmentNotebookDataBean" />
```

was changed to:

```
<databean name="segmentDetails"
    class="com.mycompany.tools.segmentation.MySegmentNotebookDataBean"
    stoplevel="2" />
```

8. Extend the controller command that saves the customer profile. This command is `com.ibm.commerce.tools.segmentation.SegmentSaveControllerCmd`. Please inspect the sample command implementation in `MySegmentSaveControllerCmdImpl` in the following directory:

AIX

/usr/WebSphere/CommerceServer/profcust/lib/com/mycompany/tools/segmentation/

400

/QIBM/UserData/WebSphere/CommerceServer/profcust/lib/com/mycompany/tools/segmentation/

Linux

/opt/WebSphere/CommerceServer/profcust/lib/com/mycompany/tools/segmentation/

Solaris

/opt/WebSphere/CommerceServer/profcust/lib/com/mycompany/tools/segmentation/

Windows

drive:\WebSphere\CommerceServer\profcust\lib\com\mycompany\tools\segmentation\

This example will construct a simple condition for the beverage condition.

In order to get the application to recognize this new command, you will need add it to the `CMDREG` table. Refer to `beverage.sql` from step `1a` above for more details.

9. Extend the task command that evaluates the customer profile, which is `com.ibm.commerce.membergroup.commands.CheckUserInMemberGroupCmd`. Inspect the sample command, `MyCheckUserInMemberGroupCmdImpl.java`, in the following directory, for details on how you to evaluate a new condition:

AIX

/usr/WebSphere/CommerceServer/profcust/lib/com/mycompany/membergroup/

commands/

400

/QIBM/UserData/WebSphere/CommerceServer/profcust/lib/com/mycompany/membergroup/commands/

Linux

/opt/WebSphere/CommerceServer/profcust/lib/com/mycompany/membergroup/commands/

Solaris

/opt/WebSphere/CommerceServer/profcust/lib/com/mycompany/membergroup/commands/

Windows

drive:\WebSphere\CommerceServer\profcust\lib\com\mycompany\membergroup\commands\

You also need to register the command in the CMDREG table. See beverage.sql for more details.

10. Extend the task command that displays the list of constraints for a customer profile. This command is com.ibm.commerce.tools.segmentation.SegmentConstraintListCmd. Inspect the sample command, MySegmentConstraintListCmdImpl, in the following directory for details on what you must change:

AIX

/usr/WebSphere/CommerceServer/profcust/lib/com/mycompany/tools/segmentation/

400

/QIBM/UserData/WebSphere/CommerceServer/profcust/lib/com/mycompany/tools/segmentation/

Linux

/opt/WebSphere/CommerceServer/profcust/lib/com/mycompany/tools/segmentation/

Solaris

/opt/WebSphere/CommerceServer/profcust/lib/com/mycompany/tools/segmentation/

Windows

drive:\WebSphere\CommerceServer\profcust\lib\com\mycompany\tools\segmentation\

You also need to register the command in the CMDREG table. See bevarage.sql for more details.

11. Add the following classes

- com.mycompany.membergroup.commands. MyCheckUserInMemberGroupCmdImpl
- com.mycompany.tools.segmentation.MySegmentConstraintListCmdImpl
- com.mycompany.tools.segmentation.MySegmentNotebookDataBean
- com.mycompany.tools.segmentation.MySegmentSaveControllerCmdImpl

located in the following directory:

AIX /usr/WebSphere/CommerceServer/profcust/lib

400 /QIBM/UserData/WebSphere/CommerceServer/profcust/lib

Linux /opt/WebSphere/CommerceServer/profcust/lib

Solaris /opt/WebSphere/CommerceServer/profcust/lib

Windows *drive*:\WebSphere\CommerceServer\profcust\lib

into the wcsmcruntime.jar file located in the following directory:

AIX

/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/lib

400

/QIBM/ProdData/WebAsAdv4/installedApps/WC_Enterprise_App_demo.ear/lib

Linux

/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/lib

Solaris

/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/lib

Windows

drive:\WebSphere\AppServer\installedApps/WC_Enterprise_App_demo.ear\lib

12. Copy the BeveragePanel.jsp file from step 1 into the following directory:

AIX

/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wctools.war/tools/segmentation

400

/QIBM/ProdData/WebAsAdv4/installedApps/WC_Enterprise_App_demo.ear/wctools.war/tools/segmentation

Linux

/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wctools.war/tools/segmentation

Solaris

/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wctools.war/tools/segmentation

Windows

drive:\WebSphere\AppServer\installedApps/WC_Enterprise_App_demo.ear\wctools.war\tools\segmentation

13. These changes require changes to your access control settings which are beyond the scope of this guide. Please refer to the *WebSphere Commerce Access Control Guide* available at the following URL:

www.ibm.com/software/webservers/commerce/wc_be/lit-tech-general.html

Chapter 5. Campaigns

Entering rules directly in the database

While the user interface is the easiest method with which to input rule data into the database, some situations may require that you manually input rules in the RULE column of the INITIATIVE table. Rules are defined using XML, and in this case, you must adhere to the document type definition (DTD), presented below:

The DTD for the RULE column of the INITIATIVE table is as follows:

```
<!DOCTYPE rule [
   <!ELEMENT rule (comment?, (orListCondition |andListCondition |simpleCondition |trueCondition |openCondition), action)>

   <!ELEMENT comment EMPTY>
   <!ATTLIST comment text CDATA #REQUIRED>

   <!ELEMENT action (parameter*)>
   <!ATTLIST action name CDATA #REQUIRED>

   <!ELEMENT orListCondition (not?, (orListCondition |andListCondition | simpleCondition | trueCondition |openCondition)+)>

   <!ELEMENT andListCondition (not?, (orListCondition |andListCondition | simpleCondition | trueCondition |openCondition)+)>

   <!ELEMENT simpleCondition (not?, variable, operator, value, qualifier*)>

   <!ELEMENT openCondition (not?, parameter*)>
   <!ATTLIST openCondition name CDATA #REQUIRED>

   <!ELEMENT trueCondition (not?)>

   <!ELEMENT not EMPTY>

   <!ELEMENT variable EMPTY>
   <!ATTLIST variable name CDATA #REQUIRED>

   <!ELEMENT operator EMPTY>
   <!ATTLIST operator name CDATA #REQUIRED>

   <!ELEMENT value EMPTY>
   <!ATTLIST value data CDATA #REQUIRED>

   <!ELEMENT qualifier EMPTY>
   <!ATTLIST qualifier name CDATA #REQUIRED>
   <!ATTLIST qualifier data CDATA #REQUIRED>

   <!ELEMENT parameter (parameter*)>
   <!ATTLIST parameter name CDATA #REQUIRED>
   <!ATTLIST parameter value CDATA #REQUIRED>
]>
```

simpleCondition elements

The default implementation of initiatives supports the following simpleCondition elements. You may combine these in any combination with orListCondition, andListCondition, and openCondition elements to create the condition portion of a rule.

Table 2.

Variable name	Supported operators	Supportedvalues	Qualifiers
segment	= !=	The name of a customer profile.	none
shoppingCartSku	= !=	A product SKU.	none
shoppingCartCategory	= !=	A product catagory name.	language
purchaseHistorySku	= !=	A product SKU.	none
purchaseHistoryCategory	= !=	A product category name.	language

© Copyright IBM Corp. 2001

Table 2. (continued)

Variable name	Supported operators	Supportedvalues	Qualifiers
shoppingCartTotal	= != > < >= <=	A decimal value.	currency
dayOfWeek	= !=	SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY	none

openCondition elements

The default implementation supports the following openCondition elements. You may combine these in any combination with orListCondition, andListCondition, and simpleCondition elements to create the condition portion of a rule.

Table 3.

Open Condition Name	Supported Parameters
shoppingCartTotal	comparisonType - may be one of ″>″, ″<″, or ″=″. value - a decimal value currency
shoppingCartCategory	comparisonType - may be one of ″=″ or ″!=″ value - name of the category language
shoppingCartSku	comparisonType - may be one of ″>″, ″<″, or ″=″ value - a decimal value currency
purchaseHistoryCategory	comparisonType - may be one of ″=″ or ″!=″ value - name of a category currency
purchaseHistorySku	comparisonType - may be one of ″>″, ″<″, or ″=″ value - a decimal value currency

Action elements

The default implementation supports the following action elements. These may be with any condition elements to construct a rule.

Table 4.

Action Name	Supported Parameters	Supported Values	Supported Subparameters
suggestiveSell	queryOperator	or and	none
	queryType	skuQuery genericQuery	none
	sku	A product SKU	none
	skuPrefix	A product SKU prefix	none
	category	A product category name	language
	productDescription	A product category description	language
	inventoryQuantity	An integer	comparisonOperator
	listPrice	A decimal value	comparisonOperator currency
	availabilityDate	A date of the form yyyy-mm-dd-00.00.00	comparisonOperator
collaborativeFiltering	none		
awarenessAd	collateral	An advertisement name	
discountAd	discountCollateral	An advertisement name	Discount identifier
couponAd	couponCollateral	An advertisement name	
categoryRecommendation	category	Name of a category	

The following example initiative shows a spring ad to men and a fall ad to women:

```
<ruleSet>
   <rule>
      <comment text="show spring ad to men"/>
      <simpleCondition>
         <variable name="segment"/>
         <operator name="="/>
         <value data="men"/>
      </simpleCondition>
      <action name="awarenessAd">
         <parameter name="collateral" value="Spring">
         </parameter>
      </action>
   </rule>
   <rule>
      <comment text="show fall ad to women"/>
      <simpleCondition>
         <variable name="segment"/>
         <operator name="="/>
         <value data="women"/>
      </simpleCondition>
      <action name="awarenessAd">
         <parameter name="collateral" value="Fall">
```

```
        </parameter>
      </action>
    </rule>
</ruleSet>
```

Infrastructure elements

Rule package

The campaigns component makes use of the `com.ibm.commerce.rule` package. This package provides classes which help you convert a set of rules to and from XML format. It also provides support for calling the rules in the rule set.

Table 5.

Class/Interface name	Description
Action	The Action class provides properties for the name and parameters used by the action part of a rule. The parameters have names and values and optionally they may also have subparameters.
ActionHandler	The ActionHandler interface must be implemented in order to provide the implementation of the action part of a rule. There is one method ″performAction″ that is called when the condition portion of the rule evaluates to true. The appropriate instance of the Action class will be passed into the implementation of the ″performAction″ method.
Rule	The Rule class provides methods that help you convert an instance of Rule to and from XML format. The Rule class has three properties. One for the condition part of the rule. One for the action part of the rule. And one for the comment. The condition property must be an instance of the Condition class that can be found in the com.ibm.commerce.condition package. The Rule class also has a method ″invoke″ which accepts an implementation of the Evaluator interface and the ActionHandler interface. If the condition evaluates to true, then the ″performAction″ method of the ActionHandler implementation is invoked.
RuleConstants	The RuleConstants interface contains some public constants that are used by the rule package.
RuleSet	The RuleSet class provides methods that help you convert an instance of RuleSet to and from XML format. It has a property that is an array of Rule objects. It also has an invoke method that accepts an implementation of the Evaluator and ActionHandler interfaces. When the invoke method is called, it loops through the array of Rule objects and invokes one after the other.

Campaigns use the rule package to save and load the rule portion of the campaign initiative to and from XML format. It also helps invoke the rules. The `CampaignInitiativeEvaluateCmd` task command provides the implementation of the `ActionHandler` interface that is used to handle the campaign initiative actions.

Condition package

The rule package makes use of the `com.ibm.commerce.condition` package. This package provides classes which help you convert a boolean condition to and from XML format. It also provides support for evaluating a condition.

Table 6.

Class/Interface name	Description
AndListCondition	The AndListCondition class extends the ConditionList class to provide a condition that consists of a list of conditions joined by the AND boolean operator.
Condition	The Condition class is an abstract class which provides methods which help you to convert to and from XML format. It also provides an abstract method that can be used to evaluate the condition.
ConditionConstants	The ConditionConstants interface provides constant values that are used by the condition package.
ConditionList	The ConditionList class is an abstract class that extends the Condition class. It provides support for a condition that consists of a list of conditions.
ConditionUtil	The ConditionUtil class provides a number of static methods that can be used during condition evaluation.
Evaluator	The Evaluator interface must be implemented in order to evaluate a simple condition or an open condition.
OpenCondition	The OpenCondition class extends the Condition class to provide a generic condition. This condition consists of a list of name value pairs.
OrListCondition	The OrListCondition class extends the ConditionList class to provide a condition that consists of a list of conditions joined by the OR boolean operator.
SimpleCondition	The SimpleCondition class extends the Condition class to provide a condition of the form Variable, operator, value.
TrueCondition	The TrueCondition class extends the Condition class to provide a condition that always evaluates to true.

The rule package uses the condition package to save and load the condition portion of the rules to and from XML format. It also helps to evaluate the conditions.

There are five concrete extensions of the abstract Condition class. Three of these (AndListCondition, OrListCondition, and TrueCondition) know how to evaluate

themselves. The remaining two (OpenCondition and SimpleCondition) require an implementation of the Evaluator interface in order for them to be evaluated. The CampaignInitiativeEvaluateCmd task command provides the implementations of the Evaluator interface that is used to evaluate campaign initiative conditions.

Blaze rule project

The default implementation of campaign initiatives makes use of a Blaze rule project that helps to evaluate the campaign initiative rules. By default, all of the open conditions cause the Blaze rule project to be invoked by the **CampaignInitiativeEvaluateCmd** task command. Locate the project, `CampaignInitiativeEvaluator`, in the following directory:

AIX `/usr/WebSphere/CommerceServer/rules/campaigns`

400 `/QIBM/UserData/WebSphere/CommerceServer/rules/campaigns`

Linux `/opt/WebSphere/CommerceServer/rules/campaigns`

Solaris `/opt/WebSphere/CommerceServer/rules/campaigns`

Windows *drive*`:\WebSphere\CommerceServer\rules\campaigns`

The default implementation of this project is capable of evaluating all of the open conditions that are created by the campaign initiative user interface. You can add support for additional open conditions to this project.

Customization

It is expected that our customers will want to extend the default implementation of campaign initiatives to add more supported conditions and possibly additional actions and display types.

Adding run time support for new initiative conditions

The blaze project that evaluates campaign initiative conditions accepts an instance of the **CampaignInitiativeContext**. This context provides access to the command context as well as the name and parameters of the open condition. To add support for a new open condition, you will need to add a new rule to the `conditionEvaluator` rule. This rule will need to check the name of the new condition and evaluate the condition. During the evaluation of the condition, the you can access the open condition parameters by name through the context object.

Adding build time support for new initiative conditions

In order to allow the marketing manager to take advantage of new conditions, the campaign initiative user interface will need to be extended to provide support for the new condition. This can be done by adding a new condition to the Which panel of the campaign initiative wizard or notebook (`WhenAddPanel.jsp`). You will also need to extend the **CampaignInitiativeSaveControllerCmdImpl** class, in the `com.ibm.commerce.tools.campaigns` package, to construct the new Condition object. Locate the file in the following directory:

AIX

`/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wctools.war/tools/campaigns/`

400

`/QIBM/ProdData/WebAsAdv4/installedApps/WC_Enterprise_App_demo.ear/wctools.war/tools/campaigns/`

Linux

`/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wctools.war/tools/campaigns/`

Solaris

/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wctools.war/tools/campaigns/

Windows

drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_demo.ear\wctools.war\tools\campaigns\

Adding run time support for new campaign initiative actions

In order to provide support for new campaign initiative actions, you will need to extend the **CampaignInitiativeEvaluateCmdImpl** class, in the `com.ibm.commerce.tools.campaigns` package, and override the **performAction** method. If the name of the action matches your new action name, then you should perform the action, otherwise you should call the super method to perform the default implementation.

Adding build time support for new campaign initiative actions

In order to allow the marketing manager to take advantage of the new campaign initiative actions, you will need to update the What panel of the campaign initiative wizard or notebook (`InitiativeWhatPanel.jsp`). You will also need to extend the **CampaignInitiativeSaveControllerCmdImpl** class to construct the new Action object. Locate the file in the following directory:

AIX

/usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wctools.war/tools/campaigns/

400

/QIBM/ProdData/WebAsAdv4/installedApps/WC_Enterprise_App_demo.ear/wctools.war/tools/campaigns/

Linux

/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wctools.war/tools/campaigns/

Solaris

/opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_demo.ear/wctools.war/tools/campaigns/

Windows

drive:\WebSphere\AppServer\installedApps\WC_Enterprise_App_demo.ear\wctools.war\tools\campaigns\

Adding support for new e-Marketing spot types

If you find that you need to provide additional output types, you can extend the `EMarketingSpot` class, in the `com.ibm.commerce.marketing.beans` package, to provide a new property that can be used by the page designer. You will also need to extend the **CampaignInitiativeEvaluateCmdImpl** class, in the `com.ibm.commerce.tools.campaigns` package, to return the new output type.

Chapter 6. Catalog search

The elements discussed in this chapter describe activities that are needed to extend the existing search functionality. The additional functionality includes guidance on introducing new attributes and tables accessible by the search component. Optimizing the schema to help reduce the cost to the run time can also improve the performance of the search components. This component includes the following customizable elements:

- Search bean
- Search engine
- Summary table definition scripts

Table 8 summarizes the list of database columns that are searched by the catalog search bean. Due to varying site requirements, you will probably find it necessary to provide search on a column that is not described in this table.

Customization scenarios

To add a search attribute to search bean where these attributes are already available from the search engine, implement scenario 1 by adding the attribute to search bean. If the search engine does not support search on this attribute, but it is available in one of the tables it already searches with, implement scenario 2 to add the attribute to the engine. If this attribute is not in one of the tables the search engine searches, implement scenario 3 to add the table to the search engine. The procedure to add rich attributes to search bean is the same as adding any other attribute to search bean (implement scenario 1), but differs slightly in implementation. Refer to scenario 4. To help improve run time performance, implement scenario 5 to modify and optimize summary table definition scripts by creating one table for each category to reduce the number of rows searched. The following scenarios outline when to customize this component:

Scenario 1: Add attribute to search bean
Introduce new attribute to bean where it is already available in search engine.

Scenario 2: Add attribute to search engine
Introduce new attribute to search engine where it already searches with that table.

Scenario 3: Add table to search engine
Introduce new table to search engine.

Scenario 4: Add rich attribute to search bean
Introduce rich attribute to bean.

Scenario 5: Improve performance with optimized summary tables
Use knowledge of the data set to optimize summary table definition.

Note: References to the Search Engine, the Search Interface, and the Unified Search Framework are synonymous and used interchangeably in the documentation.

© Copyright IBM Corp. 2001

Scenario 1: Add attribute to search bean

Implementation overview

The first thing you may want to customize is to add another searchable attribute. This activity requires changes to the search bean and optionally the search engine if this attribute is not available there either (described in Scenario 2). Customization of search bean requires sub-classing it and extending existing methods. Sample code on how to do this is provided in the `samples/search` directory.

Customization steps

This example shows how to add additional search attributes to the catalog search bean, for which the metadata classes exist in the `com.ibm.search.catalog` package. The catalog search data bean is a class. To customize it, do the following:

1. Create a subclass of the `CatEntrySearchListDataBean` class.
2. Declare variables for new searchable attributes that you want to add. You can declare two types of variables to store the information associated with a searchable attribute. The first type may be used to store the value for the attribute. The second type may contain related information that can be used in search like search operator, boolean search, case sensitive. The information stored in these variables may be used in generation of SQL queries that query catalog information.

   Note: Database columns that are used in the generation of SQL queries must be previously defined in the Search Interface RuleQuery class or in one of its sub-classes (see 'Search Interface Customization') before they are available as constraints in a generated query.
3. Create accessors (get and set methods) for the declared variables.
4. Create a `populate()` method which does the following:
   a. Instantiate either `RuleQuery` or a subclass of `RuleQuery`.
   b. Reference this instance of `RuleQuery` to the instance in the parent bean class using the `setRuleQuery(ruleQueryInstance)` method.
   c. Call the `super.setPredefinedAttributes()` method.
   d. Formulate a `<Predicate>` for the new searchable constraints that use the Search Interface.
   e. Use `super.setIsAllNull(false)` to notify the parent that a new search attribute has been added.
   f. Call the `super.execute()` method.

Example

This example shows how to add new searchable attributes onSpecial to catalog search bean assuming that the metadata classes exist in the `com.ibm.search.catalog` package.:

1. Create a new subclass.
2. Create new variables for the searchable attributes.

```
public class CustomCatEntrySearchListDataBean
        extends CatEntrySearchListDataBean {
  private static final String COPYRIGHT =
        com.ibm.commerce.copyright.IBMCopyright.SHORT_COPYRIGHT;
  protected java.lang.String onSpecial;
  protected java.lang.String onSpecialOperator;
  }
```

The `onSpecial` variable is used to store the value of searchable attributes. The `onSpecialOperator` variable is used to store the search operator (for example; like, =, <, >, <=, >=).

3. Create accessors for all the variables

```
public java.lang.String getOnSpecial(){
      return onSpecial;
}

public void setOnSpecial(java.lang.String newOnSpecial) {
  onSpecial = newOnSpecial;
}
```

4. Create a `populate()` method

```
public void populate() throws Exception {
   String langId = commandContext.getLanguageId().toString();
   RuleQuery ruleQueryInstance = new RuleQuery();
   setRuleQuery(ruleQueryInstance);
   super.setPredefinedAttributes();
      if(onSpecial != null){
      ruleQueryInstance.addSelectAttribute(ruleQueryInstance.CATENTRY_ONSPECIAL_Attr,
  getNumeric(onSpecialOperator), onSpecial);
      ruleQueryInstance.addSelectOperator(ruleQueryInstance.AND_Operator);
      }

   super.setIsAllNull(false);
   }
   q.addSelectOperator(q.AND_Operator);
   }
   super.execute(); //Step 4f
}
```

Scenario 2: Add an attribute to the search engine

Implementation overview

The search engine requires information about attributes and tables to properly formulate queries to retrieve results from the catalog. This information is kept as metadata. To extend the search engine to add new searchable attributes, additional attribute metadata and optionally table metadata definitions are needed. Table metadata is only required to introduce a new table to the search engine, and further details are provided in Scenario 3. Upon completion of this scenario, you must implement Scenario 1: Add Attribute to Search Bean to make it available to your JSP page designer. Sample code on how to do this is provided in `samples/search` directory.

Customization steps

The search engine is part of the unified search framework. It is represented as a class, named`RuleQuery` and additional metadata classes such as `AttributeInfo` and `TableInfo` describing the column and table names respectively. This example shows how to add new attributes to search engine by creating metadata classes for database columns that are not in the `com.ibm.search.catalog` package (but metadata for the table to which the column belongs does exist). To customize the search engine, do the following:

1. Define Search Interface metadata for each column and table you want to make searchable. This requires the following:
   a. Each searchable table must have a corresponding class that is a subclass of the `TableInfo` class. This subclass must specify the table name.
   b. Each searchable column must have a corresponding class that is a subclass of the `AttributeInfo` class. This subclass must specify the column name, the `TableInfo` sub-class to which the column belongs, and the SQL data type of the column.
2. Create a subclass of the `RuleQuery` class, and define static integer references for each new database column. Note that integer values 0 to 1000 are reserved for the system.
3. Create a `findAttributeInfoName()` method. To prevent this method from overriding the parent `findAttributeInfoName()` method it is recommended that

you call the `super.findAttributeInfoName()` method. If the parent `findAttributeInfoName()` is overridden, then the database columns defined in the parent class will not be available.

4. Add factory class creation logic to this method to create an `AttributeInfo` metadata class for each new database column.
5. Modify the `search.xml` file adding predefined table join relationships for any new searchable tables. Join relationships are required for all table combinations. To modify the `search.xml`, do the following:
   a. Add a `COMMENT` section to describe the table join relationship that is formulated by the entry.
   b. Add a `KEY` section to uniquely identify the entry. The table that are joined is used as unique key
   c. Add a `VALUE` section to specify values that are associated with a key. The entry in the value section defines a join relationship. The column names in the value section are defined using `AttributeInfo` subclass of that column.

Example

The following example will create a new searchable database column `CATENTRY.ONSPECIAL`:

1.
   a. Create a subclass of the `TableInfo` class to specify the table name `CATENTRY` (if the class for this table does not exist):

```
public final class CatEntryTableInfo extends TableInfo
{
   private static final String COPYRIGHT =
      com.ibm.commerce.copyright.IBMCopyright.SHORT_COPYRIGHT;

   // singleton idiom in Java
   private static CatEntryTableInfo info = new CatEntryTableInfo ();

   /**
    * The constructor set's the table name.
    */
         public CatEntryTableInfo() {
            super("CATENTRY");
         }
   /**
    * The method returns the sigleton idiom.
    * @return com.ibm.commerce.search.catalog.CatEntryTableInfo
    */
         public static CatEntryTableInfo getSingleton()
         {
            return info;
         }
}
```

   b. Create a subclass of the `AttributeInfo` class to specify the column name ONSPECIAL:

```
public final class CatEntryOnSpecialAttributeInfo extends AttributeInfo
{
   private static final String COPYRIGHT =
      com.ibm.commerce.copyright.IBMCopyright.SHORT_COPYRIGHT;

   // singleton idiom in Java
   private static CatEntryOnSpecialAttributeInfo info = new
         CatEntryOnSpecialAttributeInfo(CatEntryTableInfo.getSingleton());

   /**
    * The constructor set's the column name, column's table name, and datatype.
    */
   public CatEntryOnSpecialAttributeInfo(TableInfo info) {
      super(info);
      columnName = "ONSPECIAL";
      sqltype = SQLTYPE_NUM;
   }

   /**
    * The method returns the sigleton idiom.
    * @return com.ibm.commerce.search.catalog.CatEntryOnSpecialAttributeInfo
    */
   public static CatEntryOnSpecialAttributeInfo getSingleton()
   {
      return info;
   }
}
```

2. Create a subclass of search interface RuleQuery:

```
public class CustomQuery extends RuleQuery {
   private static final String COPYRIGHT =
      com.ibm.commerce.copyright.IBMCopyright.SHORT_COPYRIGHT;

   public final static int CATENTRY_ONSPECIAL_Attr = 1001; //Static reference
}
```

3. Create findAttributeInfoName() method:

```
protected String findAttributeInfoName(int attrId) {
   String className;
      className = super.findAttributeInfoName(attrId);
      switch (attrId) {
         case CATENTRY_ONSPECIAL_Attr:
         className = new String("CatEntryOnSpecialAttributeInfo");
         break;
      }
      return className;
}
```

4. Modify Search.xml if necessary (in this example it is not necessary to modify search.xml file)

Note: To add this new metadata to search bean follow steps in scenario 1. But you must use an instance of the sub-query of RuleQuery that has metadata information (refer to CustomQuery in the above example) instead of an instance of the RuleQuery class.

Scenario 3: Add table to search engine

Implementation overview

To add attributes from a new table not currently supported by the search engine, you must define information about the table and describe how this table relates to other tables used by the engine. This information is the join relationships used to derive join predicates relating each table to other tables and is kept as metadata. To extend the search engine to add new searchable tables, you must define each table and its relationship to other tables.

You must implement scenario 2: ″Add Attribute to Search Engine″ to make attributes related to this table available to the search bean creator. Sample code on how to do this is provided in samples/search directory.

Customization steps

These examples show how to create join relationship metadata classes for database tables. Typically, you should define one join relationship for every logical table relationship combination.

Example

This example changes an existing search.xml to include a new join relationship entry between the CATENTRY table and the OFFERPRICE table:

1. Add COMMENT section:

```
<--  ***** WWWW *****
   setW.add("OFFER.CATENTRY_ID = CATENTRY.CATENTRY_ID");
   setW.add("OFFER.OFFER_ID = OFFERPRICE.OFFER_ID");
-->
```

2. Add KEY and VALUE sections:

```
<entry>
   <key1>OFFERPRICE</key1>
   <key2>CATENTRY</key2>
   <value>
```

```
      <attrinfo1>OfferOfferIdAttributeInfo</attrinfo1>
      <attrinfo2>OfferPriceOfferIdAttributeInfo</attrinfo2>
   </value>
   <value>
      <attrinfo1>CatEntryIdentifierAttributeInfo</attrinfo1>
      <attrinfo2>OfferCatentryIdAttributeInfo</attrinfo2>
   </value>
</entry>
```

Scenario 4: Add rich attributes to search bean

Implementation overview

The process to add search on rich attributes to the bean is same as adding any other searchable attribute to the bean. Once the data set is known and identified any rich attributes, follow scenario 1 to make these attributes available in search bean. The only difference between this scenario and scenario 1 is in the method parameters used to construct the rich attribute predicate (see the text in bold in the example below for details). The search engine already has the capability to search for rich attributes that are defined in ATTRIBUTE and ATTRVALUE tables.

Customization steps

Same as customization scenario 1, Add Attribute to search bean.

Example

This example shows how to add a new searchable rich attribute, `color`, to the catalog search bean..

1. Create a new subclass.
2. Create new variables for the searchable attributes.

```
public class ExtendedCatEntrySearchListDataBean
        extends CatEntrySearchListDataBean implements
        ExtendedCatEntrySearchListInputDataBean,
        ExtendedCatEntrySearchListSmartDataBean {
  private static final String COPYRIGHT =
        com.ibm.commerce.copyright.IBMCopyright.SHORT_COPYRIGHT;
  protected java.lang.String colorValue;
  protected java.lang.String colorValueCaseSensitive;
  protected java.lang.String colorValueOperator;
  }
```

   The variable `colorValue` stores the value of searchable attributes. The variable `colorValueOperator` is stores the search operator (for example; like, =, <, >, <=, >=).

3. Create accessors for all the variables

```
public java.lang.String getColorValue(){
      return colorValue;
}

public void setColorValue(java.lang.String newColorValue) {
  colorValue = newColorValue;
}
```

4. Create a `populate()` method

```
public void populate() throws Exception {
   String langId = commandContext.getLanguageId().toString();
   RuleQuery ruleQueryInstance = new RuleQuery();
   setRuleQuery(ruleQueryInstance);
   super.setPredefinedAttributes();
      if(isEmpty(colorValue)){
      if(colorValueCaseSensitive ==null || !colorValueCaseSensititve.equals(CASE_SENSITIVE)){

        ruleQueryInstance.addSelectAttribute("Color",
     getStringOperator(colorValueOperator),
          colorValue.toUpperCase(),
```

```
            ruleQueryInstance.ATTRVALUE_STRINGVALUE_Attr,langId, null, ruleQueryInstance.UPPER_Function);
          super.setIsAllNull=false;
          }else{
          ruleQueryInstance.addSelectAttribute("Color",
       getStringOperator(colorValueOperator),
            colorValue,
            ruleQueryInstance.ATTRVALUE_STRINGVALUE_Attr,langId, null);
          super.setIsAllNull=false;
          }
          ruleQueryInstance.addSelectOperator(q.AND_Operator);
      }
    super.setIsAllNull(false);

    q.addSelectOperator(q.AND_Operator);
    }
    super.execute(); //Step 4f
    }
```

Scenario 5: Improve performance with optimized summary tables

Implementation overview

When the data set is defined, this knowledge can be used to create more optimal summary tables to significantly improve search performance. The goal is to create tables which resemble the anticipated query types to be encountered. In most situations these summary tables will be much smaller, and thus, queries applied over these tables will be faster than using the larger underlying native tables or other less optimal summary tables.

Customization steps

This example shows how to create a rich attribute category group summary table for each category group. If there are many category groups (CATGROUPS) that have items with rich attributes, define a summary table definition for each group.

Steps to create this summary table:

1. Define a table create statement with a select query to be used as your summary table definition. This select statement must have the same columns specified in a "group by" clause as listed in the result set. For more details, see the DB2 documentation for "Automatic Summary Tables" (AST), or the Oracle documentation for "Materialized Views."
2. In the select statement, specify a category group predicate for the category.
3. Create index with the same result set columns and in the same order as specified in the above query.
4. Repeat the above steps for each category group.

Example

This example shows a summary table creation script for rich attribute with category groups for each of three category groups 10000, 10001, and 10002. The bold parts are new compared to the general definition shown in file wcs.summary.richAttributeCatGroup.create.sql from which this example is based on.

```
// Summary table definition for catgroup 10000
CREATE TABLE RICHATTRCG0 AS
(
  SELECT ATTRIBUTE.NAME,
         ATTRIBUTE.LANGUAGE_ID,
         ATTRVALUE.CATENTRY_ID,
         ATTRVALUE.FLOATVALUE,
         ATTRVALUE.INTEGERVALUE,
         ATTRVALUE.STRINGVALUE,
         CATGPENREL.CATGROUP_ID,
         COUNT(*) AS C5
```

```
  FROM ATTRIBUTE, ATTRVALUE, CATGPENREL
  WHERE ATTRIBUTE.ATTRIBUTE_ID=ATTRVALUE.ATTRIBUTE_ID
         AND ATTRIBUTE.CATENTRY_ID=CATGPENREL.CATENTRY_ID
         AND CATGPENREL.CATGROUP_ID = 10000
  GROUP BY ATTRIBUTE.NAME,
           ATTRIBUTE.LANGUAGE_ID,
           ATTRVALUE.CATENTRY_ID,
           ATTRVALUE.FLOATVALUE,
           ATTRVALUE.INTEGERVALUE,
           ATTRVALUE.STRINGVALUE,
           CATGPENREL.CATGROUP_ID
)DATA INITIALLY DEFERRED REFRESH IMMEDIATE NOT LOGGED INITIALLY;
ALTER TABLE RICHATTRCG0 ACTIVATE NOT LOGGED INITIALLY;
REFRESH TABLE RICHATTRCG0;

commit;

SET CURRENT REFRESH AGE = ANY;

CREATE INDEX I_RICHATTRCG0 ON RICHATTRCG0
(
   NAME           ASC,
   LANGUAGE_ID    ASC,
   CATENTRY_ID    ASC,
   FLOATVALUE     ASC,
   INTEGERVALUE   ASC,
   STRINGVALUE     ASC,
   CATGROUP_ID     ASC
);

commit;

// Summary table definition for catgroup 10001
CREATE TABLE RICHATTRCG1 AS
(
  SELECT ATTRIBUTE.NAME,
         ATTRIBUTE.LANGUAGE_ID,
         ATTRVALUE.CATENTRY_ID,
         ATTRVALUE.FLOATVALUE,
         ATTRVALUE.INTEGERVALUE,
         ATTRVALUE.STRINGVALUE,
         CATGPENREL.CATGROUP_ID,
         COUNT(*) AS C5
  FROM ATTRIBUTE, ATTRVALUE, CATGPENREL
  WHERE ATTRIBUTE.ATTRIBUTE_ID=ATTRVALUE.ATTRIBUTE_ID
        AND ATTRIBUTE.CATENTRY_ID=CATGPENREL.CATENTRY_ID
        AND CATGPENREL.CATGROUP_ID = 10001
  GROUP BY ATTRIBUTE.NAME,
           ATTRIBUTE.LANGUAGE_ID,
           ATTRVALUE.CATENTRY_ID,
           ATTRVALUE.FLOATVALUE,
           ATTRVALUE.INTEGERVALUE,
           ATTRVALUE.STRINGVALUE,
           CATGPENREL.CATGROUP_ID
)DATA INITIALLY DEFERRED REFRESH IMMEDIATE NOT LOGGED INITIALLY;
ALTER TABLE RICHATTRCG1 ACTIVATE NOT LOGGED INITIALLY;
REFRESH TABLE RICHATTRCG1;

commit;
```

```
SET CURRENT REFRESH AGE = ANY;

CREATE INDEX I_RICHATTRCG1 ON RICHATTRCG1
(
   NAME           ASC,
   LANGUAGE_ID    ASC,
   CATENTRY_ID    ASC,
   FLOATVALUE     ASC,
   INTEGERVALUE   ASC,
   STRINGVALUE    ASC,
   CATGROUP_ID    ASC
);

commit;

// Summary table definition for catgroup 10002
CREATE TABLE RICHATTRCG2 AS
(
  SELECT ATTRIBUTE.NAME,
         ATTRIBUTE.LANGUAGE_ID,
         ATTRVALUE.CATENTRY_ID,
         ATTRVALUE.FLOATVALUE,
         ATTRVALUE.INTEGERVALUE,
         ATTRVALUE.STRINGVALUE,
         CATGPENREL.CATGROUP_ID,
         COUNT(*) AS C5
  FROM ATTRIBUTE, ATTRVALUE, CATGPENREL
  WHERE ATTRIBUTE.ATTRIBUTE_ID=ATTRVALUE.ATTRIBUTE_ID
        AND ATTRIBUTE.CATENTRY_ID=CATGPENREL.CATENTRY_ID
        AND CATGPENREL.CATGROUP_ID = 10002  GROUP BY ATTRIBUTE.NAME,
           ATTRIBUTE.LANGUAGE_ID,
           ATTRVALUE.CATENTRY_ID,
           ATTRVALUE.FLOATVALUE,
           ATTRVALUE.INTEGERVALUE,
           ATTRVALUE.STRINGVALUE,
           CATGPENREL.CATGROUP_ID
)DATA INITIALLY DEFERRED REFRESH IMMEDIATE NOT LOGGED INITIALLY;
ALTER TABLE RICHATTRCG2 ACTIVATE NOT LOGGED INITIALLY;
REFRESH TABLE RICHATTRCG2;

commit;

SET CURRENT REFRESH AGE = ANY;

CREATE INDEX I_RICHATTRCG2 ON RICHATTRCG2
(
   NAME              ASC,
   LANGUAGE_ID       ASC,
   CATENTRY_ID       ASC,
   FLOATVALUE        ASC,
   INTEGERVALUE      ASC,
   STRINGVALUE       ASC,
   CATGROUP_ID       ASC
);

commit;
```

Catalog search data bean variables

List of variables that must be passed to Catalog search bean through the URL (JSP.)

Table 7.

Name	Data Type	Description
beginIndex	string	This variable is used for paging the result set. The value must be the index of the first result row in a page.
catGroupId	string	The value of this variable is used in search on a category ID.
categoryTerm	string	The value of this variable is used in searches on either category name or descriptions or both.
categoryTermCaseSensitive	string	A user can choose case sensitive or case insensitive search. The value in this variable is used to identify if a search is case sensitive or not. The value must be either `yes`, for case sensitive searches, or `no`, for case insensitive searches.
categoryTermOperator	string	A user can choose either `like` or `equal` as search operators. The value in this variable is used to store a user's choice. The value must be either `LIKE`, for the like operator, or `EQUAL`, for equal operator.
categoryTermScope	Integer	A user can restrict the scope of a search to one of name; name and short description; or name, short description, and long description. The value in this variable is used to store a user's choice. The value must be one of `1`for name and short description, or `2` for name only, or `3` for name, short description and long description.
categoryType	string	A user can specify three types of search criteria: All, Any, or Exact Phrase. The value in this variable is used to store a user's search criteria. The value must be `ALL` for all search criteria, `ANY` for any search criteria or `EXACT` for exact phrase criteria.

Table 7. (continued)

currency	String	The value of this variable is used in searches on a currency.
currencyCaseSensitive	String	A user can choose case sensitive or case insensitive search. The value in this variable is used to identify if a search is case sensitive or not. The value must be either `yes` for a case sensitive search or `no` for a case insensitive search.
currencyOperator	string	A user can choose either `like` or `equal` as search operators. The value in this variable is used to store a user's choice. The value must be either `LIKE` for a like operator or `EQUAL` for equal operator.
filterTerm	string	The value in this variable is used to filter a search on a specified value.
filterTermCaseSensitive	string	A user can choose case sensitive or case insensitive search. The value in this variable is used to identify if a search is case sensitive or not. The value must be either `yes` for a case sensitive search or `no` fpr a case insensitive search.
filterTermOperator	string	A user can choose either `like` or `equal` as search operators. The value in this variable is used to store a user's choice. The value must be either `LIKE` for the like operator or `EQUAL` for the equal operator.
filterType	string	A user can specify three types of search criteria `All`, `Any`, or `Exact Phrase`. The value in this variable is used to store a user's search criteria. The value must be one of `ALL` for all search criteria, `ANY` for any search criteria, or `EXACT` for exact phrase criteria.

Table 7. (continued)

isListPriceOn	string	Catalog search bean user can choose to expose list price or standard price. To expose list price, set this variable to `yes`, and to expose standard price, set this variable to `no`. By default, the search bean exposes standard price.
manufacturer	string	The value of this variable is used in search on a manufacturer's name.
manufacturerCaseSensitive	string	A user can choose case sensitive or case insensitive search. The value in this variable is used to identify if a search is case sensitive or not. The value must be either `yes` for a case sensitive search or `no` for a case insensitive search.
manufacturerOperator	string	A user can choose either `like` or `equal` as search operators. The value in this variable is used to store a user's choice. The value must be either `LIKE` for the like operator or `EQUAL` for the equal operator.
manufacturerPartNum	string	The value of this variable is used in search on a manufacturer's part number.
manufacturerPartNumCaseSensitive	string	A user can choose case sensitive or case insensitive search. The value in this variable is used to identify if a search is case sensitive or not. The value must be either `yes` for a case sensitive search, or `no` for a case insensitive search.
manufacturerPartNumOperator	string	A user can choose either `like` or `equal` as search operators. The value in this variable is used to store a user's choice. The value must be either `LIKE` for a like operator or `EQUAL` for an equal operator.
maxPrice/minPrice	string	The values of these variables are used in search on price range.
pageSize	string	The value of this variable specifies number of search result rows to be displayed per page.

Table 7. (continued)

price	string	The value of this variable is used in search on price.
priceOperator	string	A user can choose one of the following operators as search operators: =, <, >, !=, <=, >=. The value in this variable is used to store a user's choice. The value must be any one of the following: `EQUAL`, `NOTEQUAL`, `GREATER`, `LESS`, `GREATER_EQUAL`, `LESS_EQUAL`.
qtyAvailable	string	The value of this variable is used in search on inventory of a product or item.
qtyAvailableOperator	string	A user can choose one of the following operators as search operators: =, <, >,!=, <=, >=. The value in this variable is used to store a user's choice. The value must be any one of the following:`EQUAL`, `NOTEQUAL`, `GREATER`, `LESS`, `GREATER_EQUAL`, `LESS_EQUAL`
qtyMeasure	string	The value of this variable is used in search on quantity measure.
qtyMeasureCaseSensitive	string	A user can choose case sensitive or case insensitive search. The value in this variable is used to identify if a search is case sensitive or not. The value must be either `yes` for a case sensitive search, or `no` for a case insensitive search.
qtyMeasureOperator	string	A user can choose either `like` or `equal` as search operators. The value in this variable is used to store a user's choice. The value must be either `LIKE` for a like operator or `EQUAL` for an equal operator.
resultCount	string	This variable will contain total number of results returned for a search.

Table 7. (continued)

resultType	string	Merchant can specify if they want to show Products or Items or both Products and Items in a search result. The value in this variable is used to store this value. The value must be one of `1` for products only, `2` for items only, or `3` both products and items.
searchTerm	string	The value of this variable is used in search on a word.
searchTermCaseSensitive	string	A user can choose case sensitive or case insensitive search. The value in this variable is used to identify if a search is case sensitive or not. The value must be either `yes` for a case sensitive search, or `no` for a case insensitive search.
searchTermOperator	string	A user can choose either `like` or `equal` as search operators. The value in this variable is used to store a user's choice. The value must be either `LIKE` for a like operator or `EQUAL` for an equal operator.
searchTermScope	Integer	A user can restrict the scope of a search to one of name; name and short description; name, short description, and long description; or keyword. The value in this variable is used to store a user's choice. The value must be one of `1` for name and short description, or `2` for name only, `3` for name, short description, and long description, or `4` for keyword.
searchType	string	A user can specify three types of search criteria `All`, `Any`, or `Exact Phrase`. The value in this variable is used to store a user's search criteria. The value must be `ALL` for all search criteria, `ANY` for any search criteria, or `EXACT` for exact phrase criteria.
sku	string	The value of this variable is used in searches on a SKU.

Table 7. (continued)

skuCaseSensitive	string	A user can choose case sensitive or case insensitive search. The value in this variable is used to identify if a search is case sensitive or not. The value must be either `yes` for a case sensitive search or `no` for a case insensitive search.
skuOperator	string	A user can choose either `like` or `equal` as search operators. The value in this variable is used to store a user's choice. The value must be either `LIKE` for a like operator or `EQUAL` for an equal operator.

Catalog search database columns

List of database columns that the Catalog search bean can search:

Table 8.

Searchable Attribute defined in bean interface	**Table**	**Columns**	**Supports Boolean expressions**
Category Group - All Name and Descriptions	CatGrpDesc	NAME LONGDESCRIPTION SHORTDESCRIPTION	YES
Category Group - Title Only	CatGrpDesc	NAME	YES
Category Group - Title and Description (suggested default)	CatGrpDesc	NAME SHORTDESCRIPTION	YES
Category Language ID	CatGrpDesc	LANGUAGE_ID	NO
Category Identifier	CatGpEnRel	CATGROUP_ID	No
CatEntry Description - All Name and Descriptions	CatEntDesc	NAME LONGDESCRIPTION SHORTDESCRIPTION	YES
CatEntry - Title Only	CatEntDesc	NAME	YES
CatEntry - Title and Description (suggested default)	CatEntDesc	NAME SHORTDESCRIPTION	YES
CatEntry Language ID	CatEntDesc	LANGUAGE_ID	NO
CatEntry Manufacturer Name	CatEntry	MFNAME	NO
CatEntry Manufacturer PartNumber	CatEntry	MFPARTNUMBER	NO
CatEntry SKU	CatEntDesc	PARTNUMBER	NO
Price	Listprice/ Offerprice	PRICE	NO
Price Range	Listprice/ Offerprice	PRICE	No

Table 8. (continued)

Searchable Attribute defined in bean interface	Table	Columns	Supports Boolean expressions
Currency	Listprice/ Offerprice	CURRENCY	NO
Quantity Available	InvStVw/ StoreInv	QTYAVAILABLE	NO
Quantity Measure	InvStVw/ StoreInv	QUANTITYMEASURE	NO

User of the bean can expose more columns like rich attributes by customizing the bean. For more information on customization, refer to the JavaDoc for the ExtendedCatEntrySearchListDataBean class in the online help.

Chapter 7. Coupons

The elements discussed in this chapter represent the user interface to the Coupons components in the WebSphere Commerce Accelerator. They facilitate the creation and maintenance of coupons, coupon offers, and other tasks that a Seller or Merchant performs on a daily basis. The coupons component includes the following elements:

- Coupons wizard

Customization examples

The following examples outline how to customize the coupons tools in the WebSphere Commerce Accelerator.

Scenario 1: Add new information when creating coupon discounts

Implementation overview

This scenario guides you through the steps necessary to customize the Coupon tools to record additional information in the coupon, such as who created the coupon discount.

Customization steps

1. Implement a new **CustomCreateCouponDiscountCmdImpl** which extends the task command implementation **CreateCouponDiscountCmdImpl** and implements **CreateCouponDiscountCmd**.
2. Use the `FIELD1` column of the `CALCODE` table to store the `userId` of the user who created this coupon discount.
3. Register the new task command in the `CMDREG` table. Assume that the customer store ID is 1, and then run the following SQL statement.

   ```
   insert into cmdreg( storeent_id,interfacename, classname)
     values(1,'com.ibm.commerce.tools.ecoupon.CreateCouponDiscountCmd',
     'com.ibm.commerce.tools.ecoupon.CustomCreateCouponDiscountCmdImpl')
   ```

4. Update the **CustomCreateCouponDiscountCmdImpl**. In the command, implement the `performExecute()` method:

   ```
   public void performExecute() throws ECSystemException, ECException {
      super.performExecute();
      /** Get the userId from the command context
      Store the userId to table calcode
      **/
      }
   ```

© Copyright IBM Corp. 2001

Chapter 8. Discounts

The elements discussed in this chapter represent the user interface to the Discounts components in the WebSphere Commerce Accelerator. They facilitate the creation and maintenance of discounts and other actions that a Seller or Merchant performs on a daily basis. The discounts component includes the following elements:

- Discount wizard
- Discount notebook

Customization examples

The following examples outline how to customize the discounts component of the WebSphere Commerce Accelerator.

Scenario 1: Add additional information when creating discounts

Implementation overview

This scenario guides you through the steps required if you want to add additional information when creating discounts. The example assumes that you want to create an audit trail of who created the discount. This scenario does not require that you submit new parameters.

Customization steps

1. **Implement a new CustomCreateDiscountCmdImpl**, which extend task command implementation **CreateDiscountCmdImpl** and implement **CreateDiscountCmd**.
2. **Use the field1 column of the CALCODE table to store the userId of the user who creates this discount.**
3. **Register the new task command in the CMDREG table.** Assume that the customer store ID is 1, and then run the following SQL statement:

   ```
   insert into cmdreg( storeent_id,interfacename, classname)
     values(1,'com.ibm.commerce.tools.promotions.CreateDiscountCmd',
     'com.ibm.commerce.tools.promotions.CustomCreateDiscountCmdImpl')
   ```

4. **In CustomCreateDiscountCmdImpl,** implement method performExecute():

   ```
   public void performExecute() throws ECSystemException, ECException {
     super.performExecute();
     /**  Get the userId from the command context
     Store the userId to table calcode
     **/
     }
   ```

Scenario 2: Change default behavior

Implementation overview

This scenario changes the default behavior of the discount wizard so that you can specify the sequence in which the discount is applied to an order. This scenario also requires that you pass new parameters through the URL. This requires adding a field to the first panel of the Discount wizard to obtain the extra data.

Customization steps

1. Update the JSP file responsible for the Discount wizard by doing the following:

© Copyright IBM Corp. 2001

a. Change to the following directory:

   AIX `/usr/WebSphere/CommerceServer/wcstool.war/tools/promotions/`

   400 `/QIBM/UserData/WebSphere/CommerceServer/wcstool.war/tools/promotions/`

   Linux `/opt/WebSphere/CommerceServer/wcstool.war/tools/promotions/`

   Solaris `/opt/WebSphere/CommerceServer/wcstool.war/tools/promotions/`

   Windows *drive*`:\WebSphere\CommerceServer\wcstool.war\tools\promotions\`

b. Copy `DiscountWizardWelcome.jsp` to `MyDiscountWizardWelcome.jsp`

c. Update `MyDiscountWizardWelcome.jsp`. In the form block, add the following code:

```
<input name="sequence" type="TEXT" size=3 MAXLENGTH=3>  Sequence
```

d. Save the entry to the framework. In the your JSP, there is a JavaScript function `savePanelData`, add the following line in the function:

```
parent.put("sequence", document.welcomeForm.sequence.value);
```

e. Register a new entry to the VIEWREG table. For example, if the customer store ID is 1, run the following SQL statement:

```
insert into viewreg (viewname, devicefmt_id, storeent_id, interfacename,
       classname, properties, https, internal)
  values('CusDiscountWizWelcomeView', -1, 0,
       'com.ibm.commerce.tools.command.ToolsForwardViewCommand',
       'com.ibm.commerce.tools.command.ToolsForwardViewCommandImpl',
       'docname=tools/promotions/extend/CusDiscountWizardWelcome.jsp', 0, 1)
```

2. Implement a new **DiscountSaveCmdCustomImpl**, which extend controller command implementation **DiscountSaveCmdImpl** and implements **DiscountSaveCmd**.

3. Register this new implementation in the `CMDREG` table. Assume that the customer store ID is 1, and then run the following SQL statement.

```
insert into cmdreg(storeent_id,interfacename,classname,target)
   values (1,'com.ibm.commerce.tools.promotions.DiscountSaveCmd',
   'com.ibm.commerce.tools.promotions.DiscountSaveCmdCustomImpl')
```

4. In the **DiscountSaveCmdCustomImpl** command, implement two methods by using the following sample code:

```
Public  void validateParameters() {
  super.validateParameters();
  /*
  Get your new parameters from requestProperty
  */
  }
Public void customMethod() {
  super.customMethod();
  /*
  Your action here.
  */
  }
```

Chapter 9. RFQ response

Customization examples

In this document, we provide three customization samples. They are:

- Copying a response
- Delete the Retracted state from the life cycle of an RFQ response
- Entering descriptive information for a response during creation

Scenario 1: Copying a response

This sample customization adds functionality to the RFQ response tools which enables a user to create a duplicate RFQ response based on the selected RFQ response. The customization includes adding the **Duplicate** button which, when clicked, prompts the user to enter a unique name for the duplicate response. When created, the new response is in the `draft` state.

Implementation overview

Retrieve all the detail information of this response, and create the new response by using the information retrieved. As the response name can not be the same in current schema, we need to provide a page to let users supply another name. This helps prepare all of the information in this page and passes it to the **RFQResponseCopyCmd** command, which raises a create response event. The UBF then calls the **RFQResponseCreateCmd** command to create a response.

Customization Steps

1. Add an interface **RFQResponseCopyCmd** and its implementation **RFQResponseCopyCmdImpl** by doing the following:
   a. Add the new interface **RFQResponseCopyCmd**. The code defining the interface follows:

```
public interface RFQResponseCopyCmd extends ControllerCommand{
   public final static String NAME =
"com.ibm.commerce.rfq.commands.RFQResponseCopyCmd";
   public final static String defaultCommandClassName =
      "com.ibm.commerce.rfq.commands.RFQResponseCopyCmdImpl";
}
```

   b. Add the command implementation **RFQResponseCopyCmdImpl** by extending `com.ibm.commerce.ubf.commands.ToolsBusinessFlowEventCmdImpl` and implementing **RFQResponseCopyCmd**. This command uses the data submitted through the dialog to trigger a create response event. The event causes UBF to call **RFQResponseCreateCmdImpl** to create a new response. The code defining the command follows:

```
public void performExecute() throws ECException
{
   TypedProperty parms = null;
   BusinessFlowEventData data = null;
   try {
    parms = getRequestProperties();
 parms.put(BusinessFlowConstants.EC_FLOWID,RFQConstants.EC_FLOW_RESPONSE_ID);
 parms.put(BusinessFlowConstants.EC_BUSINESS_FLOW_EVENT_IDENTIFIER,
          "createRFQResponse");

 data = new BusinessFlowEventData(getCommandContext(), parms);
 BusinessFlowEvent event = new BusinessFlowEvent(data,true);

 parms = data.getResponseProperties();
```

© Copyright IBM Corp. 2001

```
responseProperties = new TypedProperty();

for (Enumeration pns = parms.keys(); pns.hasMoreElements();) {
   String paramName = (String) pns.nextElement();
if (paramName.equals(ECConstants.EC_VIEWTASKNAME))
   responseProperties.put(ECConstants.EC_VIEWTASKNAME, "DialogNavigation");
else if (paramName.equals(UIProperties.SUBMIT_FINISH_MESSAGE))
   responseProperties.put(UIProperties.SUBMIT_FINISH_MESSAGE,
              "The response was copied successfully!");
else {
           Object newVal = parms.get(paramName);
           responseProperties.put(paramName, newVal);
}
}
} catch (ECException e) {
   parms = e.getErrorProperties();
responseProperties = new TypedProperty();
responseProperties.put(ECConstants.EC_VIEWTASKNAME, "DialogNavigation");
responseProperties.put(UIProperties.SUBMIT_ERROR_STATUS, "ERROR");
responseProperties.put(UIProperties.SUBMIT_ERROR_MESSAGE,
        "Copying response failed, please input another name.");
throw new ECApplicationException(new ECMessage(ECMessageSeverity.INFO,
        ECMessageType.USER, "_ERR_RESPONSE_COPY",
        "com.ibm.commerce.tools.rfq.properties.RFQMessages"),
        this.getClass().getName(), "",
        "DialogNavigation",responseProperties);
  }
}
```

c. Register **RFQResponseCopy** in the URLREG table Run the following SQL to register the command:

```
insert into urlreg values('RFQResponseCopy',0,'com.ibm.commerce.rfq.commands.RFQResponseCopyCmd',0,null,null,1);
```

2. Add a JSP to enable users to input a name for the response, and which retrieves all of the information associated with the response:

   a. This JSP file provides a text field for users to supply a new response name. The following is the source code for the text field:

```
<FORM name="responseCopyForm>
<table>
    <tr><td>
       <label><%= rfqNLS.get("name") %> <%= rfqNLS.get("required") %></label>
  </td></tr>
  <tr><td>
       <input type="Text" name="response_name" size="30" maxlength="200">
  </td></tr>
</table>
</FORM>
```

   When users click **OK**, the following code saves the response name in the input field:

```
function savePanelData() {
var form = document.responseCopyForm;
parent.put("<%= RFQConstants.EC_RFQ_RESPONSE_NAME %>",form.response_name.value);
return true;
}
```

   When initializing this JSP file, the following code retrieves all the information associated with the response and saves it. The information will be provided to **RFQResponseCopyCmdImpl**.

```
function initializeState(){
parent.setContentFrameLoaded(false);
parent.put("<%= RFQConstants.EC_RFQ_REQUEST_ID %>"  ,<%= RequestId %>);
parent.put("<%= RFQConstants.EC_RFQ_RESPONSE_REMARK%>",
      "<%= UIUtil.toJavaScript((String)RFQres.getRemarks())%>");

var rfqCommentsArrary = new Array() ;
<%
RFQResCommentsPair[] commentsPair =
       RFQResProdHelper.getRFQLevelCommentsPair(RequestId,ResponseId,null);
for (int index=0; commentsPair  != null && index <commentsPair.length; index++){
  %>

rfqCommentsArrary[<%=index%>] = new Object();
   rfqCommentsArrary[<%=index%>].<%=RFQConstants.EC_REQUEST_TC_ID%>="<%=
              commentsPair[index].getRFQ_TC_ID() %>";
```

```
            rfqCommentsArrary[<%=index%>].<%=RFQConstants.EC_TC_RFQ_LEVEL_COMMENTS%>="
                <%= UIUtil.toJavaScript((String)commentsPair[index].getRFQ_value())%>";
            rfqCommentsArrary[<%=index%>].<%=RFQConstants.EC_ATTR_MANDATORY%>= "
                <%= commentsPair[index].getMandatory() %>";
            rfqCommentsArrary[<%=index%>].<%=RFQConstants.EC_ATTR_CHANGEABLE%>= "
                <%= commentsPair[index].getChangeable()%>";
        rfqCommentsArrary[<%=index%>].<%=RFQConstants.EC_ATTR_RES_CMMENTS_VALUE%>="
                <%= UIUtil.toJavaScript((String)commentsPair[index].getRes_value())%>";
<%}%>
parent.put("<%= RFQConstants.EC_TC_RFQ_LEVEL_COMMENTS %>",rfqCommentsArrary);

var ProductsArray = new Array();
<%
RFQResNewProd[] ResPros = RFQResProdHelper.getResAllProds(RequestId,ResponseId,langId);
int i=0;
for(;ResPros != null && i < ResPros.length;i++){
%>
   ProductsArray[<%=i%>] = new Object();
    ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_CATENTRYID%>="
          <%=ResPros[i].getCatentry_id() %>";
      ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRICE%>= "
          <%=ResPros[i].getPrice() %>";
    ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_QUANTITY%>="
          <%=ResPros[i].getQuantity() %>";
 ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_CURRENCY%>= "
          <%=ResPros[i].getCurrency() %>";
 ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_UNIT%>="
          <%=ResPros[i].getUnit() %>";
 ProductsArray[<%=i%>].<%=RFQConstants.EC_ATTR_PRODUCT_COMMENTS%>=new Array();
    ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRODATTRLIST%>=new Array();
<%
RFQResProdAttributes[] resAttrs=RFQResProdHelper.getResAllAttributes(RequestId,
           ResponseId, ResPros[i].getCatentry_id(), langId,
           rfqNLS.get("valuedelim").toString());
int m=0,n=0;
for (int j=0;resAttrs != null && j<resAttrs.length>

ProductsArray[<%=i%>].<%=RFQConstants.EC_ATTR_PRODUCT_COMMENTS%>[<%=m++%>] = new Object;
ProductsArray[<%=i%>].<%=RFQConstants.EC_ATTR_PRODUCT_COMMENTS%>[<%=m-1%>].
              <%=RFQConstants.EC_ATTR_PATTRID%> = "<%= resAttrs[j].getPAttribute_id()%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_ATTR_PRODUCT_COMMENTS%>[<%=m-1%>].
              <%=RFQConstants.EC_ATTR_NAME%> = "<%= UIUtil.toJavaScript((String)resAttrs[j].getName())%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_ATTR_PRODUCT_COMMENTS%>[<%=m-1%>].
              <%=RFQConstants.EC_ATTR_VALUE%> = "<%= UIUtil.toJavaScript((String)resAttrs[j].getRes_value())%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_ATTR_PRODUCT_COMMENTS%>[<%=m-1%>].
              <%=RFQConstants.EC_ATTR_MANDATORY%> = "<%= resAttrs[j].getMandatory()%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_ATTR_PRODUCT_COMMENTS%>[<%=m-1%>].
              <%=RFQConstants.EC_ATTR_CHANGEABLE%> = "<%= resAttrs[j].getChangeable()%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_ATTR_PRODUCT_COMMENTS%>[<%=m-1%>].
              <%=RFQConstants.EC_REQUEST_TC_ID%> = "<%= resAttrs[j].getReq_tc_id()%>";
<%}else{%>
ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRODATTRLIST%>[<%=n++%>] = new Object;
ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRODATTRLIST%>[<%=n-1%>].
              <%=RFQConstants.EC_ATTR_PATTRID%> = "<%= resAttrs[j].getPAttribute_id()%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRODATTRLIST%>[<%=n-1%>].
              <%=RFQConstants.EC_ATTR_NAME%> = "<%= UIUtil.toJavaScript((String)resAttrs[j].getName())%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRODATTRLIST%>[<%=n-1%>].
              <%=RFQConstants.EC_ATTR_VALUE%> = "<%= UIUtil.toJavaScript((String)resAttrs[j].getRes_value())%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRODATTRLIST%>[<%=n-1%>].
              <%=RFQConstants.EC_ATTR_VALUEDELIM%> = "<%=rfqNLS.get("valuedelim")%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRODATTRLIST%>[<%=n-1%>].
              <%=RFQConstants.EC_ATTR_MANDATORY%> = "<%= resAttrs[j].getMandatory()%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRODATTRLIST%>[<%=n-1%>].
              <%=RFQConstants.EC_ATTR_CHANGEABLE%> = "<%= resAttrs[j].getChangeable()%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRODATTRLIST%>[<%=n-1%>].
              <%=RFQConstants.EC_REQUEST_TC_ID%> = "<%= resAttrs[j].getReq_tc_id()%>";
   ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRODATTRLIST%>[<%=n-1%>].
              <%=RFQConstants.EC_ATTR_OPERATOR%> = "<%= resAttrs[j].getOperator_id()%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRODATTRLIST%>[<%=n-1%>].
              <%=RFQConstants.EC_ATTR_UNIT%> = "<%= resAttrs[j].getUnit()%>";
ProductsArray[<%=i%>].<%=RFQConstants.EC_OFFERING_PRODATTRLIST%>[<%=n-1%>].
              <%=RFQConstants.EC_ATTR_TYPE%> = "<%= resAttrs[j].getType()%>";

<% }

}%>
<%
}
%>
parent.put("<%= RFQConstants.EC_OFFERING_PRODITEM %>",ProductsArray);

parent.setContentFrameLoaded(true);
}
.
.
.
<BODY class="content" onLoad="initializeState()">
```

b. Run the following SQL statement to register a view command into the VIEWREG table which maps the new JSP file.

```
insert into viewreg values('RFQRspDuplicateDialog', -1, 0,
   'com.ibm.commerce.tools.command.ToolsForwardViewCommand',
   'com.ibm.commerce.tools.command.ToolsForwardViewCommandImpl',
   'docname=tools/rfq/rfq_response_duplicate_dialog.jsp', null, 0, null, 0);
```

c. Add a new XML file called `rfq_response_duplicate_dialog.xml` into the following directory:

AIX /usr/WebSphere/CommerceServer/xml/tools/rfq

400 /QIBM/UserData/WebSphere/CommerceServer/xml/tools/rfq

Linux /opt/WebSphere/CommerceServer/xml/tools/rfq

Solaris /opt/WebSphere/CommerceServer/xml/tools/rfq

Windows *drive*:\WebSphere\CommerceServer\xml\tools\rfq

The finishURL is **RFQResponseCopy**, which maps the registered URL. The panel of this dialog is **RFQRspDuplicateDialog**, which maps the registered view command. The code of the XML file is as follows:

```
<?xml version="1.0"?>

<dialog resourceBundle="utf.utfNLS"
   windowTitle="dupliateDialog_Title"
   finishURL="RFQResponseCopy">

   <panel name="general"
       url="/webapp/wcs/tools/servlet/RFQRspDuplicateDialog"
       parameters="responseId"
       hasFinish="YES"/>
   <jsFile src="/wcs/javascript/tools/rfq/rfq_response_duplicate_dialog.js"/>
</dialog>
```

d. Register this XML in resource.xml.

   1) Backup the resource.xml in the following directory:

      AIX /usr/WebSphere/CommerceServer/xml/tools/rfq

      400 /QIBM/UserData/WebSphere/CommerceServer/xml/tools/rfq

      Linux /opt/WebSphere/CommerceServer/xml/tools/rfq

      Solaris /opt/WebSphere/CommerceServer/xml/tools/rfq

      Windows *drive*:\WebSphere\CommerceServer\xml\tools\rfq

   2) Add the following two lines to the resource.xml file:

```
<XML name="rfqRspDuplicateDialog"
 file="rfq/rfq_response_duplicate_dialog.xml"/>
```

e. Add a new JavaScript file called rfq_response_duplicate_dialog.js in the following directory:

   AIX /usr/WebSphere/CommerceServer/web/javascript/tools/rfq

   400 /QIBM/UserData/WebSphere/CommerceServer/web/javascript/tools/rfq

   Linux /opt/WebSphere/CommerceServer/web/javascript/tools/rfq

   Solaris /opt/WebSphere/CommerceServer/web/javascript/tools/rfq

   Windows *drive*:\WebSphere\CommerceServer\web\javascript\tools\rfq

   The JavaScript file contains some JavaScript functions used by the dialog. The following is the source code of the JavaScript file:

```
function submitErrorHandler (errMessage){
   self.CONTENTS.alertDialog(errMessage);
}

function submitFinishHandler (finishMessage){
   alertDialog(finishMessage);
   top.goBack();
}

function submitCancelHandler(){
   top.goBack();
}
```

3. Integrate this function into the RFQ response list page.

a. Add a **Duplicate** button to the response list page. That is, change the XML file as follows:

   Note: Backup the XML file `rfq_response_list.xml`, found in the following directory:

   AIX `/usr/WebSphere/CommerceServer/xml/tools/rfq`

   400 `/QIBM/UserData/WebSphere/CommerceServer/xml/tools/rfq`

   Linux `/opt/WebSphere/CommerceServer/xml/tools/rfq`

   Solaris `/opt/WebSphere/CommerceServer/xml/tools/rfq`

   Windows *drive*`:\WebSphere\CommerceServer\xml\tools\rfq`

   In the end of the `<button>... </button>` node in the `rfq_response_list.xml`, add the following line.

```
<menu name="duplicate"
   action="basefrm.duplicateRes()"
   selection="single">
</menu>
```

b. Add new JavaScript functions, `getDuplicateBCT()`, and `duplicateRes()` in `rfq_response_list.jsp`:

```
function getDuplicateBCT() {
   return "<%= rfqNLS.get("duplicate") %>";
}

function duplicateRes(){
   if(isButtonDisabled(parent.buttons.buttonForm.duplicateButton))
   return;
   var checkedEntries = parent.getChecked().toString();
   var parms = checkedEntries.split(';');
   var resId = parms[0];
   top.setContent(getDuplicateBCT(),'/webapp/wcs/tools/servlet/DialogView?
          XMLFile=rfq.rfqRspDuplicateDialog&responseId='+resId,true)
}
```

c. Add a JavaScript function to `rfq_response_list.jsp` and call it to disable the **Duplicate** button when the state of the corresponding RFQ is not ACTIVE. Add the following function into the `rfq_response_list.jsp` and call it after calling `DisableNewButton()`:

```
function DisableDuplicateButton()
{
   var reqState;
   var active=<%= com.ibm.commerce.utf.helper.UTFOtherConstants.EC_STATE_ACTIVE %>;
   reqState="<%=rfqState%>";
   if (reqState !=active){
      parent.hideButton('duplicate');
   }
}
```

4. Add a new message in the `RFQMessages_en_US.properties` file found in the following directory:

   AIX

   `/usr/WebSphere/CommerceServer/properties/com/ibm/commerce/tools/rfq/properties`

   400

   `/QIBM/UserData/WebSphere/CommerceServer/properties/com/ibm/commerce/tools/rfq/properties`

   Linux

   `/opt/WebSphere/CommerceServer/properties/com/ibm/commerce/tools/rfq/properties`

   Solaris

   `/opt/WebSphere/CommerceServer/properties/com/ibm/commerce/tools/rfq/properties`

   Windows

   drive`:\WebSphere\CommerceServer\properties\com\ibm\commerce\tools\rfq\`

properties.
Add the following line to the file. This message displays when copying fails:

```
_ERR_RESPONSE_COPY                  = Copying Response failed.
```

Scenario 2: Deleting the Retracted state from the life cycle of an RFQ response

Out of the box, when a user retracts a response, the response changes from the Active state to the Retracted state. If instead, the user wants the response to be set to the Draft state directly after retracting it, you must customize the function by deleting the Retracted state from the life cycle of a response.

Implementation overview

In this customization, we first need to delete the Retracted state from the response state machine. Once there is no longer a Retracted state, the Retracted state should also be removed from the view list of the response list page.

Customization steps

1. Change the UBFStatemachine.xml and UBFStatemachine_en_US.xml and reload them. Backup UBFStatemachine.xml and UBFStatemachine_en_US.xml in the following directory:

   AIX

   /usr/WebSphere/CommerceServer/xmlloadutility/businessfollows/xml

   400

   /QIBM/UserData/WebSphere/CommerceServer/xmlloadutility/businessfollows/xml

   Linux

   /opt/WebSphere/CommerceServer/xmlloadutility/businessfollows/xml

   Solaris

   /opt/WebSphere/CommerceServer/xmlloadutility/businessfollows/xml

   Windows

   drive:\WebSphere\CommerceServer\xmlloadutility\businessfollows\xml

   Make the following changes to those two files:

   - Change the target state of the retractRFQResponse transition from RETRACTED to DRAFT.
   - Delete the information about the RETRACTED state and all transitions from this state

   The code sample below illustrates the changes required to the UBFStatemachine.xml file:

   Before changes

```
<Flow identifier="RFQ response process totally" priority="2">
   <State identifier="ACTIVE">
      <Transition eventidentifier="retractRFQResponse" approval="0" priority="2">
      <Action
          interface="com.ibm.commerce.rfq.commands.RFQResponseChangeStateAdvCmd">
      <AccessControlGuard actiongroup="RFQResponseManage">
      <TargetState identifier="RETRACTED"/>
      </Transition>
   </State>
   <State identifier="RETRACTED">
      <Transition eventidentifier="changeRFQResponseToDraft" approval="0" priority="1">
      <Action
          interface="com.ibm.commerce.rfq.commands.RFQResponseChangeStateAdvCmd">
      <AccessControlGuard actiongroup="RFQResponseManage">
      <TargetState identifier="DRAFT"/>
      </Transition>
      <Transition eventidentifier="cancelRFQResponse" approval="0" priority="2">
      <Action
          interface="com.ibm.commerce.rfq.commands.RFQResponseChangeStateAdvCmd">
      <AccessControlGuard actiongroup="RFQResponseManage">
      <TargetState identifier="CANCELLED"/>
      </Transition>
      <Transition eventidentifier="closeRFQRequest" approval="0" priority="3">
```

```
      <Action
          interface="com.ibm.commerce.rfq.commands.RFQResponseChangeStateBaseCmd">
      <AccessControlGuard actiongroup="RFQManage">
      <TargetState identifier="CANCELLED"/>
      </Transition>
      <Transition eventidentifier="cancelRFQRequest" approval="0" priority="4">
      <Action
          interface="com.ibm.commerce.rfq.commands.RFQResponseChangeStateBaseCmd">
      <AccessControlGuard actiongroup="RFQManage"/>
      <TargetState identifier="CANCELLED"/>
      </Transition>
   </State>
</Flow>
```

After changes

```
<Flow identifier="RFQ response process totally" priority="2">
   <State identifier="ACTIVE">
      <Transition eventidentifier="retractRFQResponse" approval="0" priority="2">
      <Action
          interface="com.ibm.commerce.rfq.commands.RFQResponseChangeStateAdvCmd">
      <AccessControlGuard actiongroup="RFQResponseManage"/>
      <TargetState identifier="DRAFT"/>
      </Transition>
   </State>
</Flow>
```

The changes to `UBFStatemachine.xml` are shown as follows:

Before changes

```
<State identifier="ACTIVE">
   <TransitionDesc eventidentifier="closeRFQRequest"
      transitiondescription=
      "Change the state of response to In-evaluation when closing request"
      eventdescription="closeRFQRequest">
   <TargetState identifier="IN-EVALUATION"/>
   </TransitionDesc>
   <TransitionDesc eventidentifier="retractRFQResponse"
      transitiondescription="Retract the response"
      eventdescription="retractRFQResponse">
   <TargetState identifier="RETRACTED"/>
   </TransitionDesc>
   <TransitionDesc eventidentifier="cancelRFQRequest"
      transitiondescription="Cancel the response when cancelling the rfq"
      eventdescription="cancelRFQRequest">
   <TargetState identifier="CANCELLED"/>
   </TransitionDesc>
</State>
<State identifier="RETRACTED">
   <TransitionDesc eventidentifier="changeRFQResponseToDraft"
      transitiondescription="Change the retracted response to draft"
      eventdescription="changeRFQResponseToDraft">
   <TargetState identifier="DRAFT"/>
   </TransitionDesc>
   <TransitionDesc eventidentifier="cancelRFQResponse"
      transitiondescription="Cancel the response"
      eventdescription="cancelRFQResponse">
   <TargetState identifier="CANCELLED"/>
   </TransitionDesc>
   <TransitionDesc eventidentifier="closeRFQRequest"
      transitiondescription="Cancel the response when closing the rfq"
      eventdescription="closeRFQRequest">
   <TargetState identifier="CANCELLED"/>
   </TransitionDesc>
   <TransitionDesc eventidentifier="cancelRFQRequest"
      transitiondescription="Cancel the response when cancelling the rfq"
      eventdescription="cancelRFQRequest">
   <TargetState identifier="CANCELLED"/>
   </TransitionDesc>
</State>
```

After changes

```
<State identifier="ACTIVE">
   <TransitionDesc eventidentifier="closeRFQRequest"
      transitiondescription=
      "Change the state of response to In-evaluation when closing request"
      eventdescription="closeRFQRequest">
   <TargetState identifier="IN-EVALUATION"/>
   </TransitionDesc>
   <TransitionDesc eventidentifier="retractRFQResponse"
      transitiondescription="Retract the response"
      eventdescription="retractRFQResponse">
   <TargetState identifier="DRAFT"/>
   </TransitionDesc>
   <TransitionDesc eventidentifier="cancelRFQRequest"
      transitiondescription="Cancel the response when cancelling the rfq"
      eventdescription="cancelRFQRequest">
   <TargetState identifier="CANCELLED"/>
   </TransitionDesc>
</State>
```

2. Since there is no longer a `Retracted` state in the life cycle of a response, delete the following lines in `rfq_response_list.xml` and

rfq_response_state_list.xml. The states here include draft, cancelled, pendingapproval, rejected, active, inevaluation, win, lost, wincomplete, and lostcomplete.

Note: The line splits here for display purposes.

```
<view name="resretracted"
   action="top.setContent(basefrm.getPageTitle(),
   '/webapp/wcs/tools/servlet/NewDynamicListView?
   ActionXMLFile=rfq.rfqresponseretractedlist&
   cmd=RFQResponseList&rfqId='+basefrm.getRfqId(),false)"/>
```

Notes:

1. This sample illustrates how to delete a state. If you want to add a state instead, perform similar steps, except in reverse:
   a. Add the new state transitions in the state machine XML file and reload it.
   b. Add the new view into the view list of the response list page

   For information about how to reload the state machine, refer to the UBF online help.

Scenario 3: Entering descriptive information for a response during creation

If the user wants to enter some descriptive text when creating a response to an RFQ, you must add a text field in the first panel of the RFQ creation wizard.

Implementation overview

Add a new text field in the first panel of RFQ response creation wizard. Since the **RFQResponseCreateCmdImpl** doesn't handle the extra input, a new interface **MyRFQResponseCreateCmd** and the implementation of it, command **MyRFQResponseCmdImpl** will be create by extending **RFQResponseCreateCmd** and **RFQResponseCreateCmdImpl**.

Customization steps

1. Add an interface **MyRFQResponseCreateCmd** and its implementation **MyRFQResponseCreateCmdImpl**.
   a. Add a new interface **MyRFQResponseCreateCmd**. The code of the interface is as following:

```
public interface MyRFQResponseCreateCmd extends RFQResponseCreateCmd {
   public final static String NAME =
       "com.ibm.commerce.rfq.commands.MyRFQResponseCreateCmd";
   public static final String COPYRIGHT =
       com.ibm.commerce.copyright.IBMCopyright.SHORT_COPYRIGHT;
   public static String defaultCommandClassName =
       "com.ibm.commerce.rfq.commands.MyRFQResponseCreateCmdImpl";
   }
```

   b. Add the command implementation **MyRFQResponseCreateCmdImpl** by extending **RFQResponseCreateCmdImpl** and implementing **MyFQResponseCreateCmd**. Add a new field responseDescription to store the response description transferred from the user interface. Also, add a getter method and a setter method for this field:

```
protected String responseDescription ="";
public java.lang.String getResponseDescription() {
   return responseDescription;
}
public void setResponseDescription(String name, boolean isReq) throws ECApplicationException {
   try {
      responseDescription = (String)getToolXMLObject().get(name);
   } catch (Exception e) { }

   if(isReq) {
      if (getResponseDescription()==null || getResponseDescription().length()==0) {
         setErrorFlag(true);
```

```
      getErrorContent().put(ECRFQMessageKey._ERR_RFQ_MISSING_RESPONSEREMARKS,"");
      }
   }
}
```

Override the `initParameters()` method. This method will be invoked by `checkParameters()` in **RFQResponseCreateCmdImp** to initialize parameters. The difference between this method and the one in the corresponding super class is that the invocation of **setResponseDescription (MyRFQConstants.EC_RFQ_RESPONSE_DESCRIPTION, false)** gets the response description transferred from the user interface. The source code of the method is as follows:

```
protected void initParameters()
throws com.ibm.commerce.exception.ECApplicationException
{
   setRequestId(RFQConstants.EC_RFQ_REQUEST_ID, true);
   setResponseName(RFQConstants.EC_RFQ_RESPONSE_NAME, true);
   setResponseRemarks(RFQConstants.EC_RFQ_RESPONSE_REMARK, false);
   setCommentsRFQLevelList(RFQConstants.EC_TC_RFQ_LEVEL_COMMENTS,false);
   setResProductsList(RFQConstants.EC_OFFERING_PRODITEM, false);

   //get response description from the user interface
   setResponseDescription(MyRFQConstants.EC_RFQ_RESPONSE_DESCRIPTION, false);
}
```

Override the `createResponse()` method. This method is identical, except that you must set the description field in the **RFQResponseAccessBean** and **TradingDescriptionAccessBean**. This source code of the method follows:

```
protected RFQResponseAccessBean createResponse ()
throws ECApplicationException,ECException {
   RFQResponseDataBean responseDB = null;
   try{
      CreateResponseBasicInfoCmd createResCmd = null;
      createResCmd = (CreateResponseBasicInfoCmd)
        CommandFactory.createCommand(CreateResponseBasicInfoCmd.NAME, getStoreId());
      createResCmd.setRequestId(getRequestId());
      createResCmd.setOwnerId(getOwnerId());
      createResCmd.setResponseName(getResponseName());
      createResCmd.setResponseRemarks(getResponseRemarks());
      createResCmd.setStateIdentifier(getStateIdentifier());
      createResCmd.setCommandContext(getCommandContext());

      createResCmd.execute();

      responseDB=createResCmd.getResponseDataBean();
      //This id will be picked by BusinessFlowEventListener
      //to add a record in FLINSTANCE table

      this.setEntityObject(responseDB);

      //set this response reference number
      setResponseId(responseDB.getRfqResponseIdInEJBType());

      responseDB.setDescription(responseDescription);
      responseDB.commitCopyHelper();
      TradingDescriptionAccessBean trdDesc = new TradingDescriptionAccessBean();
      trdDesc.setInitKey_languageId(getCommandContext().getLanguageId().toString());
      trdDesc.setInitKey_tradingId(responseDB.getRfqResponseId());
      trdDesc.refreshCopyHelper();
      if (getResponseDescription()==null || getResponseDescription().length()==0) {
         trdDesc.setShortDescription(responseDB.getName());
      }
      trdDesc.setShortDescription(getResponseDescription());
      trdDesc.setTimeCreated(TimestampHelper.systemCurrentTimestamp());
      trdDesc.commitCopyHelper();
   }catch (javax.ejb.CreateException e) {
      throw new ECApplicationException(ECRFQMessage._ERR_RESPONSE_BASICINFO_SAVE,
          this.getClass().getName(), "performExecute");
   }catch (javax.naming.NamingException e) {
      throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
          this.getClass().getName(), "createResponse");
   }catch (java.rmi.RemoteException e) {
      throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
          this.getClass().getName(), "createResponse");
   }catch (javax.ejb.FinderException e) {
      throw new ECSystemException(ECMessage._ERR_FINDER_EXCEPTION,
          this.getClass().getName(), "createResponse");
   }
   return responseDB;
}
```

c. Run the following SQL statement to register the **MyRFQReponseCreate** command into URLREG table:

```
insert into urlreg values('MyRFQResponseCreate', 0,
   'com.ibm.commerce.ubf.commands.ToolsBusinessFlowEventCmd', 0,
   null, null, 1);
```

2. Create a new constants class `MyRFQConstants` by extending `RFQConstants`. There is only one new constant definition:

```
public final static String EC_RFQ_RESPONSE_DESCRIPTION = "response_description";
```

3. Change the `rfq_w_response_general.jsp` file to add a new text field and the associated processing. Locate th file in the following directory:

   AIX `/usr/WebSphere/CommerceServer/web/tools/rfq`

   400 `/QIBM/UserData/WebSphere/CommerceServer/web/tools/rfq`

   Linux `/opt/WebSphere/CommerceServer/web/tools/rfq`

   Solaris `/opt/WebSphere/CommerceServer/web/tools/rfq`

   Windows *drive*`:\WebSphere\CommerceServer/web/tools/rfq`

   Change the `Form` section as follows:

```
<FORM name="rfqCreateForm">
<table COLS=3 WIDTH="60%">
<tr>
<td><%= rfqNLS.get("name") %></td>
</tr>
<tr>
<td><INPUT name="response_name" maxlength=100></td>
</tr>
<tr>
<td><%= rfqNLS.get("remark") %></td>
</tr>
<tr>
<td><TEXTAREA rows="4" cols="40" name="response_remark"></TEXTAREA></TD>
</tr>
<tr>
<td><%= rfqNLS.get("desc") %></td>
</tr>
<tr>
<td><TEXTAREA rows="4" cols="40"  name="response_description"></TEXTAREA></TD>
</TR>
</table>
</FORM>
```

   In the `savePanelData()` function, add code to save the response description. The changed function is as follows:

```
function savePanelData(){
   VPDResult = validatePanelData0();
   if(!VPDResult)
      return;
   if (isFirstTimeLogonWizard== "1")
   parent.put("<%=RFQConstants.EC_RFQ_REQUEST_ID%>", getRequestId());
   parent.put("<%=RFQConstants.EC_RFQ_RESPONSE_NAME%>",
       document.rfqCreateForm.response_name.value);
   parent.put("<%=RFQConstants.EC_RFQ_RESPONSE_REMARK%>",
       document.rfqCreateForm.response_remark.value);
   parent.put("<%=BusinessFlowConstants.EC_FLOWID%>",
       "<%=RFQConstants.EC_FLOW_RESPONSE_ID%>");
   parent.put("<%=BusinessFlowConstants.EC_BUSINESS_FLOW_EVENT_IDENTIFIER%>",
       "createRFQResponse");
   parent.put("<%=MyRFQConstants.EC_RFQ_RESPONSE_DESCRIPTION%>",
       document.rfqCreateForm.response_description.value);
}
```

   In the `retrievePanelData()` function, add code to retrieve response description. The changed code is as follows:

```
function retrievePanelData(){
   var form = document.rfqCreateForm;
   form.response_name.value = parent.get("<%=RFQConstants.EC_RFQ_RESPONSE_NAME%>","");
   form.response_remark.value = parent.get("<%=RFQConstants.EC_RFQ_RESPONSE_REMARK%>","");
   form.response_description.value = parent.get("<%=MyRFQConstants.EC_RFQ_RESPONSE_DESCRIPTION%>","");\
}
```

4. Change the target `finishURL` in the `rfq_response_wizard.xml` file from `RFQResponseCreate` to `MyRFQResponseCreate`. Locate the file in the following directory:

   AIX `/usr/WebSphere/CommerceServer/xml/tools/rfq`

   400 `/QIBM/UserData/WebSphere/CommerceServer/xml/tools/rfq`

Linux /opt/WebSphere/CommerceServer/xml/tools/rfq

Solaris /opt/WebSphere/CommerceServer/xml/tools/rfq

Windows *drive*:\WebSphere\CommerceServer/xml/tools/rfq

Change the according to the following code segment.

```
<wizard resourceBundle="utf.utfNLS"
   windowTitle="matchlisttitle"
   finishConfirmation="ex_finish"
   cancelConfirmation="ex_cancel"
   finishURL="MyRFQResponseCreate"
   tocBackgroundImage="/wcs/images/tools/toc/W_merchand.jpg">
```

5. Change the UBFStatemachine.xml file and reload it. Locate the file in the following directory:

AIX

/usr/WebSphere/CommerceServer/xmlloadutility/businessfollows/xml

400 /QIBM/UserData/WebSphere/CommerceServer/xmlloadutility/businessfollows/xml

Linux

/opt/WebSphere/CommerceServer/xmlloadutility/businessfollows/xml

Solaris

/opt/WebSphere/CommerceServer/xmlloadutility/businessfollows/xml

Windows

drive:\WebSphere\CommerceServer\xmlloadutility\businessfollows\xml

Change the action interface from com.ibm.commerce.rfq.commands.RFQResponseCreateCmd to com.ibm.commerce.rfq.commands.RFQResponseCreateCmd in both files. The following sample code illustrates the required changes:

Before changes

```
<Flow identifier="RFQ response process totally" priority="2">
   <StartState identifier="START">
      <Transition eventidentifier="createRFQResponse" approval="0" priority="1">
         <Action interface="com.ibm.commerce.rfq.commands.RFQResponseCreateCmd"/>
         <AccessControlGuard actiongroup="RFQResponseCreate"/>
         <TargetState identifier="DRAFT"/>
      </Transition>
   </StartState>
```

After changes

```
<Flow identifier="RFQ response process totally" priority="2">
   <StartState identifier="START">
      <Transition eventidentifier="createRFQResponse" approval="0" priority="1">
      <Action interface="com.ibm.commerce.rfq.commands.RFQResponseCreateCmd"/>
      <AccessControlGuard actiongroup="RFQResponseCreate"/>
      <TargetState identifier="DRAFT">
      </Transition>
   </StartState>
```

Note: For information about how to reload the state machine, refer to the UBF online help.

Chapter 10. Expected Inventory

Customization examples

The following examples outline how to customize this part of the WebSphere Commerce Accelerator.

Scenario 1: Add new information when creating Expected Inventory Records

This scenario guides you through the steps necessary to customize the Expected Inventory tools to record additional information in the Expected Inventory, such as who created the Expected Inventory Record.

1. Add a column to the `RA` table for the appropriate `userID`. Create a new `ExpectedInventoryRecords` enterprise bean and access bean.
2. Implement a new **CustomExpectedInventoryRecordCreateCmdImpl** which extends the controller command implementation **ExpectedInventoryRecordCreateCmdImp** and implements **ExpectedInventoryRecordCreateCmd**.
3. Register the new controller command in the `CMDREG`. Assume that the customer store ID is 1, and then run the following SQL statement:

```
insert into cmdreg(storeent_id,interfacename,classname)
   values(1,'com.ibm.commerce.tools.inventory.ExpectedInventoryRecordCreateCmd ',
   'com.ibm.commerce.tools.inventory.CustomExpectedInventoryRecordCreateCmdImpl ')
```

4. Update the **CustomExpectedInventoryRecordCreateCmdImpl** In the command, implement the `performExecute()` method:

```
public void performExecute()throws ECSystemException,ECException {
   super.performExecute();
   /**Get the userID from the command context
   Store the userID to the RA table
   **/
}
```

© Copyright IBM Corp. 2001

Chapter 11. Business intelligence

Dynamic context

Dynamic context is a list of grouped actions that users can choose to perform. The list contains the names and brief descriptions of the actions. The list changes dynamically based the roles of the user and components that are enabled.

Customization examples

Scenario: Add a new action to an existing context

To add a new action to an existing context, do the following:

1. Add an action to the context definition XML file. For example, the `biContext.xml` file in the `xml/tools/bi` directory. You need to find the context to which you want to add, and then add an entry to it. The following is the Campaign action of the Campaign context:

```
<entry nameKey="campaign" descriptionKey="campaignDescription"
      breadCrumbTrailTextKey="Report" toolsComponent="CommerceAnalyzer">
   <roles>
      <role>siteOwner</role>
      <role>siteAdmin</role>
      <role>seller</role>
      <role>merchant</role>
      <role>makMgr</role>
      <role>podMgr</role>
   </roles>
   <command name = "BIShowReport">
      <parameter name="reportId"  value="BICampaignsIndex.html" />
   </command>
</entry>
```

   To view the Campaign context, you select **Campaigns** from the **Marketing** menu of WebSphere Commerce Acceleratorr, then click **Reports**.

   In the above code fragment, the `toolsComponent` is an optional attribute. If specified, the action will be listed only when the specified component is enabled from the Configuration Manager.

   If the user is one of the roles that defined in the roles element, then the action will be listed for the user.

   The name attribute of the command element is the name of the command performed when the action is selected from the list. The parameter and the value attributes are the command parameters and the values of the parameters.

   The `appendQueryString` attribute of the entry element is not shown in the above code fragment. If the attribute presents and is set to true, the requested properties of the current request will be appended to the action command in the form of name value pairs. For example, the code sample below illustrates the Orders by Account Report action in the account report context:

```
entry nameKey="ordersByAccount" descriptionKey="ordersByAccountDescription"
     breadCrumbTrailTextKey="reportCriteria" appendQueryString="true">
   <roles>
      <role>siteOwner</role>
```

© Copyright IBM Corp. 2001

```
      <role>siteAdmin</role>
      <role>seller</role>
      <role>actRep</role>
      <role>salesMgr</role>
   </roles>
   <command name = "OrdersByAccountDialogView">
      <parameter name="XMLFile" value="reporting.OrdersByAccountReportDialog"/>
   </command>
</entry>
```

The `ContextEntry` class generates the following command based on the above definition (the command is one line, split here for presentation purposes):

```
/webapp/wcs/tools/servlet/OrdersByAccountDialogView
   ?contextConfigXML=contract.brmReportContext
   &startindex=0&ActionXMLFile=contract.rptAccountContextList
   &accountId=10001&docname=tools/bi/ContextList.jsp&resultsize=0
   &storeId=135&context=account&langId=-1
   &XMLFile=reporting.OrdersByAccountReportDialog&listsize=15
```

In the above command, the following name value pairs are extracted from the request properties and appended to the command: contextConfigXML=contract.brmReportContext &startindex=0 &ActionXMLFile=contract.rptAccountContextList &accountId=10001&docname=tools/bi/ContextList.jsp &resultsize=0&storeId=135 &context=account&langId=-1&listsize=15.

2. Add the property keys to the properties file.

   For national language enablement, the `nameKey`, `descriptionKey`, and `breadCrumbTrailTextKey` are the keys in the properties file. The property file of the above example is `properties/com/ibm/commerce/tools/bi/properties/BINLS_en_US.properties`. The `nameKey` maps to the name of the action. The `descriptionKey` maps to the description of the actions. The `breadCrumbTrailTextKey` maps to the text appended to the breadcrumb trail when the action is selected.

Chapter 12. Overview of the Reporting framework

The reporting framework provides a generic, customizable reporting functionality for almost any aspect of your site. The reports are accessible by any of the roles that use the Commerce Accelerator. Access for a particular report can be defined and limited within the report. WebSphere Commerce Accelerator users can request the reports at any time, and the framework generates the report using data contained in the production database, and displays the report in real time.

The framework consists of a generic controller command, a data bean, and a generic view which displays the result. You can customize the framework by adding valid SQL queries, and defining JSP files used to request and display the generated reports.

Customizing the reporting framework

Reports are accessible from the Commerce Accelerator. Consequently, each report requires a number of associated assets. While the report itself consists of data that are represented in a tabular format, the underlying assets consist of the report identifier, an SQL query, access control elements, and so on. The report request launches a controller command on the server. The controller command calls tasks to set the generic view, unless the report specifies a particular view. The command also sets a number of required variables, and returns this data to populate the ReportDataBean in the target JSP file. Access control for the reports is set on the views which request (input) and display the report. The results returned from the database are stored in the data bean as a vector of hashtable. Finally, the JSP file displays the report. If the report is empty, the JSP file displays a generic text string instead.

Access control for the reports is set on the views to request (input) and output the report.

Adding a new report requires the following steps:

1. Define the report in an XML file.
2. Create the JSP file from which the report is requested, if necessary.
3. Create the JSP file to display the report, unless the generic JSP is used.

Customization examples

Defining the report in an XML file

Individual reports are defined using XML files. Each report has a corresponding *reportName*.xml file. This file contains all of the information necessary to generate a report. The *reportname*.xml file looks similar to the following example for the MyStoreOverviewReport:

```
<?xml version="1.0" standalone="yes" ?>
<Reporting>
<!-- owner="ownerName" location="path_to_this_XML_file " -->
<!-- A Collection consists of SQLs for WCS reporting  -->

<Report reportName="MyStoreOverviewReport" online="true">
<comment>store_overview, yesterday, all measurements</comment>
<SQLvalue>
</SQLvalue>
<mergeOperation>1000000,1000001</mergeOperation>
<display>
<standardInfo>
```

© Copyright IBM Corp. 2001

```
<resource>reporting.ReportingString</resource>
<title></title>
<message>messageMyReport</message>
<columnTitles>CRITERIA,KEY,VALUE,CURRENCY,DATESTMP</columnTitles>
</standardInfo>
<userDefinedParameters>
</userDefinedParameters>
</display>
</Report>
<Report reportName="1000000" online="true">
<comment>store_overview, yesterday, revenue</comment>
<SQLvalue>
      select {revenue} as criteria, storeent_id as key,
             sum(totalproduct+totalshipping+totaltax+totaltaxshipping) as value,
             currency as currency, 0 as datestmp
       from orders
       where $DB_DATE_GREATER_EQUAL_FUNC(lastupdate,{beginDate})$  and
             $DB_DATE_LESS_EQUAL_FUNC(lastupdate,{endDate})$
             and status in ('C','M','S') and storeent_id={storeent_id}
       group by storeent_id, currency
</SQLvalue>
<display>
<standardInfo>
<resource>reporting.ReportingString</resource>
<title></title>
<message>message1000000</message>
<columnTitles>CRITERIA,KEY,VALUE,CURRENCY,DATESTMP</columnTitles>
</standardInfo>
<userDefinedParameters>
</userDefinedParameters>
</display>
</Report>
<Report reportName="1000001" online="true">
   <comment>store_overview, yesterday, number of orders</comment>
   <SQLvalue>
       select {orders} as criteria, storeent_id as key, count(*) as value,
             '-' as currency,
              0 as datestmp
       from orders
       where $DB_DATE_GREATER_EQUAL_FUNC(lastupdate,{beginDate})$  and
             $DB_DATE_LESS_EQUAL_FUNC(lastupdate,{endDate})$
             and status in ('C','M','S') and storeent_id={storeent_id}
       group by storeent_id
</SQLvalue>
<display>
      <standardInfo>
<resource>reporting.ReportingString</resource>
<title></title>
<message>message1000000</message>
<columnTitles>CRITERIA,KEY,VALUE,CURRENCY,DATESTMP</columnTitles>
</standardInfo>
<userDefinedParameters>
</userDefinedParameters>
</display>
</Report>
</Reporting>
```

To create a new report, you must create an XML file similar to the example above. For a detailed explanation of each XML element, refer to the section below entitled "Valid XML elements".

Valid XML elements: Define reports by using the following XML elements:

<Reporting></Reporting>
This is the root element.

<Report></Report>
This required element defines a particular report. You can define multiple reports in a single XML file by including more than one `<Report>` element. This element has two required attributes:

ReportName
A string defining the unique name for the report.

online A boolean value which specifies whether the report is available in real time. Currently, `true` is the only supported value.

<comment></comment>
An optional element in which you can describe the report. This string does not require translation. This element can only be defined within an existing <Report> element.

<SQLvalue></SQLvalue>
A required element which defines the SQL query used to generate the report. Although it is required, this element can be empty. This element can only be defined within an existing <Report> element.

<mergeOperation></mergeOperation>
An optional element which allows you to combine multiple SQL queries into one report. It contains a list of comma-delineated reportNames pointing to the reportName attributes of other <Report> elements. All of the SQL queries in the referenced reports must return the same number of columns, and use the same column names which identify the keys of the hashtable. Each SQL query is independent of the other queries. Each SQL query produces its own vector of hashtable, and before presentation, these vector are appended to each other to form a single report.

The Store Overview report example in the above section shows the use of the <mergeOperation> element. The first row of the final report comes from one SQL query, and the second row comes from a subsequent query. This element can only be defined within an existing <Report> element.

<extended_object_class></extended_object_class>
An optional element which contains a Java class name used to create an SQL statement by using an extended Java class. The class generates a report, and then sends the data back to the reporting control center. Use this element to create a recursive report. This element can only be defined within an existing <Report> element.

<display></display>
An optional element used to define the parameters used in the display of the result. This element can only be defined within an existing <Report> element.

<standardInfo></standardInfo>
A required element used to group elements together which are accessible through getters in the ReportDataBean. These elements are basic elements to most reports. This element can only be defined within an existing <display> element.

<resourceBundle></resourceBundle>
A required element used to specify the properties file to be used in for the report. The value must also be referenced in the reports/resources.xml file. This element can only be defined within an existing <standardInfo> element.

<title></title>
A required element used to specify the title of the report. The value must be a key in the properties file. This element can only be defined within an existing <standardInfo> element.

<message></message>
A required element used to display a message related to the report. For instance, this can be used to provide a description for the report. The value must be a key in the properties file. This element can only be defined within an existing <standardInfo> element.

<columnTitles></columnTitles>

A required element used to define the column titles. It contains a list of comma-delineated names. The names must be keys defined in the properties file. If the key cannot be found in the properties file then the key provided in the element is used as the column title. This element can only be defined within an existing `<standardInfo>` element.

<userDefinedParameters></userDefinedParameters>

An optional element used to define custom elements in the reporting framework. The reporting framework expects to see elements of the form:

```
<element1>value1</element1>
<element2>value2</element2>
```

The ReportDataBean provides a getter method which returns a hashtable of the above elements to be used in the customized display JSP. This element can only be defined within an existing `<display>` element.

Note: While elements contained within the `<display>` element are listed as required, this is only true if the optional display element is defined.

Variables in SQL queries: When defining variables in the *reportName*`.xml` file, the variable must be contained within curly braces ({*variableName*}). This indicates to the reporting framework that the value is a client variable, and must be obtained from the client hashtable. In the sample XML file presented above, {`revenue`}, {`beginDate`}, {`endDate`}, {`storeent_id`}, and {`orders`} are all client variables.

Create the JSP file from which the report is requested

Depending on the amount of information required to generate the report, you must decide whether to use either a dialog or a Wizard to gather the required data. Whichever element is appropriate, the JSP must include the `savePanelData` JavaScript function:

```
function savePanelData()
{

var reportInputData = new Object();

   reportInputData.SQLid = "the requested report  name" ;
reportInputData.reportXML = "some file";
   reportInputData.variable1 = "some value 1";
   reportInputData.variable2 = "some value 2" ;
.
   .
   .
reportInputData.variableN = "some value N";
   reportInputData.varProperties = "a list of variable separated by a comma";
   parent.put("reportInputData", reportInputData);

// The section below can be used to indicate a different View to be used
// var reportResultPage = new Object();
// reportResultPage.cmd = "ASpecificDisplayReportView";
// parent.put("reportResultPage",reportResultPage);

   return true;
}
```

The references to `reportInputData` and `reportResultPage` are required to pass the parameters to the controller command. The `SQLid` and `reportXML` variables are also required. The `variable1` through `variableN`, and `varProperties` are optional. In the example `variable1` through `variableN` represent the variables used in the SQL query. For example, `variable1` and `variable2` could be replaced by `beginDate` and `endDate`. Thus the following code would be present inside the `savePanelData()` function:

```
reportInputData.begindate = " some value";
reportInputData.enddate   = " some value";
```

The varProperties variable lists variables which obtain their values from a properties file. For example, it might look similar to the following:

```
reportInputData.varProperties = "revenue,orders,pages,customers,visits";
```

If the object reportResultPage is not referenced in the JSP, then the controller command sets it to use the generic view provided by the reporting framework to display the report. By setting reportResultPage.cmd, you have the ability to specify which view to use.

The code sample below shows an example of a JSP file used to gather input data for a report:

```
<!-- ========================================================================
  Licensed Materials - Property of IBM

  5724-A18

  (c) Copyright IBM Corp. 2001

  US Government Users Restricted Rights - Use, duplication or
  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 ------------------------------------------------------------------------------
 OrderSummaryReportInputView.jsp
 ==============================================================================-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<%@page import="java.util.*" %>
<%@page import="com.ibm.commerce.tools.util.*" %>
<%@page import="com.ibm.commerce.tools.xml.*" %>

<%@include file="common.jsp" %>
<%@include file="ReportStartDateEndDateHelper.jsp" %>
<%@include file="ReportFrameworkHelper.jsp" %>

<HTML>
<HEAD>
   <%=fHeader%>

   <TITLE><%=reportsRB.get("OrderSummaryReportInputViewTitle")%></TITLE>

   <SCRIPT SRC="/wcs/javaScript/tools/common/Util.js"></SCRIPT>
   <SCRIPT SRC="/wcs/javaScript/tools/common/DateUtil.js"></SCRIPT>
   <SCRIPT SRC="/wcs/javaScript/tools/common/SwapList.js"></SCRIPT>
   <SCRIPT SRC="/wcs/javascript/tools/reporting/ReportHelpers.js"></SCRIPT>

   <SCRIPT>

      ////////////////////////////////////////////////////////////////////
      // Call the initialize routines for the various elements of the page
      ////////////////////////////////////////////////////////////////////
      function initializeValues()
      {
         onLoadStartDateEndDate("enquiryPeriod");
         if (parent.setContentFrameLoaded) parent.setContentFrameLoaded(true);
      }

      ////////////////////////////////////////////////////////////////////
      // Call the save routines for the various elements of the page
      ////////////////////////////////////////////////////////////////////
      function savePanelData()
      {
         saveStartDateEndDate("enquiryPeriod");

         ////////////////////////////////////////////////////////////////
         // Specify the report framework particulars
         ////////////////////////////////////////////////////////////////
         setReportFrameworkOutputView("DialogView");
         setReportFrameworkParameter("XMLFile","reporting.OrderSummaryReportOutputDialog");
         setReportFrameworkReportXML("reporting.OrderSummaryReport");
         setReportFrameworkReportName("OrderSummaryReport");

         ////////////////////////////////////////////////////////////////
         // Specify the report specific parameters and save
         ////////////////////////////////////////////////////////////////
         setReportFrameworkParameter("StartDate", returnStartDateAsJavaTimestamp("enquiryPeriod"));
         setReportFrameworkParameter("EndDate", returnEndDateAsJavaTimestamp("enquiryPeriod"));
         saveReportFramework();
         return true;
      }

      ////////////////////////////////////////////////////////////////////
      // Call the validate routines for the various elements of the page
      ////////////////////////////////////////////////////////////////////
```

```
      function validatePanelData()
      {
         if (validateStartDateEndDate("enquiryPeriod") == false) return false;
         return true;
      }

</SCRIPT>
</HEAD>

<BODY ONLOAD="initializeValues()" CLASS=content>

   <H1><%=reportsRB.get("OrderSummaryReportInputViewTitle") %></H1>
   <i><%=reportsRB.get("OrderSummaryReportDescription")%></i>
   <p>

   <DIV ID=pageBody STYLE="display: block; margin-left: 20">
      <%=generateStartDateEndDate("enquiryPeriod", reportsRB, null)%>
   </DIV>

</BODY>
</HTML>
```

Reporting framework commands

The reporting framework uses the following commands shipped with WebSphere Commerce:

View Commands

Table 9. View commands used by the reporting framework

View Name	JSP file
ReportRedirectView	\tools\reporting\ReportRedirect.jsp
ReportGenericView	\tools\reporting\ReportGenericView.jsp

Controller Commands

Table 10. Controller commands used by the reporting framework

URL	Interface
GenericReportController	com.ibm.commerce.tools.reporting.command. GenericReportControllerCmd

For more information, refer to the JavaDoc help shipped with WebSphere Commerce for the following packages:

- com.ibm.commerce.tools.reporting.commands
- com.ibm.commerce.tools.reporting.framework
- com.ibm.commerce.tools.reporting.reports
- com.ibm.commerce.tools.reporting.util

Reporting framework object model

You must use the ReportDataBean when creating your results JSP file. The populate() method contains logic to support real-time reports. The following methods are available in the data bean.

Table 11.

Method name	Return type	Description
populate()	void	runs the SQL query to generate the report
getErrorCode()	int	getter method for the error code
getNumberOfColumns()	int	getter method for the number of columns in the report

Table 11. (continued)

Method name	Return type	Description
getNumberOfRows()	int	getter method for the number of rows in the report
getColumnTitlesName(int)	string	getter method for the (i+1) column name
getRow(i)	hashtable	getter method for the (i+1) row
getValue(i.j)	string	getter for the (i+1,j+1) value in the report
getValue(i,keyname)	string	getter method for the (i+1) row in the column associated with the keyname
getUserDefinedParameters()	hashtable	getter method for the hashtable created from user defined parameters in *reportName*.xml
getEnv()	hashtable	getter method for the hashtable containing the input parameters defined in the input JSP file

For more information, refer to the JavaDoc help shipped with WebSphere Commerce.

Reusable components for Reporting JSP files

Report JSPs, which coordinate with Reporting Framework, provide input and output view for reports.

Each report has an input view and output view. The task or purpose of all input JSPs is the same. They share the same look and feel; they request different input criteria yet from a shared input widget pool. On the other hand, output views have common structure and formats. Based on the nature of reports, all reusable parts of report JSPs are built as sharable components. Those components are built based on the following file structure and naming convention.

Where *ReportName* should be replaced with the new report's name, and *InputComponent* should be replaced with the input component's name. All input, and output report JSP files utilize `Reports.properties` to resolve multilingual issues. Thus, properties files map all titles and keys used in the XML and JSP files.

Here is a list of components initially build for CSA operational reports:

ReportDaysWaitedHelper.jsp
Days waited editable box component

ReportFulfillmentHelper.jsp
Fulfillment selection component

ReportInventoryAdjustmentCodeHelper.jsp
Inventory adjustment code selection

ReportProductHelper.jsp
Select product component

ReportStartDateEndDateHelper.jsp
A pair of date input component

ReportVendorHelper.jsp
Vendor selection component

ReportFrameworkHelper.jsp
Common functions for input JSP files

ReportOutputHelper.jsp
Output page formatter component

ReportProductFindDialogView.jsp
Search criteria input page

ProductSearch.jsp
Search result selection page

`ReportProductFindView` and `ReportProductSearchView` are standalone pages used by `ReportProductHelper`.

`Common.jsp` is shared by both report input and output pages. `ReportFrameworkHelper` is shared by all report input JSP files, and `ReportOutputHelper`, applies to all report output pages.

All other JSP files are input components and can be dragged onto a report input page that requires the input.

Each report requires three XML files. *ReportName*`ReportDefinition.xml` defines SQL statements that are used for the report, and the format of each column in the report. For how to write *ReportName*`ReportDefinition.xml`, refer to **Report Framework Design documentation**.

ReportName`ReportInputDialog.xml` is a dialog definition. It has the following syntax:

```
<?xml version="1.0"?>

<dialog resourceBundle="reporting.reportStrings"
    windowTitle="ReportNameReportWindowTitle"
    finishURL="GenericReportController" >

  <panel name="report"
    url="ReportNameReportInputView"
    hasFinish="YES"
    helpKey="CM.reports.ReportNameReportInputView.Help" />

</dialog>
```

Where *ReportName* should be replaced with report name. The window title should be mapped in the `Reports.properties` file. Thus, `reporting.reportStrings` is defined in the `xml/tools/reporting/resources.xml` file, and it points to `properties/com/ibm/commerce/tools/reporting/properties/Reports.properties`. The help key is mapped in `CMHelpMap.xml`. Finally, the URL is the view command name of the report input view JSP.

ReportName`ReportOutputDialog.xml` is another dialog definition that specifies the report output properties. It takes the following format:

```
<?xml version="1.0"?>

  <dialog resourceBundle="reporting.reportStrings"
          windowTitle="ReportNameReportOutputViewTitle"
          finishURL="" >

    <panel name="report"
           url="ReportNameReportOutputView"
           passAllParameters="true"
           hasFinish="NO"
           hasCancel="NO"
```

```
        helpKey="CM.reporting.ReportNameReportOutputView.Help" />

    <button name="ReportOutputViewPrintTitle"
        action="CONTENTS.printButton()" />

    <button name="ReportOutputViewOkTitle"
        action="CONTENTS.okButton()" />

  </dialog>
```

The sample output view above has two customized buttons, **Print**, and **OK**. Their processing functions are defined in the output view JSP. Again, the window's title is mapped in the properties file while the help key is mapped in the XML file.

The design is based on the reusable component concept. An input view JSP file is composed from a set of criteria items, depending on report specification. Since criteria items may appear on input pages of different reports, each criteria item is built as a reusable component. Applying this strategy to input view JSP page design acheives the following:

1. Easy to create a new report input view.
2. All report input views have a consistent look and feel.
3. Simplified validation, load, and save functions required by the Tools Framework, because these functions are built in to each components.

The output view JSP file is also built on reusable helpers. Helpers provide all necessary formatting and converting based on data type and language preference. Data types for each column are specified in a user-defined section of a report definition XML. The coordination of helpers and XML definition makes output view creation simple while supporting sophisticated output formats.

The next section explains how to use reusable components to create report input view and output view JSP files.

To create a report input view JSP file you must import required components you need your JSP file. The following components are available:

1. DaysWaited –specifies number of days waited until today.
2. FulfillmentCenter – faciliatates fulfillment center selection.
3. InventoryAdjustment – facilitates inventory adjustment code selection.
4. StartDateEndDate – specifies a period of time.
5. Vendor – facilitates vendor selection.

You may create other components as long as meet the design strategy. We will discuss creating helpers later.

All these components provided common interface for JSP page developers:

1. JSP function:

   ```
   String generateInputComponent(String containerName,
       Hashtable reportsRB, String label1 [, String label2])
   ```

   This function creates a component on the input view. The *InputComponent* is the name of a component. Each component has a containerName, which is a unique name that identifies the JavaScript object for this component. All JavaScript functions will refer to containerName. All components require a reports resource bundle, `reportsRB` which reflects current language preferences. Each component has at least one title mapped from labels.

2. JavaScript functions:

   function onLoad*InputComponent***(containerName)**
   This function will be called when the page loads. If this is the first time the page is being loaded within the transaction then initialize the *InputComponent* (from data bean). If this page is being reloaded within the transaction then retrieve the saved data.

   function validate*InputComponent***(containerName)**
   This function should be called before you submit request. It validates this component's input data. If the data is not valid, pop-up a dialog window with appropriate message to remind user. It returns true if data is valid, false if any part of the data is not valid

   function save*InputComponent***(containerName)**
   This function should be called whenever you navigate to another page different from current page. It will save current input data of the component for later retrieving back when navigates back to the current page.

   function visibleList(state)
   Call back function. It is called when the framework wants to display selection boxes or hide selection boxes on the page. This function should call:

   setSelect*Component***Visible(container, state)**
   Defined in each input component in which selection box is used. All input components have the same implementation with different names:

```
function setSelect<Component>Visible(container, state) {
    document.forms[container].ProductHelperSelectBox.style.visibility = state;
}
```

A report input JSP page should be named as *ReportName*`ReportInputView.jsp`, and they should be located in the following directory:

AIX

`/usr/WebSphere/AppServer/installedApps/`*ear_directory*`/wctools.war/tools/reporting`

400

`/QIBM/ProdData/WebAsAdv4/installedApps/`*ear_directory*`/wctools.war/tools/reporting`

Linux

`/opt/WebSphere/AppServer/installedApps/`*ear_directory*`/wctools.war/tools/reporting`

Solaris `/opt/WebSphere/AppServer/tools/ reporting/`

Windows

drive`:\WebSphere\AppServer\installedApps\`*ear_directory*`\wctools.war\tools\reporting`.

It is a dialog panel. To implement a dialog panel, refer to the Tools Framework User's Guide. A dialog panel contains two sections; an HTML contents-generating section, and a JavaScript function section. In the HTML section, you may call `generate`*InputComponent* for each component you want show on the input screen. In the JavaScript section, `initializeValue`, `savePanelData`, and `validatePanelData` should call the corresponding JavaScript functions defined in each input component. The `initializedValue` is called when the page is being loaded. The Tools Framework will call `savePanelData` and `validatePanelData`. SavePanelData should call the following report framework required JavaScript functions:

1. `setReportFrameworkOutputView("DialogView");`

2. `setReportFrameworkParameter("XMLFile","reporting.`*`OutputPanelName`*`")`
3. `setReportFrameworkReportXML("reporting.`*`ReportDefinitionXML`*`");`
4. `setReportFrameworkReportName("`*`SQLName`*`");`, which is specified in *ReportXML*

You may also call `setReportFrameworkParameter("`*`name`*`", `*`value`*`)` to set parameters that are required by the report output JSP file. It is a name and value pair. All labels passed into generator and value in `setReportFrameworkParameter` function calls are keys that will be mapped into correct string based on locale and language preference. The map is defined in the properties file `Reports_en_US.properties` in the following directory:

AIX

`/usr/WebSphere/AppServer/installedApps/`*`ear_directory`*`/properties/com/ibm/commerce/tools/reporting/properties`

400

`/QIBM/ProdData/WebAsAdv4/installedApps/`*`ear_directory`*`/properties/com/ibm/commerce/tools/reporting/properties`

Linux

`/opt/WebSphere/AppServer/installedApps/`*`ear_directory`*`/properties/com/ibm/commerce/tools/reporting/properties`

Solaris

`/opt/WebSphere/AppServer/installedApps/`*`ear_directory`*`/properties/com/ibm/commerce/tools/reporting/properties`

Windows

`drive``:\WebSphere\AppServer\installedApps\`*`ear_directory`*`\properties\com\ibm\commerce\tools\reporting\properties`

Using helpers for report input and output pages

A report input page, which is a dialog panel, must have a set of components for input criteria and implement the following four java script functions:

initializeValues()
Called every time the page is loaded.

savePanelData()
Called every time a user exits from this page.

validatePanelData()
Called before sending criteria to the reporting framework.

visibleList()
Called when the reporting framework requires a change in the visibility settings of components on this page.

To add a component onto a report input page, import the component JSP and add one JSP expression in the input page. It is named, by naming convention, `generate`*`InputComponent`* with a container name (unique to this page), a resource bundle, and one or two titles as parameters. As an example, in your input page, you will have something similar to the following:

```
<%page "ReportStartDateEndDateHelper.jsp" %>
...
<body>
...
<%=generateStartDateEndDate("RequestPeriod", reportRB, "RequestPeriodTitleKey") %>
...
</body>
```

The expression returns a string that will generate the visible component on the page. To coordinate with above three functions, callback functions are defined with the naming convention: `onLoad`*`InputComponent`*, `save`*`InputComponent`* and `validate`*`InputComponent`*. They should be called within `initializeValue`, `savePanelData`, and `validatePanelDate` respectively.

The `SavePanelData` function also saves information required by the reporting framework. Each input component provides return functions that give back inputted IDs, names, or other fields in that component. For details on these return functions, refer to any input component JSP.

The output pages are responsible for handling formatting. All formatting methods are contained in the `ReportOutputHelper`, coordinating with the report definition XML file. Only the report name needs to be specified in a report output JSP file. You can copy any report output JSP file and modify the `reportPrefix` value to reflect your report name.

The report definition XML file, however, specifies all columns and their formatting based on column type. The following is a list of column types:

Table 12.

Column Type	Is default	Customizable Properties	Default Alignment
string	Yes	maxEntryLength	Right
integer	No	setMinimumIntegerDigits setMaximumIntegerDigits	Left
decimal	No	setMinimumIntegerDigits setMaximumIntegerDigits setMinimumFractionDigits setMaximumFractionDigits	Left
currency	No	currencySymbolColumn	Left
enumeration	No		Right
date	No		Left
time	No		Left
month	No		Left

The default column type is string with HTML column options `"align=left height=20 nowrap"`. All column types can override the default column options by specifying a `<columnOptions>` tag in the column.

Optionally, all columns can also have their `displayInReport` value set to either `true` or `false`. The default is `true`, meaning that the column is displayed in the report. If the value is set to `false`, the column is hidden. This feature can be used to customize the report output view without changing the SQL query. It is also useful when formatting currency requires a reference column to tell the formatter what currency it is.

Integer, decimal, date, and time columns are formatted based on the language and currency values specified in the command context. Integer and decimal columns can also specify minimum and maximum number of digits for both the integer and fractional values.

Currency columns, by default, are formatted based on the language and currency values specified in the command context. If a currencySymbolColumn is specified

in a currecy column, the three-character currency symbol is retrieved from database, and is used to format the currency. The referred currency symbol column can be set to invisible if the report creator does not want show the currency symbol string.

Enumeration is a special column type that maps a value retrieved from database to a string specified by a key. For example, `Y`, or `N` may be retrieved from a table. These values may be mapped to more meaningful strings, such as either `Yes` or `No`, or `Approved` or `Denied` in different reports, or in different languages. To make this possible, the column can be defined as follows:

```
<columns>
    <columnKey>C2</columnKey>
    <columnName>yyyColumnTitle</columnName>
    <columnType>enumeration</columnType>
        <Y>Yes</Y>
        <N>No</N>
</columns>
```

or

```
<columns>
    <columnKey>C2</columnKey>
    <columnName>yyyColumnTitle</columnName>
    <columnType>enumeration</columnType>
        <Y>Approved</Y>
        <N>Denied</N>
</columns>
```

where the strings for `Yes`, `No`, `Approved`, and `Denied` are defined in the appropriate properties file to allow for multiple languages. If the query returns values such as 0,1,2, and so on, (digits) to which you want to map specific values, then you need to use `<X_n></X_n>` as an element. For example:

```
<columnType>enumeration</columnType>
   <X_0>ValueFor0</X_0>
   <X_1>ValueFor1</X_1>
```

Write a report utilizing reusable JSP page components

To write a report by utilizing reusable JSP page components, you must create the following:

- JSP files:
  `XXXReportInputView.jsp`
  `XXXReportOutputView.jsp`
- XML files:
  `XXXReportInputDialog.xml`
  `XXXReportDefinition.xml`
  `XXXReportOutputDialog.xml`
- Updated properties files:
  `Reports_en_US.properties`, in which you must add a section for all necessary mappings.
- Add view commands into database
  The `XXXReportInputView` and `XXXReportOutputView` commands must be added into the database.
- Set access control on the two views (`XXXReportInputView` and `XXXReportOutputView`) added in the previous step.

Chapter 13. Product Advisor

Customization examples

In this document, we provide three customization samples. They are:

- Changing the sample Product Explorer JSP file to use different operator icons.
- Change the default links used in Product Comparison.
- Render Product Explorer values without using supplied widgets.

Scenario 1: Using different operator icons

In the Product Explorer metaphor, the X coordinate of the selection on the operator icon (one image represents all operators) is used to determine which operator was selected. When replacing the image, ensure that the coordinates remain the same so that clicking on the image identifies the proper operator. The image files are in `CommerceDir\web\tools\pa\icons`. `equalone.gif` represents the =, <> operators, and `equaltoo.gif` represents the =,<>, <=, >= operators used for numeric attribute values. The expected X coordinates are:

- = 0-22
- <> 23-44
- <= 45-66
- >= 67-88

When the value selection passes an X coordinate parameter, the appropriate operator is applied to that selection.

Scenario 2: Links from the Product Comparison metaphor

The Product Comparison metaphor supports links from individual items to another page. There is currently support for only one other page (unless you create an attribute with URLs for values). The sample JSP links to the ProductDisplay page. Assign this page through the `productLinkName` property on the `ProductCompareDataBean`. In the sample, this is redirected through the `ClickInfo` command which collects statistics on the usage of the Product Comparison metaphor. When linking to another page, it may be necessary to pass through parameters from the Product Comparison page. Identify the parameters to pass through in the `productLinkParameters` property on the `ProductCompareDataBean`. Any parameter names identified here will have the parameter passed through to the link page if the parameter exists. In addition, the `productId` parameter, seen in the sample as `ECConstants.EC_PRODUCT_ID`, will be assigned the value of the catalog entry ID (`catentry_id`) for the item that the link is associated with. The `productId` allows the link page to identify which item the user selected. There is currently no support for other parameters with values unique to individual items in the Product Comparison table.

Scenario 3: Customization of Product Explorer rendering

The `ProductExploreDataBean` is rendered by the DynamicForm widget. To render yourself, either subclass the DynamicForm class (see the JavaDocs for method signatures) and override the render method, or get the data directly from the `ProductExploreDataBean`. Inside the render method, use the `getDataBean()` method to get the `ProductExploreDataBean` object. Use `ProductExploreDataBean.getFormElements()` method to get a collection of the

© Copyright IBM Corp. 2001

ColumnDataBeans representing each attribute column. Within each column, the ColumnDataBean object contains the values for that column. Use the ColumnDataBean.getColumn() method to get the collection of DsData objects containing the individual attribute values. Refer to the JavaDocs on ColumnDataBean and DsData for method information. Use DsData's getPresentationString() method to retrieve a formatted representation of the attribute value, and the getUnformattedData() method for the raw data. To create the proper parameters for your HTML form, use the ColumnDataBean.getFormElementName() method to retrieve the correct parameter name, and the DsData.getUnformattedData() method to get each value. For proper operator selection, refer to the Operator icons information.

Chapter 14. Rule Projects

The life cycle of a rule project include the following stages:

1. Create the rule project
2. Configure a rule service based on the new rule project
3. Invoke the rule service
4. Remove the rule service based on the rule project

Assumption: Information here only covers 2, 3, and 4. We assume the rule project is already created.

For information on how to create rule projects, please consult Blaze professional services.

How to configure a rule service based on customized rule project

The rules system does not distinguish between a rule project that comes with WebSphere Commerce or a customized rule project. You can always use Administration console to create, modify, or remove a rule service from a rule project. However, the customized rule project must satisfy certain requirements. The requirements are:

1. An external event posted into the rule project must trigger the execution of the rule project
2. The external event has to implement the `com.ibm.commerce.rules.InvocationContext` interface. This interface is only a simple tag interface.

How to invoke a rule service based on a customized rule project

You can invoke rule services (created from rule projects that satisfy the above two criteria) in the same fashion as the rule services created from rule projects shipped with WebSphere Commerce. That is, by calling the following task command: `com.ibm.commerce.rules.commands.InvokePersonalizationRuleServiceCommand`.

© Copyright IBM Corp. 2001

Part 2. Appendixes

© Copyright IBM Corp. 2001

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504–1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be

© Copyright IBM Corp. 2001

incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Canada Ltd.
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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 and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the

names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Credit card images, trademarks and trade names provided in this product should be used only by merchants authorized by the credit card mark's owner to accept payment via that credit card.

Trademarks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

Blaze Advisor is a trademark of HNC Software Inc., in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Netscape is a registered trademark of Netscape Communications Corporation in the United States, other countries, or both.

Oracle is a trademark or registered trademark of Oracle Corporation in the United States, other countries, or both.

Java, JavaBeans, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc.

Other company, product or service names may be the trademarks or service marks of others.

Printed in U.S.A.