IBM WebSphere Commerce

WebSphere Commerce Accelerator Reporting Framework Customization Guide

Version 5.5

IBM WebSphere Commerce

WebSphere Commerce Accelerator Reporting Framework Customization Guide

Version 5.5

Note:
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 19.

First Edition, October 2003

This edition applies to IBM WebSphere Commerce Business Edition Version 5.5, IBM WebSphere Commerce Professional Edition Version 5.5, and IBM WebSphere Commerce - Express Version 5.5 (product number 5724-A18), 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.

IBM welcomes your comments. You can send your comments by using the online IBM WebSphere Commerce documentation feedback form, available at the following URL:

www.ibm.com/software/webservers/commerce/rcf.html

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 2003. 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. 2003

About this book

Document description

This document serves as an overview of the concepts involved with customizing the WebSphere® Commerce Accelerator. It addresses the high level architecture of how the user interface interacts with the business users, and the WebSphere Commerce Server. Supplementary documents, released as they become available, build upon the knowledge developed in this document, and provide the detailed information required to customize the particular components of the WebSphere Commerce Accelerator. They also act as a resource, listing the components and assets upon which the various components depend.

Updates to this book

The latest version of this book, is available as a PDF file from the IBM® WebSphere Commerce technical library Web site:

`www.ibm.com/software/commerce/library/`

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, icons, or menu choices.
`Monospace type`	Indicates examples of text you enter exactly as shown, file names, and directory paths and names.
Italic type	Used to emphasize words. Italics also indicate names for which you must substitute the appropriate values for your system.

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

Important
These sections highlight especially important information.

Attention
These sections highlight information intended to protect your data.

Path variables

This guide uses the following variables to represent directory paths:

`WC_installdir`
This is the installation directory for WebSphere Commerce. The following are the default installation directories for WebSphere Commerce on various operating systems:

© Copyright IBM Corp. 2003

AIX `/usr/WebSphere/CommerceServer55`

400 `/QIBM/ProdData/CommerceServer55`

Linux `/opt/WebSphere/CommerceServer55`

Solaris `/opt/WebSphere/CommerceServer55`

Windows `C:\Program Files\WebSphere\CommerceServer55`

WAS_installdir
This is the installation directory for WebSphere Application Server. The following are the default installation directories for WebSphere Application Server on various operating systems:

AIX `/usr/WebSphere/AppServer`

400 `/QIBM/ProdData/WebAS5`

Linux `/opt/WebSphere/AppServer`

Solaris `/opt/WebSphere/AppServer`

Windows `C:\Program Files\WebSphere\AppServer`

WCDE_installdir
This is the installation directory for the WebSphere Commerce - Express development environment. The default installation directory for the WebSphere Commerce - Express development environment is `C:\WebSphere\CommerceDev55`.

Knowledge requirements

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

- Blaze Advisor rule technology
- HTML, JavaScript™, and XML
- Structured Query Language (SQL)
- Java™ Programming
- JavaServer Pages technology
- WebSphere Commerce Studio or WebSphere Commerce - Express Developer Edition

Please refer to the *WebSphere Commerce Programming Guide and Tutorials*for more information on customizing WebSphere Commerce. This book is available from the following Web site:

`www.ibm.com/software/commerce/library`

Part 1. Customizing the WebSphere Commerce Accelerator Reporting Framework

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

© Copyright IBM Corp. 2003

Overview of the Reporting framework

The reporting framework provides generic, customizable reporting functionality for almost any aspect of your site. The reports are accessible by any of the roles that use the WebSphere 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. The framework generates reports using data contained in the production database, and displays the reports 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 WebSphere Commerce Accelerator. Consequently, each report requires a number of associated assets. While the report itself consists of data 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 (output) 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.

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.

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>
   <resource>reporting.ReportingString</resource>
   <title></title>
   <message>messageMyReport</message>
   <columnTitles>CRITERIA,KEY,VALUE,CURRENCY,DATESTMP</columnTitles>
  </standardInfo>
  <userDefinedParameters>
  </userDefinedParameters>
```

© Copyright IBM Corp. 2003

```
 </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 vectors 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 1. View commands used by the reporting framework

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

Controller Commands

Table 2. 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 3. Methods available in the ReportDataBean

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 3. Methods available in the ReportDataBean (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 API reference information shipped with WebSphere Commerce.

Reusable components for Reporting JSP files

Report JSP files, which coordinate with the Reporting Framework, provide input and output views for reports.

Each report has an input view and an output view. The task or purpose of all input JSP files 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 JSP files are built as sharable components. Those components are built based on the following file structure and naming convention.

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 WebSphere Commerce Accelerator operational reports:

ReportDaysWaitedHelper.jsp
Days waited text entry 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 – facilitates 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 they meet the design strategy. We will discuss creating helpers later.

All these components provided common interface for JSP page developers:

1. JSP function:

   ```
   String generate InputComponent (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 the 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 a request. It validates this component's input data. If the data is not valid, it displays a dialog window with appropriate message to remind the user. It returns "true" if the data is valid, or "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 the current page. It will save the current input data of the component for later retrieval when the user 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:

/WAS_installdir/installedApps/*hostname*/WC*instance_name*.ear/CommerceAccelerator.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 calls the savePanelData and validatePanelData functions. 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:

/WAS_installdir/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 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 4.

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.

Part 2. Appendixes

© Copyright IBM Corp. 2003

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.

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 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.

© Copyright IBM Corp. 2003

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.

Trademarks

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

400	AIX	IBM
iSeries	OS/400	WebSphere

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

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

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

Printed in USA