Space Details

Key:	PentahoDoc
Name:	BI Server Documentation - Latest
Description:	Latest version of the Pentaho BI Server
Creator (Creation Date):	admin (Nov 15, 2006)
Last Modifier (Mod. Date):	admin (Nov 27, 2006)

Available Pages

- Developing a Portal Dashboard using a Filter Panel

This page last changed on Dec 04, 2006 by mdamour.

Introduction

This document describes how to create dashboards using the Pentaho BI Suite. It describes the sample dashboard provided with the Pentaho Demo server

The sample dashboard is provided in both JSP and PHP forms. Both dashboards use the same content and provide the same functionality. The JSP example shows how to use the Java API with Pentaho UI components and the PHP example shows how to use Web Services to with Pentaho UI components.

The Java API examples can be used with JSPs, Servlets, or Java applications. The Web Service examples can be used with any technology that can issue web service calls such as PHP, IBM Domino etc.

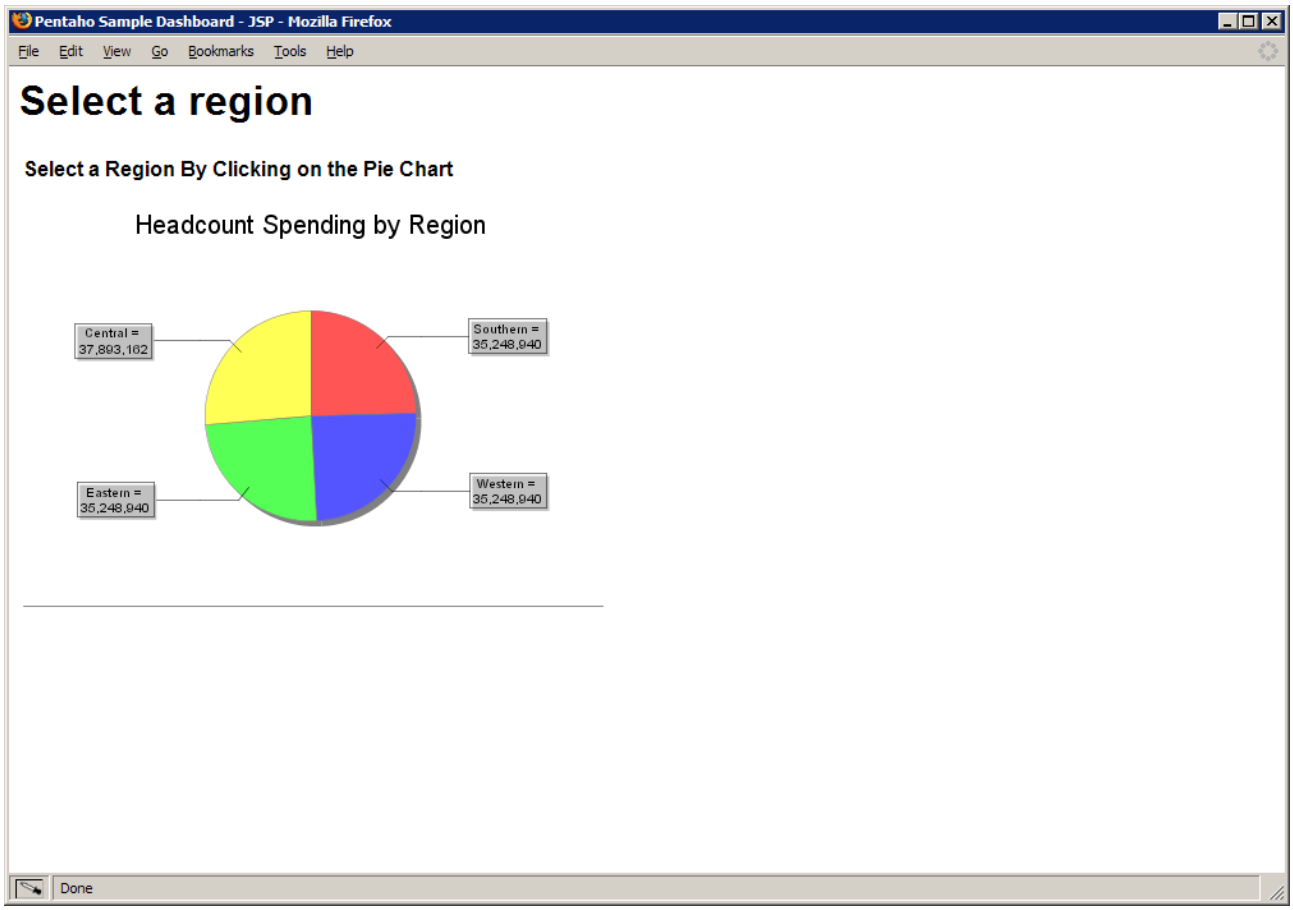
1. Sample Dashboard

This page last changed on Mar 01, 2007 by wgorman.

2. Architecture

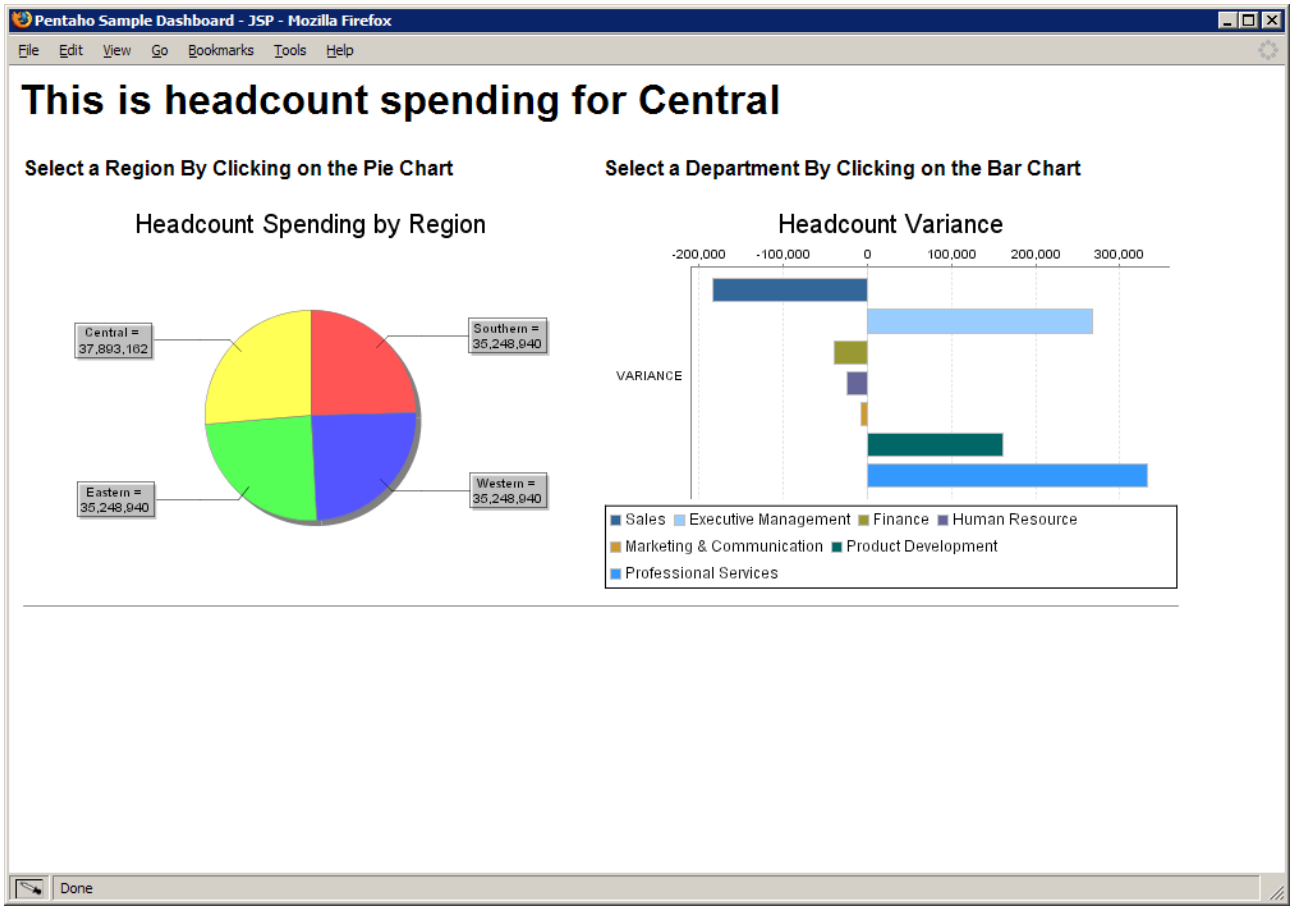
The Sample Dashboard can be found on this URL (assuming you installed the Pentaho demo on your local machine on port 8080) http://localhost:8080/pentaho/SampleDashboard

The sample dashboard first display a pie chart that shows the headcount costs for each of four regions.



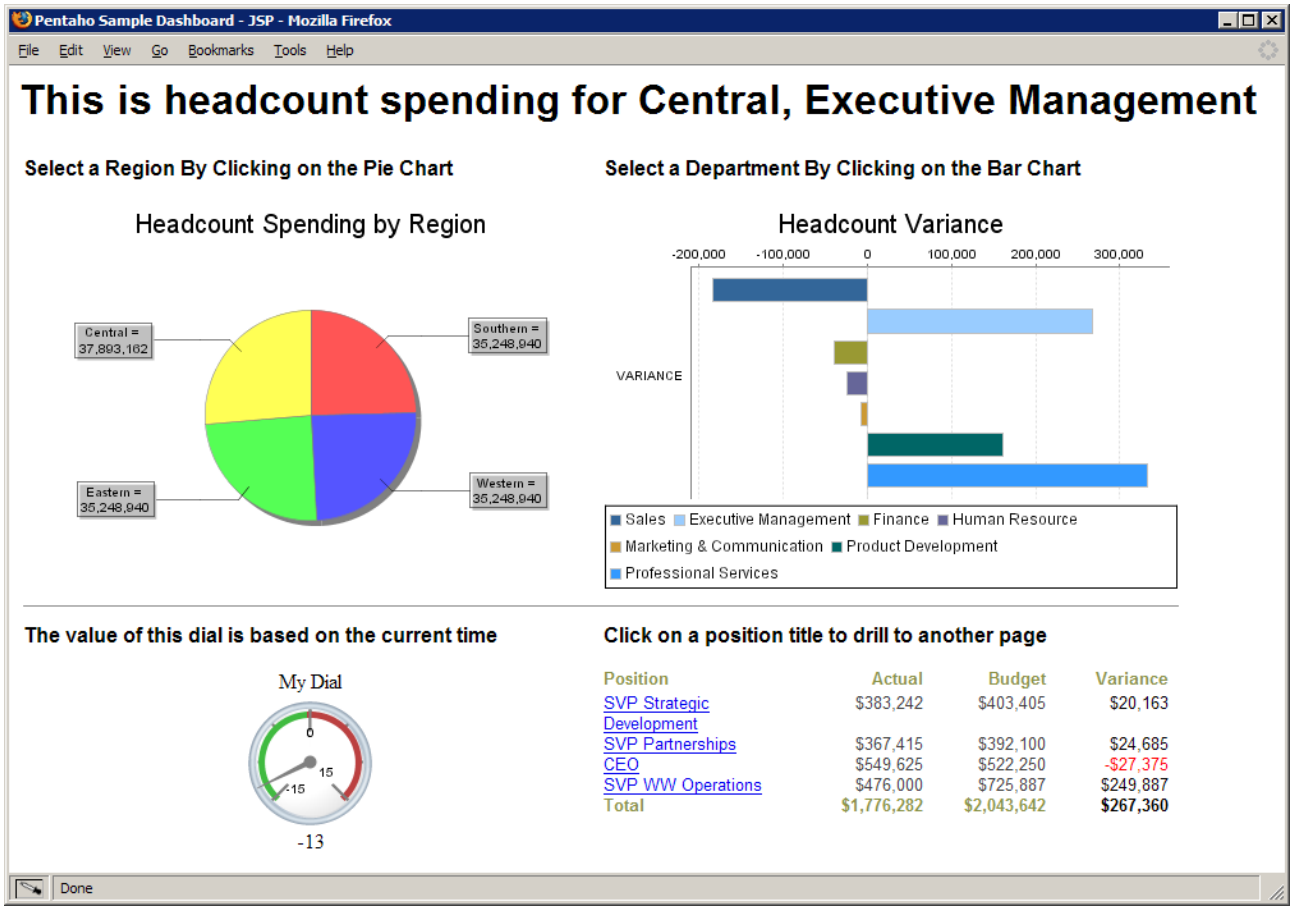
For details on how the pie chart is defined see 'Headcount Spending by Region Pie Chart' below.
For details on how the data for the pie chart is retrieved see 'Headcount Spending by Region Action Sequence' below.

When the user clicks on a slice in the pie chart a second chart displays, for each department in the selected region, the variance between the headcount budget and the actual cost.



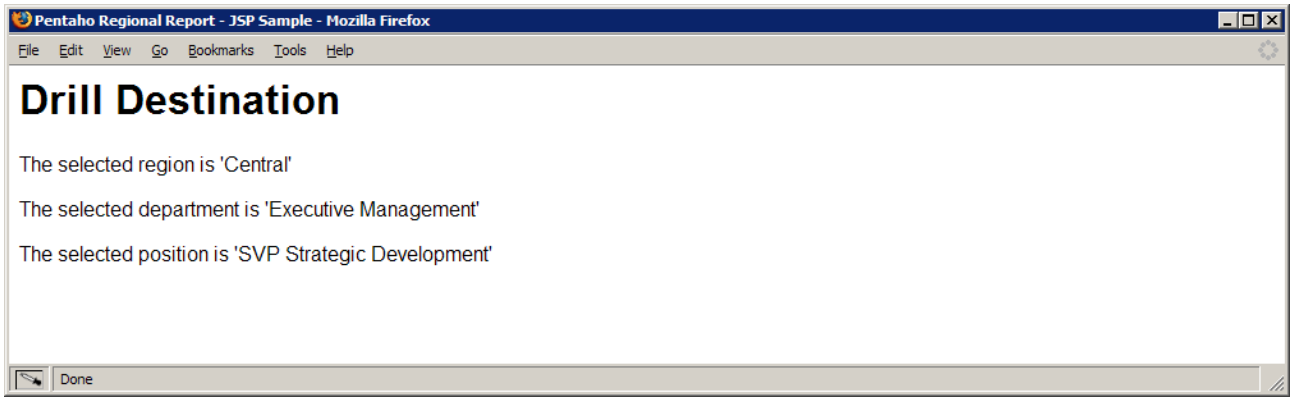
For details on how the pie chart is defined see 'Headcount Variance Pie Chart' below.
For details on how the data for the pie chart is retrieved see 'Headcount Variance Action Sequence' below.

When a user clicks on the bar for a department the dashboard displays, underneath the charts, a dial and a table of data. The dial shows a value based on the current time (so that it changes over time) table displays details about the positions within the selected department.



For details on how the dial is defined see 'Dynamic Dial' below.
For details on how the embedded report is defined see 'Embedded Report' below.

When a user clicks on the name of a position in the table of data the dashboard drills to another page and passes the region, department, and position as parameters.

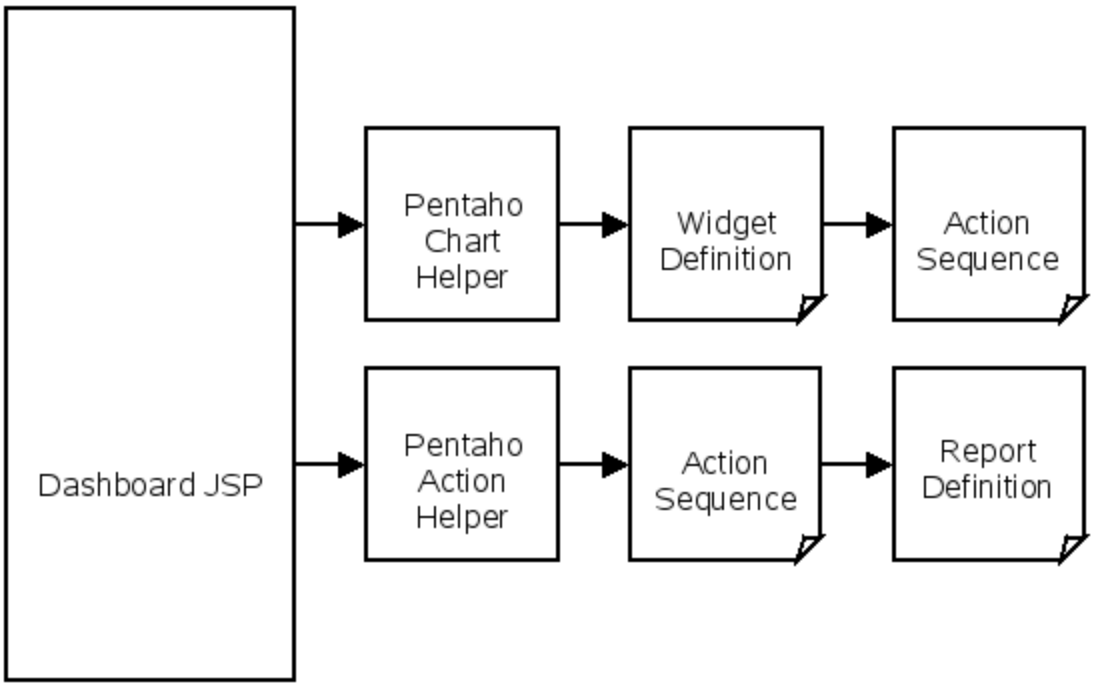


For the source code of this page see Appendix 8: Drill JSP

2. Architecture

This page last changed on Dec 04, 2006 by mdamour.

1. Sample Dashboard 3. Content Definition



3. Content Definition

This page last changed on Dec 04, 2006 by mdamour.

Dynamic Dial

This page last changed on Dec 05, 2006 by mdamour.

3. Content Definition Embedded Report

This dial is defined in
pentaho-demo/pentaho-solutions/samples/dashboard/sampledial.widget.xml

This XML file defines:

Setting	Description	Value
widget/dial/units	The units of the dial value	$
widget/dial/plot-background/texture-image	Background of the dial area	/samples/portal/dial_03.gif
widget/dial/tick-interval	The gap between ticks	5
widget/dial/tick-color	The color of the ticks	#808080
widget/dial/needle-color	The color of the needle	#808080
widget/dial/chart-background/texture-image	Background chart image	/samples/portal/dial_03.gif
widget/dial/interval/label	Interval name	'under'
widget/dial/interval/minimum	Interval lower limit	-15
widget/dial/interval/maximum	Interval upper limit	0
widget/dial/interval/color	Interval background color	#FFFFFF
widget/dial/interval/text-color	Interval text and tick color	#40BB40
widget/dial/interval/stroke-width	Interval line thickness	5

Notice that there are two intervals defined in the file: One covering the range from -15 to 0 and one covering the range from 0 to +15.

In addition to these settings defined in the widget XML file the caller specifies these settings:

Setting	Description	Value
title	The title for the dial	'My Dial'
value	The value to display on the dial	Varies over time from -10 to +15
image-width	The width of the dial image	105
image-height	The height of the dial image	105

In the sample dashboard the page creates a value for the dial based on the current time.

Embedded Report

This page last changed on Jan 30, 2007 by wgorman.

Dynamic Dial 3. Content Definition Headcount
Spending By Region
Pie Chart

The report template is defined in
pentaho-demo/pentaho-solutions/samples/dashboard/jsp/embedded_report.xml

This report template is used by the report engine (JFreeReport) to layout the embedded report. You can use the Pentaho Report Wizard to design a report definition.

You create a fragment of HTML instead of an entire HTML page you will need to edit the report definition XML file to add a property to the report configuration section in report/configuration

```
<property name="org.jfree.report.modules.output.table.html.BodyFragment">true</property>
```

To add a drill link from report content to a URL you need to edit the report definition and add a URL expression such as this one from the sample dashboard.

```
<expression name="URLCreateExpression"
class="org.jfree.report.function.TextFormatExpression">

                <properties>
                        <property
name="pattern">SampleDrill?region={0}&amp;position={1}&amp;department={2}</property>
                        <property name="field[0]">REGION</property>
                        <property name="field[1]">POSITIONTITLE</property>
                        <property name="field[2]">DEPARTMENT</property>
                </properties>

</expression>
```

Headcount Spending By Region Pie Chart

This page last changed on Jan 30, 2007 by wgorman.

Embedded Report 3. Content Definition Headcount Variance Bar Chart

This pie chart is defined in pentaho-demo/pentaho-solutions/samples/dashboard/regions.widget.xml

This XML file defines:

Setting	Description	Value
chart/title	The title for the chart	Headcount Spending by Region
chart/title-position	The position for the chart title (top, bottom, left, or right)	top
chart/title-font/font-family	The name of the title font	Ariel
chart/title-font/size	The size of the title font	20
chart/title-font/is-bold	Is the chart title bold?	false
chart/title-font/is-italic	Is the chart title italic?	false
chart/is-3D	Is the pie 3D?	false
chart/border-visible	Does the chart have a border?	false
chart/include-legend	Does the chart have a legend?	false
chart/data/data-solution	The solution for the action sequence providing data for this chart	samples
chart/data/data-path	The path to the action sequence	dashboard
chart/data/data-action	The name of the action sequence	regions_headcount_data.xaction
chart/data/data-output	The action sequence output	rule-result
chart/data/data-name	The data column containing the pie slice names	REGION
chart/data/data-value	The data column containing the pie slice values	ACTUAL
chart/data/data-orientation	The direction in which to process the data to get the pie slices (rows/columns)	rows

In addition to these settings defined in the widget XML file the caller can specify these settings:

Setting	Description	Value
drill-url	A URL template to use for generating drill paths. The URL can have replaceable	SampleDashboard?region={REGION}
inner-param	The data column containing the	REGION

	values to be replaced in the URL.	

The chart component reads all the settings from this file and then executes the specified action sequence to get the data for the chart. The action sequences in the sample dashboard only use a single component to generate the data but any sequence of BI components can be used to perform complex (e.g. scripting or in-line ETL) or remote (e.g. Web Services) tasks.

Headcount Variance Bar Chart

Headcount Spending By Region Pie Chart 3. Content Definition

This bar chart is defined in

pentaho-demo/pentaho-solutions/samples/dashboard/departments.widget.xml

This XML file defines:

Setting	Description	Value
chart/type	The type of the chart (BarChart, LineChart, AreaChart)	BarChart
chart/title	The title for the chart	Headcount Variance
chart/title-position	The position for the chart title (top, bottom, left, or right)	top
chart/title-font/font-family	The name of the title font	Ariel
chart/title-font/size	The size of the title font	20
chart/title-font/is-bold	Is the chart title bold?	false
chart/title-font/is-italic	Is the chart title italic?	false
chart/is-3D	Is the pie 3D?	false
chart/border-visible	Does the chart have a border?	false
chart/include-legend	Does the chart have a legend?	false
chart/chart-background	The background for the chart image. You can define gradients, textures, images or flat colors.	type=color #FFFFFF
chart/plot-background	The background for the plot area	type=color #EEEEEE
chart/orientation	The orientation of the chart (vertical or horizontal)	horizontal
chart/is-stacked	Is the chart stacked?	false
chart/color-palette/color	The color to use for the first series	#336699
chart/color-palette/color	The color to use for the second series	#99CCFF
chart/data/data-solution	The solution for the action sequence providing data for this chart	Samples
chart/data/data-path	The path to the action sequence	dashboard
chart/data/data-action	The name of the action sequence	regions_headcount_data.xaction

chart/data/data-output	The action sequence output	rule-result
chart/data/data-name	The data column containing the pie slice names	REGION
chart/data/data-value	The data column containing the pie slices values	ACTUAL
chart/data/data-orientation	The direction in which to process the data to get the pie slices (rows/columns)	rows

In addition to these settings defined in the widget XML file the caller specifies these settings:

Setting	Description	Value
drill-url	A URL template to use for generating drill paths. The URL can have replaceable parameters enclosed in '{' and '}'.	Dynamically set e.g. SampleDashboard?region=Central&department
inner-param	The data column containing the values to be replaced in the URL	DEPARTMENT
REGION	The filter to use in the query to select the data	Dynamically set e.g. 'Eastern'
image-width	The width of the chart image	450
image-height	The height of the chart image	300

The REGION parameter is passed to the action sequence to be used as a query filter.

4. Action Sequences

This page last changed on Dec 04, 2006 by mdamour.

Embedded Report Action Sequence

This page last changed on Dec 04, 2006 by mdamour.

Headcount Spending by Region
Action Sequence

This report is defined in
pentaho-demo/pentaho-solutions/samples/dashboard/jsp/embedded_report.xaction

You can use the Pentaho Design Studio in the Eclipse IDE to edit this file.

This action sequence contains a JFree Report component that gets data from the sample database and uses a report template to generate an HTML fragment.

The action sequence takes parameter of 'region' and 'department' that are used as filters in the query.

The report component executes this query:

```
select QUADRANT_ACTUALS.REGION,
        QUADRANT_ACTUALS.DEPARTMENT,
        QUADRANT_ACTUALS.POSITIONTITLE,
        QUADRANT_ACTUALS.ACTUAL,
        QUADRANT_ACTUALS.BUDGET,
        QUADRANT_ACTUALS.VARIANCE
from QUADRANT_ACTUALS
where QUADRANT_ACTUALS.REGION = '{region}'
and QUADRANT_ACTUALS.DEPARTMENT = '{department}'
order by QUADRANT_ACTUALS.REGION, QUADRANT_ACTUALS.DEPARTMENT
```

and uses a report template
pentaho-demo/pentaho-solutions/samples/dashboard/jsp/embedded_report.xml (see Embedded Report above)

The report component returns HTML in an output called 'report' that the action sequence directs to the response's output stream.

Headcount Spending by Region Action Sequence

This page last changed on Dec 04, 2006 by mdamour.

Embedded Report Action Sequence | Headcount Variance Action Sequence

This action sequence is defined in:
pentaho-demo/pentaho-solutions/samples/dashboard/regions_headcount_data.xaction

You can use the Pentaho Design Studio in the Eclipse IDE to edit this file.

This action sequence contains a SQL Query Rule that gets some data from the sample database that is provided with the Pentaho demo server. The action sequence does not take any parameters.

The SQL Query Rule executes this query:

```
select REGION, sum(ACTUAL) as ACTUAL
from quadrant_actuals
group by REGION
order by ACTUAL
```

and returns the data in an output called 'rule-result'. This data is used to populate the 'Headcount Spending By Region' pie chart.

Headcount Variance Action Sequence

This page last changed on Dec 04, 2006 by mdamour.

Headcount Spending by Region
Action Sequence

This action sequence is defined in:
pentaho-demo/pentaho-solutions/samples/dashboard/department_variance_data.xaction

You can use the Pentaho Design Studio in the Eclipse IDE to edit this file.

This action sequence contains a SQL Query Rule that gets some data from the sample database that is provided with the Pentaho demo server.

The action sequence takes a parameter of REGION which is used as a filter in the query.

The SQL Query Rule executes this query:

```
select department, sum(variance)
from quadrant_actuals
where region='{REGION}'
group by department
```

and returns the data in an output called 'rule-result'. This data is used to populate the 'Headcount Variance By Region' pie chart.

5. Customizing the PCI JSP Dashboard

This page last changed on Feb 01, 2007 by wgorman.

Example 1: Changing a Pie Chart to a Bar Chart

This example describes the changes necessary in the SampleDashboard solution to convert a pie chart into a bar chart. To change the widget xml and java code in the `SampleDashboard.jsp` must be updated to accommodate this.

Step 1: Updating the `SampleDashboard.jsp` file

Instead of making a call to ChartHelper.doPieChart(), make a java call to ChartHelper.doChart(). The method signatures are identical, taking in the location of the widget xml, customized parameters, a string buffer to write the output to, the user session, a messages object to store any messages, and a logger object if available.

Modify the `widget.xml` to point to our `new_regions.widget.xml` file which will be created in our next step.

Bar Charts are slightly different then pie charts when referring to their templated inner param. Replace the "{REGIONS}" portion of the "drill-url" parameter with "{SERIES}", which allows the chart to appropriately generate the outgoing link.

Step 2: Creating the `new_regions.widget.xml` file

The Pie Chart and Bar Chart <chart> xml files contain a different set of parameters for customization. Instead of going over all the parameters, let's copy `departments.widget.xml`, which is a pre-existing bar chart example, to `new_regions.widget.xml`.

Notice the chart-type element, along with the various style elements used to customize the bar chart. Modify the title element back to "Headcount Spending by Region" and the data -> data-action element back to "regions_headcount_data.xaction". We now have a configured bar chart.

Note: You must refresh the solution repository in the admin tools to have the changes to the widget take effect.

Example 2: Swapping Regions and Departments

This example describes swapping regions with departments in the dashboard.

Step 1: Updating the `SampleDashboard.jsp` file

If starting from Example 1 above, remove the changes you made to the JSP from the example.

Replace this segment of code:

```
String title = "Select a region";
 if( department != null ) {
       title = "This is headcount spending for " + region + ", " + department;
 }
 else if ( region != null ) {
       title = "This is headcount spending for " + region;
 }
```

With this:

```
String title = "Select a department";
 if( region != null ) {
    title = "This is headcount spending for " + department + ", " + region;
 }
 else if ( department != null ) {
    title = "This is headcount spending for " + department;
 }
```

To change the region pie chart over to departments:

Replace the "Select a Region By Clicking on the Pie Chart" with "Select a Department By Clicking on the Pie Chart"

Replace the "SampleDashboard?region={REGION}" with "SampleDashboard?department={DEPARTMENT}"

Replace the line

```
parameters.setParameter( "inner-param", "REGION");
```

with

```
parameters.setParameter( "inner-param", "DEPARTMENT");
```

Replace the "regions.widget.xml" with "new_departments.widget.xml"

To change the department bar chart over to regions:

Replace the line

```
if( region != null ) {
```

with

```
if( department != null ) {
```

Replace the "Select a Department By Clicking on the Bar Chart " with "Select a Region By Clicking on the Bar Chart"

Replace the line

```
parameters.setParameter( "drill-url", "SampleDashboard?region="+region+"&department={SERIES}"
);
```

with

```
parameters.setParameter( "drill-url",
"SampleDashboard?department="+department+"&region={SERIES}" );
```

Replace the line

```
parameters.setParameter( "REGION", region );
```

with

```
parameters.setParameter( "DEPARTMENT", department );
```

Replace the line

```
parameters.setParameter( "outer-params", "REGION" );
```

with

```
parameters.setParameter( "outer-params", "DEPARTMENT" );
```

Replace the line

```
parameters.setParameter( "inner-param", "DEPARTMENT");
```

with

```
parameters.setParameter( "inner-param", "REGION");
```

Replace the "departments.widget.xml" with "new_regions.widget.xml"

Replace the two lines containing

```
if( department != null ) {
```

with

```
if( region != null ) {
```

Step 2: Creating the `new_departments.widget.xml` file

Copy the `regions.widget.xml` file to `new_departments.widget.xml`.

Replace the "Headcount Spending by Region" with "Headcount Spending by Department"

Replace the "regions_headcount_data.xaction" with "departments_headcount_data.xaction"

Replace the line

```
<data-name>REGION</data-name>
```

with

```
<data-name>DEPARTMENT</data-name>
```

Step 3: Creating the departments_headcount_data.xaction file

Copy the `regions_headcount_data.xaction` file to `departments_headcount_data.xaction`

Replace the "Return the actual headcount costs total for each region" with "Return the actual headcount costs total for each department"

Replace the SQL Query

```
select REGION, sum(ACTUAL) as ACTUAL from QUADRANT_ACTUALS group by REGION order by ACTUAL
```

with

```
select DEPARTMENT, sum(ACTUAL) as ACTUAL from QUADRANT_ACTUALS group by DEPARTMENT order by
ACTUAL
```

Step 4: Creating the new_regions.widget.xml file

Copy the `departments.widget.xml` file to `new_regions.widget.xml` file

Replace the line

```
<data-action>department_variance_data.xaction</data-action>
```

with

```
<data-action>region_variance_data.xaction</data-action>
```

Step 5: Creating the region_variance_data.xaction file

Copy the `department_variance_data.xaction` file to `region_variance_data.xaction`

Replace the "Return the variance between headcount actual and budget for every department in the specified region" with "Return the variance between headcount actual and budget for every region in the specified department"

Replace

```
<inputs>
    <REGION type="string">
      <sources>
        <request>REGION</request>
      </sources>
    </REGION>
  </inputs>
```

with

```
<inputs>
    <DEPARTMENT type="string">
      <sources>
        <request>DEPARTMENT</request>
      </sources>
    </DEPARTMENT>
  </inputs>
```

Replace "Define an input called 'REGION'." with "Define an input called 'DEPARTMENT'."

Replace

```
<REGION type="string"/>
```

with

```
<DEPARTMENT type="string"/>
```

Replace the SQL Query

```
select department, sum(variance) from QUADRANT_ACTUALS where region in ( {PREPARE:REGION} )
```

```
group by department
```

with

```
select region, sum(variance) from QUADRANT_ACTUALS where department in ( {PREPARE:DEPARTMENT} )
group by region
```

Appendix 01 Dashboard JSP

This page last changed on Jan 30, 2007 by wgorman.

5. Customizing the PCI JSP Dashboard

Appendix 02 Pie Chart Definition

This is the source for the JSP Sample Dashboard.

```
<%@ page language="java"
        import="java.util.ArrayList,
        java.util.Date,
        java.io.ByteArrayOutputStream,
        org.pentaho.core.ui.SimpleUrlFactory,
        org.pentaho.messages.Messages,
        org.pentaho.core.system.PentahoSystem,
        org.pentaho.ui.component.DashboardWidgetComponent,
        org.pentaho.core.solution.HttpRequestParameterProvider,
        org.pentaho.core.solution.HttpSessionParameterProvider,
        org.pentaho.core.session.IPentahoSession,
        org.pentaho.core.util.UIUtil,
        org.pentaho.util.VersionHelper,
        org.pentaho.messages.util.LocaleHelper,
        org.pentaho.core.solution.ActionResource,
        org.pentaho.core.solution.IActionResource,
        org.pentaho.core.solution.SimpleParameterProvider,
        org.pentaho.ui.ChartHelper,
        java.io.*"
         %><%

/*
 * Copyright 2006 Pentaho Corporation.  All rights reserved.
 * This software was developed by Pentaho Corporation and is provided under the terms
 * of the Mozilla Public License, Version 1.1, or any later version. You may not use
 * this file except in compliance with the license. If you need a copy of the license,
 * please go to http://www.mozilla.org/MPL/MPL-1.1.txt. The Original Code is the Pentaho
 * BI Platform.  The Initial Developer is Pentaho Corporation.
 *
 * Software distributed under the Mozilla Public License is distributed on an "AS IS"
 * basis, WITHOUT WARRANTY OF ANY KIND, either express or  implied. Please refer to
 * the license for the specific language governing your rights and limitations.
 *
 * Created Feb 16, 2006
 * @author James Dixon
 */

/*
 * This JSP is an example of how to use Pentaho components to build a dashboard.
 * The script in this file controls the layout and content generation of the dashboard.
 * See the document 'Dashboard Builder Guide' for more details
 */

        // set the character encoding e.g. UFT-8
 response.setCharacterEncoding(LocaleHelper.getSystemEncoding());

        // get the current Pentaho session or create a new one if needed
 IPentahoSession userSession = UIUtil.getPentahoSession( request );
        %>
<html>
        <head>
                <title>Pentaho Sample Dashboard - JSP</title>
        </head>
        <body>

        <%
        // See if we have a 'department' parameter
 String department = request.getParameter("department");
        // See if we have a 'region' parameter
 String region = request.getParameter("region");
```

```
        // Create the title for the top of the page
 String title = "Select a region";
        if( department != null ) {
                title = "This is headcount spending for " + region + ", " + department;
        }
        else if ( region != null ) {
                title = "This is headcount spending for " + region;
        }
        %>

<h1 style='font-family:Arial'><%= title %></h1>

<table>
        <tr>
                <td valign="top"><span style="font-family:Arial;font-weight:bold">Select a
Region By Clicking on the Pie Chart</span>

        <%
                // Make a pie chart showing the regions
         // create the parameres for the pie chart
         SimpleParameterProvider parameters = new SimpleParameterProvider();
                // define the click url template
         parameters.setParameter( "drill-url", "SampleDashboard?region={REGION}" );
                // define the slices of the pie chart
         parameters.setParameter( "inner-param", "REGION"); //$NON-NLS-1$ //$NON-NLS-2$
         // set the width and the height
         parameters.setParameter( "image-width", "450"); //$NON-NLS-1$ //$NON-NLS-2$
         parameters.setParameter( "image-height", "300"); //$NON-NLS-1$ //$NON-NLS-2$
         StringBuffer content = new StringBuffer();
                ArrayList messages = new ArrayList();
                // call the chart helper to generate the pie chart image and to get the HTML
content
         // use the chart definition in 'samples/dashboard/regions.widget.xml'
         ChartHelper.doPieChart( "samples", "dashboard", "regions.widget.xml", parameters,
content, userSession, messages, null );
        %>

                <%= content.toString() %>

                </td>
                        <td valign="top"><span style="font-family:Arial;font-weight:bold">
        <%
                if( region != null ) {
                        // if the user has clicked on a slice of the pie chart we should have a
region to work with
 %>
                        Select a Department By Clicking on the Bar Chart
        <%
                        // Make a bar chart showing the department
                 // create the parameres for the bar chart
                 parameters = new SimpleParameterProvider();
                        // define the click url template
                 parameters.setParameter( "drill-url",
"SampleDashboard?region="+region+"&amp;department={SERIES}" );
                        parameters.setParameter( "REGION", region );
                        parameters.setParameter( "outer-params", "REGION" );
                        // define the category axis of the bar chart
                 parameters.setParameter( "inner-param", "DEPARTMENT"); //$NON-NLS-1$
//$NON-NLS-2$
                 // set the width and the height
                 parameters.setParameter( "image-width", "450"); //$NON-NLS-1$ //$NON-NLS-2$
                 parameters.setParameter( "image-height", "300"); //$NON-NLS-1$ //$NON-NLS-2$
                 content = new StringBuffer();
                        messages = new ArrayList();
                        // call the chart helper to generate the pie chart image and to get the
HTML content
                 // use the chart definition in 'samples/dashboard/regions.widget.xml'
                 ChartHelper.doChart( "samples", "dashboard", "departments.widget.xml",
parameters, content, userSession, messages, null );
        %>
                        </span>
                        <br/>
                        <%= content.toString() %>
        <%
                }
        %>
        </tr>
        <tr>
```

```
                <td colspan="2" valign="top" style="font-family:Arial;font-weight:bold"><hr
size="1"/>
        </tr>
        <tr>
                <td valign="top"><span style="font-family:Arial;font-weight:bold">
        <%
                if( department != null ) {

                        // if the user has clicked on a bar of the bar chart we should have a
region and department to work with

                        // create a dial and supply a value we create from the current time
                 Date now = new Date();
                        int seconds = now.getSeconds();
                        // create a value from -15 to +15
                 int dialValue = -15+seconds/2;
                        // create the parameres for the bar chart
                 parameters = new SimpleParameterProvider();
                        // set the value displayed on the dial
                 parameters.setParameter( "value", ""+dialValue );
                        // set the title for the dial
                 parameters.setParameter( "title", "My Dial" );
                        // set the width and the height
                 parameters.setParameter( "image-width", "105"); //$NON-NLS-1$ //$NON-NLS-2$
                 parameters.setParameter( "image-height", "105"); //$NON-NLS-1$ //$NON-NLS-2$
                 content = new StringBuffer();
                        messages = new ArrayList();
                        // call the chart helper to generate the pie chart image and to get the
HTML content
                 // use the chart definition in 'samples/dashboard/regions.widget.xml'
                         ChartHelper.doDial( "samples", "dashboard", "sampledial.widget.xml",
parameters, content, userSession, messages, null );
        %>
                        The value of this dial is based on the current time
                        </span>
                        <p/>
                        <%= content.toString() %>
        <%
                }
        %>

                </td>
                <td valign="top" style="font-family:Arial;font-weight:bold">

        <%
                if( department != null ) {


                        // if the user has clicked on a bar of the bar chart we should have a
region and department to work with

                        // run a report and embed the content into this page

                        // create the parameres for the report
                 parameters = new SimpleParameterProvider();
                        // pass the region and department to the report
                 parameters.setParameter( "region", region );
                        parameters.setParameter( "department", department );
                        // create an output stream for the report content
                 ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
                        messages = new ArrayList();
                        // run the action sequence 'samples/dashboard/jsp/report.xaction'
                 ChartHelper.doAction( "samples", "dashboard/jsp", "embedded_report.xaction",
"SampleDashboard", parameters, outputStream , userSession, messages, null );
                        // write the report content into this page
 %>
                        Click on a position title to drill to another page
                        </span>
                        <p/>
                        <% out.write( outputStream.toString() ); %>
        <%
                }
        %>

                </td>
        </tr>
</table>
```

```
    </body>
    </html>
```

Appendix 02 Pie Chart Definition

This page last changed on Jan 30, 2007 by wgorman.

Appendix 01 Dashboard JSP Appendix 03 Region Data

```
<chart>
        <!--         This file defines the pie chart that shows the actual values for each
region -->
        <!-- Specify the title of the chart -->
        <title>Headcount Spending by Region</title>
        <!-- Specify the location of the title -->
        <title-position>none</title-position>
        <!-- Specify the font of the title -->
        <title-font>
                <font-family>Ariel</font-family>
                <size>20</size>
                <is-bold>false</is-bold>
                <is-italic>false</is-italic>
        </title-font>
        <width>450</width>
        <height>300</height>
        <!-- Specify the 3D-ness of the bars -->
        <is-3D>false</is-3D>
        <!-- Specify if the chart has a border and the border color -->
        <border-visible>false</border-visible>
        <border-paint>#bbbbff</border-paint>
        <!-- Specify is the chart legend should be shown -->
        <include-legend>false</include-legend>
        <!-- Specify where the data for the chart comes from -->
        <data>
                <!-- Specify the path to the action sequence that provides the data -->
                <data-solution>samples</data-solution>
                <data-path>dashboard</data-path>
                <data-action>regions_headcount_data.xaction</data-action>
                <!-- Specify the output of the action sequence that contains the data -->
                <data-output>rule-result</data-output>
                <data-name>POSITIONTITLE</data-name>
                <data-value>ACTUAL</data-value>
                <!-- Specify whether to get the pie series from the rows or columns -->
                <data-orientation>columns</data-orientation>
        </data>
</chart>
```

Appendix 03 Region Data

This page last changed on Jan 30, 2007 by wgorman.

Appendix 02 Pie Chart Definition Appendix 04 Department Bar Chart Definition

```
<action-sequence>

        <version>1</version>
        <title>Regions and departments</title>
        <logging-level>debug</logging-level>
        <documentation>
                <author>James Dixon</author>
                <description>Return the actual headcount costs total for each
region</description>
                <help></help>
        </documentation>

        <inputs/>

        <!-- Define an output called 'rule-result' -->
        <outputs>
                <rule-result>
                        <type>list</type>
                </rule-result>
        </outputs>

        <!-- This action sequence does not require any external resources -->
        <resources/>

        <actions>
                <action-definition>
                        <action-inputs/>
                        <!-- Define a local output called 'rule-result' -->
                        <action-outputs>
                                <rule-result type="list"/>
                        </action-outputs>

                        <!-- Specify the component to execute -->
                        <component-name>SQLLookupRule</component-name>
                        <action-type>rule</action-type>

                        <!-- Define the settings for the component -->
                        <component-definition>
                                <!-- Define the datasource for the query -->
                                <jndi>SampleData</jndi>
                                <!--         Define the query to execute.  Note the parameter
{REGION}  in the query -->
                                <query><![CDATA[
                                        select REGION, sum(ACTUAL)
                                        from quadrant_actuals
                                        group by REGION
                                        order by ACTUAL]]>
                                </query>
                        </component-definition>
                </action-definition>
        </actions>
</action-sequence>
```

Appendix 04 Department Bar Chart Definition

This page last changed on Jan 30, 2007 by wgorman.

Appendix 03 Region Data Appendix 05 Department Variance Data

```
<chart>
	<!--		This file defines the bar chart that shows the actual-to-budget variance
for each department -->
	<!-- Define the chart type -->
	<chart-type>BarChart</chart-type>
	<!-- Specify the title of the chart -->
	<title>Headcount Variance</title>
	<!-- Specify the location of the title -->
	<title-position>TOP</title-position>
	<!-- Specify the font of the title -->
	<title-font>
		<font-family>Ariel</font-family>
		<size>20</size>
		<is-bold>false</is-bold>
		<is-italic>false</is-italic>
	</title-font>
	<chart-background-color>#FF80FF</chart-background-color>
	<plot-background-color>#FFFF00</plot-background-color>
	<!-- Specify the orientation of the bars -->
	<orientation>Horizontal</orientation>
	<!-- Specify the 3D-ness of the bars -->
	<is-3D>false</is-3D>
	<!-- Specify if the bars are stacked -->
	<is-stacked>false</is-stacked>
	<!-- Specify if the chart has a border and the border color -->
	<border-visible>true</border-visible>
	<border-paint>#000000</border-paint>
	<!-- Specify is the chart legend should be shown -->
	<include-legend>true</include-legend>
	<!-- Specify the color palette for the chart -->
	<color-palette>
		<color>#336699</color>
		<color>#99CCFF</color>
		<color>#999933</color>
		<color>#666699</color>
		<color>#CC9933</color>
		<color>#006666</color>
		<color>#3399FF</color>
		<color>#993300</color>
		<color>#CCCC99</color>
		<color>#666666</color>
		<color>#FFCC66</color>
		<color>#6699CC</color>
		<color>#663366</color>
	</color-palette>
	<!-- Specify where the data for the chart comes from -->
	<data>
		<!-- Specify the path to the action sequence that provides the data -->
		<data-solution>samples</data-solution>
		<data-path>dashboard</data-path>
		<data-action>department_variance_data.xaction</data-action>
		<!-- Specify the output of the action sequence that contains the data -->
		<data-output>rule-result</data-output>
		<!-- Specify whether to get the chart series from the rows or columns -->
		<data-orientation>rows</data-orientation>
	</data>
</chart>
```

Appendix 05 Department Variance Data

This page last changed on Jan 30, 2007 by wgorman.

Appendix 04 Department Bar Chart Definition | Appendix 06 Dial Definition

This action sequence executes a SQL query to get the variance between headcount actual and budget for each department within a specified region.
The action sequence is defined in

~/pentaho-demo/pentaho-solutions/samples/dashboard/department_variance_dial.xaction.

Use the Pentaho BI Studio

```
<?xml version="1.0" encoding="UTF-8"?>
<action-sequence>
        <version>1</version>
        <title>Regions and departments</title>
        <logging-level>debug</logging-level>
        <documentation>
                <author>James Dixon</author>
                <description>Return the variance between headcount actual and budget for every
department in the specified region</description>
                <help>just testing...</help>
        </documentation>
        <!-- Define an input called 'REGION'. This will be passed in when the user clicks on a
slice of the pie chart -->

        <inputs>
                <REGION type="string">
                        <sources>
                                <request>REGION</request>
                        </sources>
                </REGION>
        </inputs>
        <!-- Define an output called 'rule-result' -->
        <outputs>
                <rule-result>
                        <type>list</type>
                </rule-result>
        </outputs>
        <!-- This action sequence does not require any external resources -->
        <resources/>
        <actions>
                <action-definition>
                        <!-- Define a local input called 'REGION' -->
                        <action-inputs>
                                <REGION type="string"/>
                        </action-inputs>
                        <!-- Define a local output called 'rule-result' -->
                        <action-outputs>
                                <rule-result type="list"/>
                        </action-outputs>
                        <!-- Specify the component to execute -->
                        <component-name>SQLLookupRule</component-name>
                        <action-type>rule</action-type>
                        <!-- Define the settings for the component -->
                        <component-definition>
                                <!-- Define the datasource for the query -->
                                <jndi>SampleData</jndi>
                                <!--         Define the query to execute. Note the parameter
{REGION} in the query -->
                                </query>
                        </component-definition>
                </action-definition>
        </actions>
</action-sequence>
```

Appendix 06 Dial Definition

This page last changed on Jan 30, 2007 by wgorman.

Appendix 05 Department Variance Data

Appendix 07 Embedded Report Template

```
<widget>
        <dial>
                <name>Gauge 1</name>
                <units>$</units>
                <!--  this is the background for the whole image -->
                <!-- background-color>#ffffff</background-color -->
                <!--  this ia the background for the dial -->
                <!-- plot-background-color>#777700</plot-background-color -->
                <plot-background type="texture">
                        <texture-image>/samples/portal/dial_03.gif</texture-image>
                </plot-background>
                <tick-interval>5</tick-interval>
                <value-color>#9999bb</value-color>
                <tick-color>#808080</tick-color>
                <!--  this is the color of the needle -->
                <needle-color>#808080</needle-color>
                <chart-background type="texture">
                        <texture-image>/samples/portal/dial_03.gif</texture-image>
                </chart-background>
                <!--         intervals define ranges on the dial that are  colored differently
from the dial background -->

                <interval>
                        <label>under</label>
                        <!-- this is the value that the range starts at -->
                        <minimum>-15</minimum>
                        <!-- this is the value that the range stops at -->
                        <maximum>0</maximum>
                        <!--  this is the color of the range -->
                        <color>#ffffff</color>
                        <!--         this is the color of the text for the range value and tick
marks -->
                        <text-color>#40bb40</text-color>
                        <stroke-width>5</stroke-width>
                </interval>

                <interval>
                        <label>over</label>
                        <minimum>0</minimum>
                        <maximum>15</maximum>
                        <color>#ffffff</color>
                        <text-color>#bb4040</text-color>
                        <stroke-width>5</stroke-width>
                </interval>
        </dial>
</widget>
```

Appendix 07 Embedded Report Template

This page last changed on Jan 30, 2007 by wgorman.

Appendix 06 Dial Definition

Appendix 08 Drill JSP

```
<?xml version="1.0" encoding="UTF-8"?>

<action-sequence>
        <version>1</version>
        <title>JFreeReport HTML Example</title>
        <logging-level>debug</logging-level>
        <documentation>
                <author>James Dixon</author>
                <description><![CDATA[
                        This is an example of an HTML report produced by JFreeReport.
                        <p/>It shows the actual headcount cost, budgeted headcount
                        cost, and variance for every position in the specified
                        department and region
                ]]></description>
                <icon>/style/icons/jfree1.png</icon>
                <help></help>
        </documentation>
        <!--         Define an input called 'region' and an input called 'department'. These
will be passed in when the user clicks on a bar on the bar chart -->

        <inputs>
                <region type="string">
                        <sources>
                                <request>region</request>
                        </sources>
                </region>
                <department type="string">
                        <sources>
                                <request>department</request>
                        </sources>
                </department>
        </inputs>
        <!-- Define an output called 'report' -->
        <outputs>
                <report type="content">
                        <destinations>
                                <response>content</response>
                        </destinations>
                </report>
        </outputs>
        <!-- This action sequence uses a report definition file embedded_report.xml -->
        <resources>
                <report-definition>
                        <solution-file>
                                <location>embedded_report.xml</location>
                                <mime-type>text/xml</mime-type>
                        </solution-file>
                </report-definition>
        </resources>
        <actions>
                <action-definition>
                        <!-- Define a local inputs called 'region' and 'department' -->
                        <action-inputs>
                                <region type="string"/>
                                <department type="string"/>
                        </action-inputs>
                        <!-- Define a local output called 'report' -->
                        <action-outputs>
                                <report type="content"/>
                        </action-outputs>
                        <!-- Specify the component to execute -->
                        <component-name>JFreeReportComponent</component-name>
                        <action-type>report</action-type>
                        <!-- Define the settings for the component -->
                        <component-definition>
                                <live>false</live>
                                <!-- Define the datasource for the query -->
```

```
                                <jndi>SampleData</jndi>
                                <source>sql</source>
                                <!--         Define the query to execute. Note the parameter
{region} and {department} in the query -->
                                </query>
                                <!-- Define the content type for the report -->
                                <output-type>html</output-type>
                            </component-definition>
                    </action-definition>
            </actions>
</action-sequence>
```

Appendix 08 Drill JSP

This page last changed on Jan 30, 2007 by wgorman.

Appendix 07 Embedded Report Template

Appendix 09 Steel Wheels JSP

```
<%@ page language="java"
        import="java.util.ArrayList,
        java.util.Date,
        java.io.ByteArrayOutputStream,
        org.pentaho.core.ui.SimpleUrlFactory,
        org.pentaho.messages.Messages,
        org.pentaho.core.system.PentahoSystem,
        org.pentaho.ui.component.DashboardWidgetComponent,
        org.pentaho.core.solution.HttpRequestParameterProvider,
        org.pentaho.core.solution.HttpSessionParameterProvider,
        org.pentaho.core.session.IPentahoSession,
        org.pentaho.core.util.UIUtil,
        org.pentaho.util.VersionHelper,
        org.pentaho.messages.util.LocaleHelper,
        org.pentaho.core.solution.ActionResource,
        org.pentaho.core.solution.IActionResource,
        org.pentaho.core.solution.SimpleParameterProvider,
        org.pentaho.ui.ChartHelper,
        java.io.*"
         %><%

/*
 * Copyright 2006 Pentaho Corporation.  All rights reserved.
 * This software was developed by Pentaho Corporation and is provided under the terms
 * of the Mozilla Public License, Version 1.1, or any later version. You may not use
 * this file except in compliance with the license. If you need a copy of the license,
 * please go to http://www.mozilla.org/MPL/MPL-1.1.txt. The Original Code is the Pentaho
 * BI Platform.  The Initial Developer is Pentaho Corporation.
 *
 * Software distributed under the Mozilla Public License is distributed on an "AS IS"
 * basis, WITHOUT WARRANTY OF ANY KIND, either express or  implied. Please refer to
 * the license for the specific language governing your rights and limitations.
 *
 * Created Feb 16, 2006
 * @author James Dixon
 */

/*

This JSP page is part of the Pentaho samples that show how JSP can be
used to present and control content that is generated by the Pentaho BI Platform.

This JSP page is the page that is used after a user clicks on a link in the embedded
report that is displayed in SampleDashboard.jsp.

The region, department, and job position title are passed in as parameters on the URL

The url is formatted in the report definition file:

~/pentaho-demo/pentaho-solutions/samples/dashboard/jsp/embedded_report.xml

The url is set in the last function that is defined in the file.
The important line is the "pattern"

     <expression name="URLCreateExpression"
class="org.jfree.report.function.TextFormatExpression">
       <properties>
         <property
name="pattern">SampleDrill?region={0}&amp;amp;position={1}&amp;amp;department={2}</property>
         <property name="field[0]">REGION</property>
         <property name="field[1]">POSITIONTITLE</property>
         <property name="field[2]">DEPARTMENT</property>
       </properties>
     </expression>

You can change the url template to point to wherever you need it to
```

```
*/

        String region = request.getParameter("region");
        String department = request.getParameter("department");
        String position = request.getParameter("position");

%>
<html>
        <head>
                <title>Pentaho Regional Report - JSP Sample</title>
        </head>
        <body>
<h1 style="font-family:Arial">Drill Destination</h1>

<span style="font-family:Arial">
The selected region is '<%= region %>'
<p/>
The selected department is '<%= department %>'
<p/>
The selected position is '<%= position %>'
</span>

</body>
</html>
```

Appendix 09 Steel Wheels JSP

This page last changed on Jan 30, 2007 by wgorman.

```
<%@ page language="java"
        import="java.util.ArrayList,
        java.util.Date,
        java.io.ByteArrayOutputStream,
        org.pentaho.core.ui.SimpleUrlFactory,
        org.pentaho.messages.Messages,
        org.pentaho.core.system.PentahoSystem,
        org.pentaho.ui.component.DashboardWidgetComponent,
        org.pentaho.core.solution.HttpRequestParameterProvider,
        org.pentaho.core.solution.HttpSessionParameterProvider,
        org.pentaho.core.session.IPentahoSession,
        org.pentaho.core.util.UIUtil,
        org.pentaho.util.VersionHelper,
        org.pentaho.messages.util.LocaleHelper,
        org.pentaho.core.solution.ActionResource,
        org.pentaho.core.solution.IActionResource,
        org.pentaho.core.solution.SimpleParameterProvider,
        org.pentaho.ui.ChartHelper,
        java.io.*"
         %><%

/*
 * Copyright 2006 Pentaho Corporation.  All rights reserved.
 * This software was developed by Pentaho Corporation and is provided under the terms
 * of the Mozilla Public License, Version 1.1, or any later version. You may not use
 * this file except in compliance with the license. If you need a copy of the license,
 * please go to http://www.mozilla.org/MPL/MPL-1.1.txt. The Original Code is the Pentaho
 * BI Platform.  The Initial Developer is Pentaho Corporation.
 *
 * Software distributed under the Mozilla Public License is distributed on an "AS IS"
 * basis, WITHOUT WARRANTY OF ANY KIND, either express or  implied. Please refer to
 * the license for the specific language governing your rights and limitations.
 *
 * Created Feb 16, 2006
 * @author James Dixon  modified by Kurtis Cruzada
 */

/*
 * This JSP is an example of how to use Pentaho components to build a dashboard.
 * The script in this file controls the layout and content generation of the dashboard.
 * See the document 'Dashboard Builder Guide' for more details
 */

        // set the character encoding e.g. UFT-8
 response.setCharacterEncoding(LocaleHelper.getSystemEncoding());

        // create a new Pentaho session
 IPentahoSession userSession = UIUtil.getPentahoSession( request );
        %>
<html>
        <head>
                <title>Pentaho Sample Dashboard - JSP</title>
        </head>
        <body>
<table class="Banner" cellpadding="0" width="100%">
  <tbody>

        <%
        // See if we have a 'territory' parameter
 String territory = request.getParameter("territory");
        // See if we have a 'productline' parameter
 String productline = request.getParameter("productline");

        // Create the title for the top of the page
 String title = "Revenue Analysis";
        if( productline != null ) {
                title = "Sales for " + territory + ", " + productline;
        }
        else if ( territory != null ) {
                title = "Sales for " + territory;
```

```
        }
        %>

        <h1 style='font-family:Arial'></h1>

        <table>
                <tr>
                        <td width="75%"><img src="/sw-style/active/logo.png" border="0" /></a>
                        </td>
                        <td align="right" style="font-family:Arial;font-weight:bold"
background="/sw-style/active/banner.png" valign="top" width="25%"></a><%= title %></td>
                </tr>
                <tr>
                        <td  width="75%">
                        </td>
                        <td align="right" valign="top" width="25%"></td>
                </tr>
        </table>

        <table style="text-align: left; width: 100%; background-color: rgb(231, 238, 248);">

        <tr>
                <td valign="top" align="center">

        <%
                // Make a pie chart showing the territories
         // create the parameres for the pie chart
         SimpleParameterProvider parameters = new SimpleParameterProvider();
                // define the click url template
         parameters.setParameter( "drill-url", "SWDashboard?territory={TERRITORY}" );
                // define the slices of the pie chart
         parameters.setParameter( "inner-param", "TERRITORY"); //$NON-NLS-1$ //$NON-NLS-2$
         // set the width and the height
         parameters.setParameter( "image-width", "350"); //$NON-NLS-1$ //$NON-NLS-2$
         parameters.setParameter( "image-height", "200"); //$NON-NLS-1$ //$NON-NLS-2$
         StringBuffer content = new StringBuffer();
                ArrayList messages = new ArrayList();
                // call the chart helper to generate the pie chart image and to get the HTML
content
         // use the chart definition in 'samples/dashboard/territory.widget.xml'
         ChartHelper.doPieChart( "samples/steel-wheels", "dashboards", "territory.widget.xml",
parameters, content, userSession, messages, null );
        %>

                <%= content.toString() %>
                        <span style="font-family:Arial;font-size:12;font-weight:plain"> (Click
on a Territory) </span>
                </td>

                <td valign="top" align="center">

        <%
                        if( territory == null ) {
                        // if the user has clicked on a slice of the pie chart we should have a
territory to work with
 %>

        <%
                        // Make a bar chart showing the department
                 // create the parameres for the bar chart
                 parameters = new SimpleParameterProvider();
                        // define the click url template
                 parameters.setParameter( "drill-url",
"SWDashboard?territory="+territory+"&amp;productline={SERIES}" );
                        parameters.setParameter( "TERRITORY", territory );
                        parameters.setParameter( "outer-params", "TERRITORY" );
                        // define the category axis of the bar chart
                 parameters.setParameter( "inner-param", "TERRITORY"); //$NON-NLS-1$
//$NON-NLS-2$
                 parameters.setParameter( "inner-param", "PRODUCTLINE"); //$NON-NLS-1$
//$NON-NLS-2$
                 // set the width and the height
                 parameters.setParameter( "image-width", "550"); //$NON-NLS-1$ //$NON-NLS-2$
                 parameters.setParameter( "image-height", "200"); //$NON-NLS-1$ //$NON-NLS-2$
                 content = new StringBuffer();
                        messages = new ArrayList();
                        // call the chart helper to generate the pie chart image and to get the
HTML content
```

```
                 // use the chart definition in 'samples/dashboard/productline.widget.xml'
                 ChartHelper.doChart( "samples/steel-wheels", "dashboards",
"productline_all.widget.xml", parameters, content, userSession, messages, null );
                %>

                <%= content.toString() %>
                        <span style="font-family:Arial;font-size:12;font-weight:plain"> (Click
on a Product Line) </span>
                <%
                        }
                %>

        <%
                if( territory != null ) {
                        // if the user has clicked on a slice of the pie chart we should have a
territory to work with
 %>

        <%
                        // Make a bar chart showing the department
                 // create the parameres for the bar chart
                 parameters = new SimpleParameterProvider();
                        // define the click url template
                 parameters.setParameter( "drill-url",
"SWDashboard?territory="+territory+"&amp;productline={SERIES}" );
                        parameters.setParameter( "TERRITORY", territory );
                        parameters.setParameter( "outer-params", "TERRITORY" );
                        // define the category axis of the bar chart
                    parameters.setParameter( "inner-param", "TERRITORY"); //$NON-NLS-1$
//$NON-NLS-2$
                 parameters.setParameter( "inner-param", "PRODUCTLINE"); //$NON-NLS-1$
//$NON-NLS-2$
                 // set the width and the height
                 parameters.setParameter( "image-width", "550"); //$NON-NLS-1$ //$NON-NLS-2$
                 parameters.setParameter( "image-height", "200"); //$NON-NLS-1$ //$NON-NLS-2$
                 content = new StringBuffer();
                        messages = new ArrayList();
                        // call the chart helper to generate the pie chart image and to get the
HTML content
                 // use the chart definition in 'samples/dashboard/productline.widget.xml'
                 ChartHelper.doChart( "samples/steel-wheels", "dashboards",
"productline.widget.xml", parameters, content, userSession, messages, null );
        %>
                        <%= content.toString() %>
                                <span style="font-family:Arial;font-size:12;font-weight:plain">
(Click on a Product Line) </span>
        <%
                }
        %>
                </td>
        </tr>
        </table>

                <table style="text-align: left; width: 100%; background-color: rgb(231, 238,
248);">
        <tr>
                        <td valign="top">
                        </td>
                        <td>

        <%
                                if( productline != null ) {

                                // if the user has clicked on a bar of the bar chart we should
have a territory and productline to work with

                                // create a dial and supply a value we create from the current
time
                                 // create the parameters for the line chart
                                 parameters = new SimpleParameterProvider();
                                parameters.setParameter( "TERRITORY", territory );
                                parameters.setParameter( "outer-params", "TERRITORY" );
                                parameters.setParameter( "PRODUCTLINE", productline );
                                parameters.setParameter( "outer-params", "PRODUCTLINE" );
                                // define the category axis of the bar chart
                                 parameters.setParameter( "inner-param", "PRODUCTLINE");
//$NON-NLS-1$ //$NON-NLS-2$
                                 // set the width and the height
```

```
                                    parameters.setParameter( "image-width", "550"); //$NON-NLS-1$
//$NON-NLS-2$
                                    parameters.setParameter( "image-height", "200"); //$NON-NLS-1$
//$NON-NLS-2$
                                    content = new StringBuffer();
                                   messages = new ArrayList();
                                   // call the chart helper to generate the pie chart image and to
get the HTML content
                                    // use the chart definition in
'samples/dashboard/regions.widget.xml'
                                    ChartHelper.doChart( "samples/steel-wheels", "dashboards",
"SalesOvertime.widget.xml", parameters, content, userSession, messages, null );
        %>

                                   <span style="font-family:Arial;font-weight:bold"></span>

        <%= content.toString() %>
        <%
                }
        %>

        <%
                                   if( productline == null ) {

                                   // if the user has clicked on a bar of the bar chart we should
have a territory and productline to work with

                                   // create a dial and supply a value we create from the current
time
                                    // create the parameters for the line chart
                                    parameters = new SimpleParameterProvider();
                                   parameters.setParameter( "TERRITORY", territory );
                                   parameters.setParameter( "outer-params", "TERRITORY" );
                                   // define the category axis of the bar chart
                                    parameters.setParameter( "inner-param", "PRODUCTLINE");
//$NON-NLS-1$ //$NON-NLS-2$
                                    // set the width and the height
                                    parameters.setParameter( "image-width", "550"); //$NON-NLS-1$
//$NON-NLS-2$
                                    parameters.setParameter( "image-height", "200"); //$NON-NLS-1$
//$NON-NLS-2$
                                    content = new StringBuffer();
                                   messages = new ArrayList();
                                   // call the chart helper to generate the pie chart image and to
get the HTML content
                                    // use the chart definition in
'samples/dashboard/regions.widget.xml'
                                    ChartHelper.doChart( "samples/steel-wheels", "dashboards",
"SalesOvertime_All.widget.xml", parameters, content, userSession, messages, null );
        %>

                <span style="font-family:Arial;font-weight:bold"></span>

                <%= content.toString() %>
        <%
                }
        %>
                        </td>
                </tr>
                </table>

</body>
</html>
```

Appendix 10 Steel Wheels Territory Chart

This page last changed on Jan 30, 2007 by wgorman.

```
<chart>

        <!-- This file defines the pie chart that shows the actual values for each region -->

        <!-- Specify the title of the chart -->
        <title>Territory</title>

        <!-- Specify the location of the title -->
        <title-position>top</title-position>

        <!-- Specify the font of the title -->
        <title-font>
                <font-family>Ariel</font-family>
                <size>14</size>
                <is-bold>true</is-bold>
                <is-italic>false</is-italic>
        </title-font>

        <width>350</width>
        <height>200</height>

        <!-- Specify the 3D-ness of the bars -->
        <is-3D>false</is-3D>

        <!-- Specify if the chart has a border and the border color -->
        <border-visible>false</border-visible>

        <!-- Specify is the chart legend should be shown -->
        <include-legend>false</include-legend>

        <!-- Specify where the data for the chart comes from -->
        <data>
                <!-- Specify the path to the action sequence that provides the data -->
                <data-solution>samples/steel-wheels</data-solution>
                <data-path>dashboards</data-path>
                <data-action>Sales_by_Territory.xaction</data-action>

                <!-- Specify the output of the action sequence that contains the data -->
                <data-output>swresult</data-output>
                <data-name>TERRITORY</data-name>
                <data-value>SOLD_PRICE</data-value>

                <!-- Specify whether to get the pie series from the rows or columns -->
                <data-orientation>columns</data-orientation>
        </data>
</chart>
```

Appendix 11 Steel Wheels Product Line All Chart

This page last changed on Jan 30, 2007 by wgorman.

```
<chart>

        <!-- This file defines the bar chart that shows the actual-to-budget variance for each
department -->

        <!-- Define the chart type -->
        <chart-type>BarChart</chart-type>

        <!-- Specify the title of the chart -->
        <title>Product Line</title>

        <!-- Specify the location of the title -->
        <title-position>TOP</title-position>

        <!-- Specify the font of the title -->
        <title-font>
                <font-family>Ariel</font-family>
                <size>14</size>
                <is-bold>true</is-bold>
                <is-italic>false</is-italic>
        </title-font>
    <range-title></range-title>
        <domain-label-rotation>1.1</domain-label-rotation>
        <domain-label-rotation-dir></domain-label-rotation-dir>
        <domain-title></domain-title>
        <chart-background type="color">#FFFFFF</chart-background>
        <plot-background type="color">#EEEEEE</plot-background>
        <!-- Specify the orientation of the bars -->
        <orientation>horizontal</orientation>

        <!-- Specify the 3D-ness of the bars -->
        <is-3D>false</is-3D>

        <!-- Specify if the bars are stacked -->
        <is-stacked>false</is-stacked>

        <!-- Specify if the chart has a border and the border color -->
        <border-visible>false</border-visible>
        <border-paint>#000000</border-paint>

        <!-- Specify is the chart legend should be shown -->
        <include-legend>true</include-legend>

        <!-- Specify the color palette for the chart -->
        <color-palette>
                <color>#336699</color>
                <color>#99CCFF</color>
                <color>#999933</color>
                <color>#666699</color>
                <color>#CC9933</color>
                <color>#006666</color>
                <color>#3399FF</color>
                <color>#993300</color>
                <color>#CCCC99</color>
                <color>#666666</color>
                <color>#FFCC66</color>
                <color>#6699CC</color>
                <color>#663366</color>
        </color-palette>

        <!-- Specify where the data for the chart comes from -->
        <data>
                <!-- Specify the path to the action sequence that provides the data -->
                <data-solution>samples/steel-wheels</data-solution>
                <data-path>dashboards</data-path>
                <data-action>Sales_by_Productline_all.xaction</data-action>

                <!-- Specify the output of the action sequence that contains the data -->
                <data-output>swresult</data-output>
```

```
                <!-- Specify whether to get the chart series from the rows or columns -->
                <data-orientation>rows</data-orientation>
        </data>

</chart>
```

Appendix 12 Steel Wheels Product Line Widget

This page last changed on Jan 30, 2007 by wgorman.

```
<chart>

        <!-- This file defines the bar chart that shows the actual-to-budget variance for each
department -->

        <!-- Define the chart type -->
        <chart-type>BarChart</chart-type>

        <!-- Specify the title of the chart -->
        <title>Product Line</title>

        <!-- Specify the location of the title -->
        <title-position>TOP</title-position>

        <!-- Specify the font of the title -->
        <title-font>
                <font-family>Ariel</font-family>
                <size>14</size>
                <is-bold>true</is-bold>
                <is-italic>false</is-italic>
        </title-font>
    <range-title></range-title>
        <domain-label-rotation>1.1</domain-label-rotation>
        <domain-label-rotation-dir></domain-label-rotation-dir>
        <domain-title></domain-title>
        <chart-background type="color">#FFFFFF</chart-background>
        <plot-background type="color">#EEEEEE</plot-background>
                <!-- Specify the orientation of the bars -->
        <orientation>horizontal</orientation>

        <!-- Specify the 3D-ness of the bars -->
        <is-3D>false</is-3D>

        <!-- Specify if the bars are stacked -->
        <is-stacked>false</is-stacked>

        <!-- Specify if the chart has a border and the border color -->
        <border-visible>false</border-visible>
        <border-paint>#000000</border-paint>

        <!-- Specify is the chart legend should be shown -->
        <include-legend>true</include-legend>

        <!-- Specify the color palette for the chart -->
        <color-palette>
                <color>#336699</color>
                <color>#99CCFF</color>
                <color>#999933</color>
                <color>#666699</color>
                <color>#CC9933</color>
                <color>#006666</color>
                <color>#3399FF</color>
                <color>#993300</color>
                <color>#CCCC99</color>
                <color>#666666</color>
                <color>#FFCC66</color>
                <color>#6699CC</color>
                <color>#663366</color>
        </color-palette>

        <!-- Specify where the data for the chart comes from -->
        <data>
                <!-- Specify the path to the action sequence that provides the data -->
                <data-solution>samples/steel-wheels</data-solution>
                <data-path>dashboards</data-path>
                <data-action>Sales_by_Productline.xaction</data-action>

                <!-- Specify the output of the action sequence that contains the data -->
                <data-output>swresult</data-output>
```

```
                <!-- Specify whether to get the chart series from the rows or columns -->
                <data-orientation>rows</data-orientation>
        </data>

</chart>
```

Appendix 13 Steel Wheels Sales Overtime Widget

This page last changed on Jan 30, 2007 by wgorman.

```
<chart>

        <!-- This file defines the bar chart that shows the actual-to-budget variance for each
department -->

        <!-- Define the chart type -->
        <chart-type>LineChart</chart-type>

        <!-- Specify the title of the chart -->
        <title>Sales Trends</title>

        <!-- Specify the location of the title -->
        <title-position>TOP</title-position>

        <!-- Specify the font of the title -->
        <title-font>
                <font-family>Ariel</font-family>
                <size>14</size>
                <is-bold>true</is-bold>
                <is-italic>false</is-italic>
        </title-font>
    <range-title></range-title>
        <domain-label-rotation>1.1</domain-label-rotation>
        <domain-label-rotation-dir></domain-label-rotation-dir>
        <domain-title></domain-title>
        <chart-background type="color">#FFFFFF</chart-background>
        <plot-background type="color">#EEEEEE</plot-background>
                <!-- Specify the orientation of the bars -->
        <orientation>vertical</orientation>

        <!-- Specify the 3D-ness of the bars -->
        <is-3D>false</is-3D>

        <!-- Specify if the bars are stacked -->
        <is-stacked>false</is-stacked>

        <!-- Specify if the chart has a border and the border color -->
        <border-visible>false</border-visible>
        <border-paint>#000000</border-paint>

        <!-- Specify is the chart legend should be shown -->
        <include-legend>false</include-legend>

        <!-- Specify the color palette for the chart -->
        <color-palette>
                <color>#336699</color>
                <color>#99CCFF</color>
                <color>#999933</color>
                <color>#666699</color>
                <color>#CC9933</color>
                <color>#006666</color>
                <color>#3399FF</color>
                <color>#993300</color>
                <color>#CCCC99</color>
                <color>#666666</color>
                <color>#FFCC66</color>
                <color>#6699CC</color>
                <color>#663366</color>
        </color-palette>

        <!-- Specify where the data for the chart comes from -->
        <data>
                <!-- Specify the path to the action sequence that provides the data -->
                <data-solution>samples/steel-wheels</data-solution>
                <data-path>dashboards</data-path>
                <data-action>sales_overtime.xaction</data-action>

                <!-- Specify the output of the action sequence that contains the data -->
                <data-output>swresult</data-output>
```

```
                <!-- Specify whether to get the chart series from the rows or columns -->
                <data-orientation>columns</data-orientation>
        </data>

</chart>
```

Appendix 14 Steel Wheels Sales Overtime All Widget

This page last changed on Jan 30, 2007 by wgorman.

```
<chart>

        <!-- This file defines the bar chart that shows the actual-to-budget variance for each
department -->

        <!-- Define the chart type -->
        <chart-type>LineChart</chart-type>

        <!-- Specify the title of the chart -->
        <title>Sales Trends</title>

        <!-- Specify the location of the title -->
        <title-position>TOP</title-position>

        <!-- Specify the font of the title -->
        <title-font>
                <font-family>Ariel</font-family>
                <size>14</size>
                <is-bold>true</is-bold>
                <is-italic>false</is-italic>
        </title-font>
    <range-title></range-title>
        <domain-label-rotation>1.1</domain-label-rotation>
        <domain-label-rotation-dir></domain-label-rotation-dir>
        <domain-title></domain-title>
        <chart-background type="color">#FFFFFF</chart-background>
        <plot-background type="color">#EEEEEE</plot-background>
                <!-- Specify the orientation of the bars -->
        <orientation>vertical</orientation>

        <!-- Specify the 3D-ness of the bars -->
        <is-3D>false</is-3D>

        <!-- Specify if the bars are stacked -->
        <is-stacked>false</is-stacked>

        <!-- Specify if the chart has a border and the border color -->
        <border-visible>false</border-visible>
        <border-paint>#000000</border-paint>

        <!-- Specify is the chart legend should be shown -->
        <include-legend>false</include-legend>

        <!-- Specify the color palette for the chart -->
        <color-palette>
                <color>#336699</color>
                <color>#99CCFF</color>
                <color>#999933</color>
                <color>#666699</color>
                <color>#CC9933</color>
                <color>#006666</color>
                <color>#3399FF</color>
                <color>#993300</color>
                <color>#CCCC99</color>
                <color>#666666</color>
                <color>#FFCC66</color>
                <color>#6699CC</color>
                <color>#663366</color>
        </color-palette>

        <!-- Specify where the data for the chart comes from -->
        <data>
                <!-- Specify the path to the action sequence that provides the data -->
                <data-solution>samples/steel-wheels</data-solution>
                <data-path>dashboards</data-path>
                <data-action>sales_overtime_all.xaction</data-action>

                <!-- Specify the output of the action sequence that contains the data -->
                <data-output>swresult</data-output>
```

```
                <!-- Specify whether to get the chart series from the rows or columns -->
                <data-orientation>columns</data-orientation>
        </data>

</chart>
```

Appendix 15 Steel Wheels Sales By Territory XAction

This page last changed on Jan 30, 2007 by wgorman.

```
<?xml version="1.0" encoding="UTF-8"?>
<action-sequence>
  <name>Sales_by_Territory.xaction</name>
  <title>05. SALES BY TERRITORY</title>
  <version>1</version>
  <logging-level>ERROR</logging-level>
  <documentation>
    <author>Kurtis Cruzada</author>
    <description>List of Territories sorted by Sales. Select date range (Jan 2003 through May
2005). Includes chart.</description>
    <icon>slsbyter.png</icon>
    <help/>
  </documentation>

  <inputs/>

  <outputs>
    <!-- an output stream will be provided by default -->
    <swresult>
      <type>list</type>
    </swresult>
  </outputs>

  <resources>
    <!-- use this section to identify any files that the component needs to execute the report
-->
  </resources>

  <actions>
    <action-definition>
      <component-name>SQLLookupRule</component-name>
      <action-type>SQL Query</action-type>
      <action-inputs>
        <territory_name type="string"/>
        <productline_name type="string"/>
      </action-inputs>
      <action-outputs>
        <query-result type="result-set" mapping="swresult"/>
      </action-outputs>
      <component-definition>
        <jndi>SampleData</jndi>
        <query><![CDATA[SELECT OFFICES.TERRITORY,
SUM(ORDERDETAILS.QUANTITYORDERED*ORDERDETAILS.PRICEEACH) SOLD_PRICE FROM ORDERS INNER JOIN
ORDERDETAILS ON ORDERS.ORDERNUMBER = ORDERDETAILS.ORDERNUMBER INNER JOIN PRODUCTS ON
ORDERDETAILS.PRODUCTCODE =PRODUCTS.PRODUCTCODE  INNER JOIN CUSTOMERS ON ORDERS.CUSTOMERNUMBER
=CUSTOMERS.CUSTOMERNUMBER  INNER JOIN EMPLOYEES ON CUSTOMERS.SALESREPEMPLOYEENUMBER =
EMPLOYEES.EMPLOYEENUMBER INNER JOIN OFFICES ON EMPLOYEES.OFFICECODE=OFFICES.OFFICECODE  GROUP
BY OFFICES.TERRITORY ORDER BY 2 DESC]]></query>
      </component-definition>
    </action-definition>

  </actions>
</action-sequence>
```

Appendix 16 Steel Wheels Sales by Productline All XAction

This page last changed on Jan 30, 2007 by wgorman.

```
<?xml version="1.0" encoding="UTF-8"?>
<action-sequence>
  <name>Sales_by_Productline.xaction</name>
  <title>06. SALES BY PRODUCTLINE</title>
  <version>1</version>
  <logging-level>ERROR</logging-level>
  <documentation>
    <author>Kurtis Cruzada</author>
    <description>List of Product Lines sorted by Sales. Select date range (Jan 2003 through May
2005). Includes chart.</description>
    <icon>slsprdln.png</icon>
    <help/>
  </documentation>

  <inputs/>

  <outputs>
    <!-- an output stream will be provided by default -->
    <swresult type="result-set"/>
  </outputs>

  <resources>
    <!-- use this section to identify any files that the component needs to execute the report
-->
  </resources>

  <actions>
    <action-definition>
      <component-name>SQLLookupRule</component-name>
      <action-type>SQL Query</action-type>
      <action-inputs/>
      <action-outputs>
        <query-result type="result-set" mapping="swresult"/>
      </action-outputs>
      <component-definition>
        <jndi>SampleData</jndi>
        <query><![CDATA[SELECT PRODUCTS.PRODUCTLINE,
SUM(ORDERDETAILS.QUANTITYORDERED*ORDERDETAILS.PRICEEACH) REVENUE FROM ORDERS INNER JOIN
ORDERDETAILS ON ORDERS.ORDERNUMBER = ORDERDETAILS.ORDERNUMBER INNER JOIN PRODUCTS ON
ORDERDETAILS.PRODUCTCODE =PRODUCTS.PRODUCTCODE  INNER JOIN CUSTOMERS ON ORDERS.CUSTOMERNUMBER
=CUSTOMERS.CUSTOMERNUMBER  INNER JOIN EMPLOYEES ON CUSTOMERS.SALESREPEMPLOYEENUMBER =
EMPLOYEES.EMPLOYEENUMBER INNER JOIN OFFICES ON EMPLOYEES.OFFICECODE=OFFICES.OFFICECODE GROUP BY
PRODUCTS.PRODUCTLINE ORDER BY 2 DESC]]></query>
      </component-definition>
    </action-definition>

  </actions>
</action-sequence>
```

Appendix 17 Steel Wheels Sales by Productline XAction

This page last changed on Jan 30, 2007 by wgorman.

```
<?xml version="1.0" encoding="UTF-8"?>
<action-sequence>
  <name>Sales_by_Productline.xaction</name>
  <title>06. SALES BY PRODUCTLINE</title>
  <version>1</version>
  <logging-level>ERROR</logging-level>
  <documentation>
    <author>Kurtis Cruzada</author>
    <description>List of Product Lines sorted by Sales. Select date range (Jan 2003 through May
2005). Includes chart.</description>
    <icon>slsprdln.png</icon>
    <help/>
  </documentation>

  <inputs>
    <TERRITORY type="string">
      <sources>
        <request>TERRITORY</request>
      </sources>
      <default-value>NULL</default-value>
    </TERRITORY>
  </inputs>

  <outputs>
    <!-- an output stream will be provided by default -->
    <swresult type="result-set"/>
  </outputs>

  <resources>
    <!-- use this section to identify any files that the component needs to execute the report
-->
  </resources>

  <actions>
    <actions>
      <condition><![CDATA[TERRITORY == "NULL"]]></condition>
      <action-definition>
        <component-name>SQLLookupRule</component-name>
        <action-type>SQL Query</action-type>
        <action-inputs/>
        <action-outputs>
          <query-result type="result-set" mapping="swresult"/>
        </action-outputs>
        <component-definition>
          <jndi>SampleData</jndi>
          <query><![CDATA[SELECT PRODUCTS.PRODUCTLINE,
SUM(ORDERDETAILS.QUANTITYORDERED*ORDERDETAILS.PRICEEACH) REVENUE FROM ORDERS INNER JOIN
ORDERDETAILS ON ORDERS.ORDERNUMBER = ORDERDETAILS.ORDERNUMBER INNER JOIN PRODUCTS ON
ORDERDETAILS.PRODUCTCODE =PRODUCTS.PRODUCTCODE  INNER JOIN CUSTOMERS ON ORDERS.CUSTOMERNUMBER
=CUSTOMERS.CUSTOMERNUMBER  INNER JOIN EMPLOYEES ON CUSTOMERS.SALESREPEMPLOYEENUMBER =
EMPLOYEES.EMPLOYEENUMBER INNER JOIN OFFICES ON EMPLOYEES.OFFICECODE=OFFICES.OFFICECODE GROUP BY
PRODUCTS.PRODUCTLINE ORDER BY 2 DESC]]></query>
        </component-definition>
      </action-definition>

    </actions>

    <action-definition>
      <component-name>SQLLookupRule</component-name>
      <action-type>SQL Query</action-type>
      <action-inputs>
        <TERRITORY type="string"/>
      </action-inputs>
      <action-outputs>
        <query-result type="result-set" mapping="swresult"/>
      </action-outputs>
      <component-definition>
        <jndi>SampleData</jndi>
        <query><![CDATA[SELECT PRODUCTS.PRODUCTLINE,
SUM(ORDERDETAILS.QUANTITYORDERED*ORDERDETAILS.PRICEEACH) REVENUE FROM ORDERS INNER JOIN
```

```
   ORDERDETAILS ON ORDERS.ORDERNUMBER = ORDERDETAILS.ORDERNUMBER INNER JOIN PRODUCTS ON
   ORDERDETAILS.PRODUCTCODE =PRODUCTS.PRODUCTCODE  INNER JOIN CUSTOMERS ON ORDERS.CUSTOMERNUMBER
   =CUSTOMERS.CUSTOMERNUMBER  INNER JOIN EMPLOYEES ON CUSTOMERS.SALESREPEMPLOYEENUMBER =
   EMPLOYEES.EMPLOYEENUMBER INNER JOIN OFFICES ON EMPLOYEES.OFFICECODE=OFFICES.OFFICECODE WHERE
   TERRITORY={PREPARE:TERRITORY} GROUP BY PRODUCTS.PRODUCTLINE ORDER BY 2 DESC]]></query>
         </component-definition>
       </action-definition>

     </actions>
   </action-sequence>
```

Appendix 18 Steel Wheels Sales Overtime XAction

This page last changed on Jan 30, 2007 by wgorman.

```
<?xml version="1.0" encoding="UTF-8"?>
<action-sequence>
  <title/>
  <version>1</version>
  <logging-level>ERROR</logging-level>
  <documentation>
    <author/>
    <description>Empty blank action sequence document</description>
    <help/>
    <icon/>
  </documentation>

  <inputs>
    <TERRITORY type="string">
      <sources>
        <request>TERRITORY</request>
      </sources>
      <default-value>NA</default-value>
    </TERRITORY>
    <PRODUCTLINE type="string">
      <sources>
        <request>PRODUCTLINE</request>
      </sources>
      <default-value>NULL</default-value>
    </PRODUCTLINE>
  </inputs>

  <outputs>
    <swresult type="result-set"/>
  </outputs>

  <resources/>

  <actions>
    <actions>
      <condition><![CDATA[PRODUCTLINE != "NULL"]]></condition>
      <action-definition>
        <component-name>SQLLookupRule</component-name>
        <action-type>SQL Query</action-type>
        <action-inputs>
          <territory type="string"/>
          <productline type="string"/>
          <PRODUCTLINE type="string"/>
          <TERRITORY type="string"/>
        </action-inputs>
        <action-outputs>
          <query-result type="result-set" mapping="swresult"/>
        </action-outputs>
        <component-definition>
          <jndi>SampleData</jndi>
          <query><![CDATA[SELECT CONCAT(CONCAT( YEAR(ORDERS.ORDERDATE),'-'
),MONTH(ORDERS.ORDERDATE)) AS TIME, SUM(ORDERDETAILS.QUANTITYORDERED*ORDERDETAILS.PRICEEACH)
SOLD_PRICE
FROM ORDERS
INNER JOIN ORDERDETAILS ON ORDERS.ORDERNUMBER = ORDERDETAILS.ORDERNUMBER
INNER JOIN PRODUCTS ON ORDERDETAILS.PRODUCTCODE =PRODUCTS.PRODUCTCODE
INNER JOIN CUSTOMERS ON ORDERS.CUSTOMERNUMBER =CUSTOMERS.CUSTOMERNUMBER
INNER JOIN EMPLOYEES ON CUSTOMERS.SALESREPEMPLOYEENUMBER = EMPLOYEES.EMPLOYEENUMBER
INNER JOIN OFFICES ON EMPLOYEES.OFFICECODE=OFFICES.OFFICECODE

WHERE PRODUCTS.PRODUCTLINE={PREPARE:PRODUCTLINE} AND OFFICES.TERRITORY={PREPARE:TERRITORY}

GROUP BY
CONCAT( MONTH(ORDERS.ORDERDATE), YEAR(ORDERS.ORDERDATE))]]></query>
        </component-definition>
      </action-definition>

    </actions>

    <actions>
```

```
        <condition><![CDATA[PRODUCTLINE == "NULL"]]></condition>
        <action-definition>
          <component-name>SQLLookupRule</component-name>
          <action-type>SQL Query</action-type>
          <action-inputs>
            <TERRITORY type="string"/>
          </action-inputs>
          <action-outputs>
            <query-result type="result-set" mapping="swresult"/>
          </action-outputs>
          <component-definition>
            <jndi>SampleData</jndi>
            <query><![CDATA[SELECT CONCAT(CONCAT( YEAR(ORDERS.ORDERDATE),'-'
),MONTH(ORDERS.ORDERDATE)) AS TIME, SUM(ORDERDETAILS.QUANTITYORDERED*ORDERDETAILS.PRICEEACH)
SOLD_PRICE
FROM ORDERS
INNER JOIN ORDERDETAILS ON ORDERS.ORDERNUMBER = ORDERDETAILS.ORDERNUMBER
INNER JOIN PRODUCTS ON ORDERDETAILS.PRODUCTCODE =PRODUCTS.PRODUCTCODE
INNER JOIN CUSTOMERS ON ORDERS.CUSTOMERNUMBER =CUSTOMERS.CUSTOMERNUMBER
INNER JOIN EMPLOYEES ON CUSTOMERS.SALESREPEMPLOYEENUMBER = EMPLOYEES.EMPLOYEENUMBER
INNER JOIN OFFICES ON EMPLOYEES.OFFICECODE=OFFICES.OFFICECODE

WHERE OFFICE.TERRITORY='{TERRITORY}'

GROUP BY
CONCAT( MONTH(ORDERS.ORDERDATE), YEAR(ORDERS.ORDERDATE))]]></query>
          </component-definition>
        </action-definition>

      </actions>

    </actions>
  </action-sequence>
```

Appendix 19 Sales Overtime All XAction

This page last changed on Jan 30, 2007 by wgorman.

```
<?xml version="1.0" encoding="UTF-8"?>
<action-sequence>
  <title/>
  <version>1</version>
  <logging-level>ERROR</logging-level>
  <documentation>
    <author/>
    <description>Empty blank action sequence document</description>
    <help/>
    <icon/>
  </documentation>

  <inputs>
    <TERRITORY type="string">
      <sources>
        <request>TERRITORY</request>
      </sources>
      <default-value>NULL</default-value>
    </TERRITORY>
  </inputs>

  <outputs>
    <swresult type="result-set"/>
  </outputs>

  <resources/>

  <actions>
    <actions>
      <condition><![CDATA[TERRITORY == "NULL"]]></condition>
      <action-definition>
        <component-name>SQLLookupRule</component-name>
        <action-type>SQL Query no territory</action-type>
        <action-inputs/>
        <action-outputs>
          <query-result type="result-set" mapping="swresult"/>
        </action-outputs>
        <component-definition>
          <jndi>SampleData</jndi>
          <query><![CDATA[SELECT CONCAT(CONCAT( YEAR(ORDERS.ORDERDATE),'-'
),MONTH(ORDERS.ORDERDATE)) AS TIME, SUM(ORDERDETAILS.QUANTITYORDERED*ORDERDETAILS.PRICEEACH)
SOLD_PRICE
FROM ORDERS
INNER JOIN ORDERDETAILS ON ORDERS.ORDERNUMBER = ORDERDETAILS.ORDERNUMBER
INNER JOIN PRODUCTS ON ORDERDETAILS.PRODUCTCODE =PRODUCTS.PRODUCTCODE
INNER JOIN CUSTOMERS ON ORDERS.CUSTOMERNUMBER =CUSTOMERS.CUSTOMERNUMBER
INNER JOIN EMPLOYEES ON CUSTOMERS.SALESREPEMPLOYEENUMBER = EMPLOYEES.EMPLOYEENUMBER
INNER JOIN OFFICES ON EMPLOYEES.OFFICECODE=OFFICES.OFFICECODE


GROUP BY
CONCAT( MONTH(ORDERS.ORDERDATE), YEAR(ORDERS.ORDERDATE))]]></query>
        </component-definition>
      </action-definition>

    </actions>

    <actions>
      <condition><![CDATA[TERRITORY !== "NULL"]]></condition>
      <action-definition>
        <component-name>SQLLookupRule</component-name>
        <action-type>SQL Query with Territory</action-type>
        <action-inputs>
          <territory type="string"/>
          <productline type="string"/>
          <TERRITORY type="string"/>
        </action-inputs>
        <action-outputs>
          <query-result type="result-set" mapping="swresult"/>
        </action-outputs>
```

```
        <component-definition>
          <jndi>SampleData</jndi>
          <query><![CDATA[SELECT CONCAT(CONCAT( YEAR(ORDERS.ORDERDATE),'-'
),MONTH(ORDERS.ORDERDATE)) AS TIME, SUM(ORDERDETAILS.QUANTITYORDERED*ORDERDETAILS.PRICEEACH)
SOLD_PRICE
FROM ORDERS
INNER JOIN ORDERDETAILS ON ORDERS.ORDERNUMBER = ORDERDETAILS.ORDERNUMBER
INNER JOIN PRODUCTS ON ORDERDETAILS.PRODUCTCODE =PRODUCTS.PRODUCTCODE
INNER JOIN CUSTOMERS ON ORDERS.CUSTOMERNUMBER =CUSTOMERS.CUSTOMERNUMBER
INNER JOIN EMPLOYEES ON CUSTOMERS.SALESREPEMPLOYEENUMBER = EMPLOYEES.EMPLOYEENUMBER
INNER JOIN OFFICES ON EMPLOYEES.OFFICECODE=OFFICES.OFFICECODE

WHERE OFFICES.TERRITORY={PREPARE:TERRITORY}

GROUP BY
CONCAT( MONTH(ORDERS.ORDERDATE), YEAR(ORDERS.ORDERDATE))]]></query>
        </component-definition>
      </action-definition>

    </actions>

  </actions>
</action-sequence>
```

Developing a Portal Dashboard using a Filter Panel

This page last changed on Feb 28, 2007 by sbarkdull.

Motivation

In a portal, a digital dashboard is an aggregation of user interface components (a.k.a. portlets) that provide a view of related business data. A digital dashboard can be a useful mechanism for efficiently presenting a set of critical business data on a single page. A developer can use the Pentaho BI Platform with a JSR168 portal to implement a digital dashboard. Here I'll configure the Pentaho BI Platform Running in the JBoss Portal server to use a FilterPanelPortlet, an ActionPortlet, and some Action Sequences to implement a dashboard in a portal page.

A Simple Portal

A simplified description of a portal might be that a portal consists of a group of related portal pages. A portal page contains a set of 1 or more windows. And each window contains one portlet, and each portlet has server side logic and client side user interface controls/displays.

Dashboard

In the Pentaho BI Platform, a portal-based dashboard can be created using a filter panel and one or more portlets containing visual displays embedded in the same portal page. The visual displays are typically things like charts and reports.

Filter Panel

A filter panel consists of client side UI controls that are associated with an HTML form, and a server side FilterPanelPortlet, which is an implementation of a JSR168 GenericPortlet. When the form is submitted to the Portal server, the FilterPanelPortlet places the form data into session scoped parameters, which are then available as inputs to other Portlets/Actions Sequences in the page.

When the other Pentaho-based portlets in the page are asked to render themselves, they can render themselves based on the values of the parameters placed in the session by the filter panel.

Configuring the Portal with a Page that Implements a Dashboard

There are several steps required to configure the dashboard in the portal.

1. Modify the portlet.xml file to define the portlets in the dashboard.
2. Modify the portlet-instances.xml file to define the portlet instances.
3. Modify the pentaho-portal-object.xml file to place the portlet instances in a page in the portal.
4. Add localized Strings for portlet and portal page titles to the portlet.properties file.
5. Create a filter panel definition file in the repository.
6. Create Action Sequences to provide list-of-values (LOV) data for the filter panel's controls.
7. Create Action Sequences that are associated with the other portlets in the page. These Action Sequences respond to changes in parameters placed in the session by the filter panel and provide the view of the business data.
8. Create a JNDI accessible datasource to provide data to the Action Sequences.

A Simple Portal Dashboard Example

In this example, I am going to configure the portal to display a dashboard by:

1. adding two portlets to the portal
2. adding an instance of each of those portlets to the portal
3. adding a new page to the portal
4. adding two windows to the page
5. adding the two portlets to the two windows

Modify the portal configuration files

In the Pentaho Pre-Configured Install (PCI), the portal configuration files live in `/server/default/deploy/pentaho.war/WEB-INF`. Their names are `portlet.xml`, `portlet-instances.xml`, and `pentaho-portal-object.xml`.

- `portlet.xml`: contains portlet definitions
- `portlet-instances.xml`: contains portlet-instance definitions
- `pentaho-portal-object.xml`: contains page and window definitions

JBoss Portal Server Specific Configuration Files
The name and format of the `portlet.xml` file is identified in the JSR168 specification and is common to all JSR168 portals.
The names and format of the `portlet-instances.xml` and `pentaho-portal-object.xml` files are specific to the JBoss Portal Server.

Modify the `portlet.xml` file to define the portlets in the dashboard.

This XML fragment describes a portlet that is implemented by the `FilterPanelPortlet` class, and whose UI is specified by the `simple-dashboard.filterpanel.xml` filter panel description file.
TODO more detail
For more information on the portlet definition file, see the JBoss Portal server documentation or the JSR168 spec.

```
<portlet-app>
   ...
   <portlet>
      <portlet-name>SimpleFilterPanelPortlet</portlet-name>
      <portlet-class>
         org.pentaho.ui.portlet.FilterPanelPortlet
      </portlet-class>
      <supports>
         <mime-type>text/html</mime-type>
         <portlet-mode>VIEW</portlet-mode>
      </supports>
      <!-- package name containing the portlet.properties file, this file contains localized
strings -->
      <resource-bundle>
         org.pentaho.locale.portlet
      </resource-bundle>
      <portlet-info>
         <!-- key into portlet.properties file containing the localized title of the portlet
            displayed in the UI -->
         <title>SimpleFilterPanelPortlet</title>
      </portlet-info>
      <portlet-preferences>
         <!-- filter panel description file -->
         <preference>
            <name>filters</name>
            <value>samples/simpleDashboard/simple-dashboard.filterpanel.xml</value>
         </preference>
      </portlet-preferences>
   </portlet>
   ...
</portlet-app>
```

This XML fragment describes a portlet that is implemented by the `ActionPortlet` class, and whose visual display is created by the `JFree_ChartComponent.xaction` Action Sequence.

```
<portlet-app>
   ...
   <portlet>
      <portlet-name>SimplePieChartPortlet</portlet-name>
      <portlet-class>
         org.pentaho.ui.portlet.ActionPortlet
      </portlet-class>
      <supports>
         <mime-type>text/html</mime-type>
         <portlet-mode>VIEW</portlet-mode>
      </supports>
      <resource-bundle>
         org.pentaho.locale.portlet
      </resource-bundle>
      <portlet-info>
         <title>SimplePieChartPortlet</title>
      </portlet-info>
      <portlet-preferences>
         <preference>
            <name>action</name>
            <value>samples/simpleDashboard/JFree_ChartComponent.xaction</value>
         </preference>
      </portlet-preferences>
   </portlet>
   ...
</portlet-app>
```

Modify the `portlet-instances.xml` file to define the portlet instance.

The `portlet-instances.xml` file describes the mapping between the portlet name (in the `portlet.xml`) file and a portlet instance name. You can have multiple instances of a portlet in the portal, where each instance has its own preferences. This example simply "inherits" its preferences from the portlet definition. The instance name is used to add portlet instances to portal pages.

For more information on the portlet instance definition file, see the JBoss Portal server documentation.

```
<deployments>
   ...
   <deployment>
      <if-exists>overwrite</if-exists>
      <instance>
         <instance-id>SimpleFilterPanelPortletInstance</instance-id>
         <portlet-ref>SimpleFilterPanelPortlet</portlet-ref>
      </instance>
   </deployment>
   <deployment>
      <if-exists>overwrite</if-exists>
      <instance>
         <instance-id>SimplePieChartPortletInstance</instance-id>
         <portlet-ref>SimplePieChartPortlet</portlet-ref>
      </instance>
   </deployment>
   ...
</deployments>
```

Modify the `pentaho-portal-object.xml` file to place the portlet instances in a page in the portal.

Add a new page to the portal by adding a `<deployment>` element to the `<deployements>` root element, and then add the `<page>` element inside the `<deployment>` element.
Add a window for each portlet by adding a corresponding `<window>` element to the `<page>` element.
Add a portlet to each window by adding the `<instance-ref>` element to the `<window>` element, and specifying the name of the portlet instance (as defined in the portlet-instances.xml file) as the element's text.
(TODO give more detail on the elements)
For more information on the portal object definition file, see the in line comments, and the JBoss Portal server documentation.

```
<deployments>
   ...
   <deployment>
      <parent-ref>pentaho</parent-ref>
      <if-exists>overwrite</if-exists>
      <page>
         <page-name>Simple Dashboard</page-name>
         <properties>
            <property>
               <name>order</name>
               <value>90</value>
            </property>
            <!-- references a localized string in the portlet.properties file -->
            <property>
               <name>resourceKey</name>
               <!-- the name of the key in the properties file -->
               <value>SimpleDashboard</value>
            </property>
         </properties>
         <window>
            <window-name>PentahoPortalNavigationWindow</window-name>
            <instance-ref>
               PentahoPortalNavigationInstance
            </instance-ref>
            <region>navigation</region>
            <height>0</height>
            <!-- keep portal and page properties for this window -->
            <properties>
               <!-- use the window renderer from the emptyRenderer renderSet -->
               <property>
                  <name>theme.windowRendererId</name>
```

```
                  <value>emptyRenderer</value>
               </property>
               <!-- use the decoration renderer from the emptyRenderer renderSet -->
               <property>
                  <name>theme.decorationRendererId</name>
                  <value>emptyRenderer</value>
               </property>
               <!-- use the portlet renderer from the emptyRenderer renderSet -->
               <property>
                  <name>theme.portletRendererId</name>
                  <value>emptyRenderer</value>
               </property>
            </properties>
         </window>
         <!-- Specification of window containing the filter panel portlet -->
         <window>
            <window-name>SimpleFilterPanelPortletWindow</window-name>
            <instance-ref>SimpleFilterPanelPortletInstance</instance-ref>
            <region>left</region>
            <height>0</height>
         </window>
         <!-- Specification of a window containing the ViewAction/pie chart portlet -->
         <window>
            <window-name>SimplePieChartPortletWindow</window-name>
            <instance-ref>SimplePieChartPortletInstance</instance-ref>
            <region>left</region>
            <height>1</height>
         </window>
         <!-- insert Secure security-constraints -->
         <security-constraint>
            <policy-permission>
               <role-name>User</role-name>
               <action-name>view</action-name>
            </policy-permission>
         </security-constraint>
      </page>
   </deployment>
   ...
</deployments>
```

Add Localized Strings for Portlet and Portal Page Titles to the `portlet.properties` File

The comments in the previous XML fragments identify text-strings that are keys that reference text-strings in localized properties files (a.k.a. resource bundles). You'll recall that the portlet.xml file identifies the package (`org.pentaho.locale.portlet`) containing these properties files. You will want to add the key/text-string mappings for your keys in the appropriate localized file.

```
...
SimpleDashboard=SimPAL DashBORED
...
SimpleFilterPanelPortlet.javax.portlet.title=Simple Filter Panel Portlet
SimplePieChartPortlet.javax.portlet.title=Simple Pie Chart Portlet
...
```

Create a Filter Panel Definition File in the Repository

In this example, the filter panel is specified in `samples/simpleDashboard/simple-dashboard.filterpanel.xml`

```
<filters>
```

```
    <name>CustomerNamesFilter</name>
    <filter>
       <!-- Title text displayed in the filter panel -->
       <title>Customer</title>
       <!-- name and id of the HTML form control representing this filter -->
       <name>customernumber</name>
       <!-- type of control to display in the filter panel -->
       <type>list</type>
       <data-solution>samples</data-solution>
       <data-path>simpleDashboard</data-path>
       <data-action>getCustomerNames.xaction</data-action>
       <!-- name of the action sequence's output parameter containing the LOV -->
       <data-output>customerNamesList</data-output>
       <!-- maps to the <option> element's text -->
       <data-display>customername</data-display>
       <!-- maps to the <option> element's value attribute in the HTML form -->
       <data-value>customernumber</data-value>
    </filter>
 </filters>
```

Each filter panel definition file defines the filters for a filter panel. Each filter corresponds to a control in the user interface (UI) (e.g. comboboxes, lists, radio buttons, etc.) These controls are used to "filter" the data returned from the platform.
A filter definition is created by adding a `<filter>` element to the root node (`i.e. <filters>`) of the filter panel definition file, and then adding the appropriate child elements to the `<filter>` element.
There are 4 different filter types that can be specified in a filter panel. Each type uses a different mechanism for getting its LOV. The mechanism to get the LOV is specified by adding one or more distinct child elements to the `<filter>` element.
In addition to the set of distinct child elements of a `<filter>` element, there is a set of elements that all `<filter>` elements have in common.

List of Values (LOV)

A List of Values (LOV) is a term to describe a list of items, where each item has a name and a value. An example might be the months of the year, where the first item's name is Jan., and its value is 0, the second item's name is Feb., and its value is 1, and so on.
In a user interface, when a user needs to choose one of many items, it is common to show the user the name of each item in the LOV, and when the user makes a selection, pass the selected item's value to the server.

The Common Elements

(* required elements)

- `<title>` The text of the title element will be displayed as a label for the filter in HTML form.
- `<name>` * Is used to set the `name` and `id` attributes of the HTML control in the form. For instance, if this filter is going to generate a list (a.k.a. in HTML, a `<select>` element), the `<select>` element's `id` and `name` attributes will be set to the text of the `<name>` element.
- `<type>` The type of HTML control to place in the filter panel. If this element is missing, or its value is not recognized, a <select> control (a.k.a. combo box) is generated. Recognized values are:
  - `radio`: creates a set of radio buttons
  - `list`: creates a list box
  - `list-multi`: creates a select list with multi-select capability
  - `check-multi`: creates a set of check boxes, all of them visible
  - `check-multi-scroll`: creates set of check boxes in a scrollable panel

    - check-multi-scroll-2-column: creates 2 columns of check boxes in a scrollable panel
    - check-multi-scroll-3-column: creates 3 columns of check boxes in a scrollable panel
    - check-multi-scroll-4-column: creates 4 columns of check boxes in a scrollable panel
- <data-display> * name of the column in the LOV that is used to provide the UI display text. For instance if the type of the filter panel is list, the values in this column will be used to provide the text in the list box.
- <data-value> * name of the column in the LOV that is used to provide values that are associated with the UI display text. These are the values that will be sent when the form is submitted to the server.

Elements Unique to Each Filter Type

Each filter in the filter panel can get its list-of-values (LOV) from one of four sources:

1. session scoped parameter
2. global scoped parameter
3. action sequence output
4. static list defined in the body of the <filter> element

Session Scoped Parameter

To configure the filter panel to get its LOV from a session scoped parameter, add the <session-attribute> as a child of the <filter> element. The text of the element is the name of a session scoped parameter which contains the LOV. This parameter must be of one of the types: list, result-set, (TODO what are the others?).

Global Scoped Parameter

To configure the filter panel to get its LOV from a global scoped parameter, add the <global-attribute> as a child of the <filter> element. The text of the element is the name of a global scoped parameter which contains the LOV. This parameter must be of one of the types: list, result-set, (TODO what are the others?).

Action Sequence Output

To configure the filter panel to get its LOV from an Action Sequence, add the elements <data-solution>, <data-path>, <data-action> and <data-output> as children of the <filter> element. The text of each element should be:

- <data-solution>: The name of the solution in the solution-repository containing the Action Sequence defintion file. For example, in the PCI the solution-repository's directory is called pentaho-solutions. The solution itself is contained in the solution-repository directory. For instance, the PCI comes with solutions called: samples, admin, and test solutions.
- <data-path>: The path from the solution to the Action Sequence definition file.
- <data-action>: The name of the Action Sequence definition file.
- <data-output>: If this element is present, its text should be the name of the of the Action Sequence output parameter that the Action Sequence will place the LOV in. If this element is missing, the filter panel will look for the LOV in an output parameter of the Action Sequence called

`resultset`.

Static List

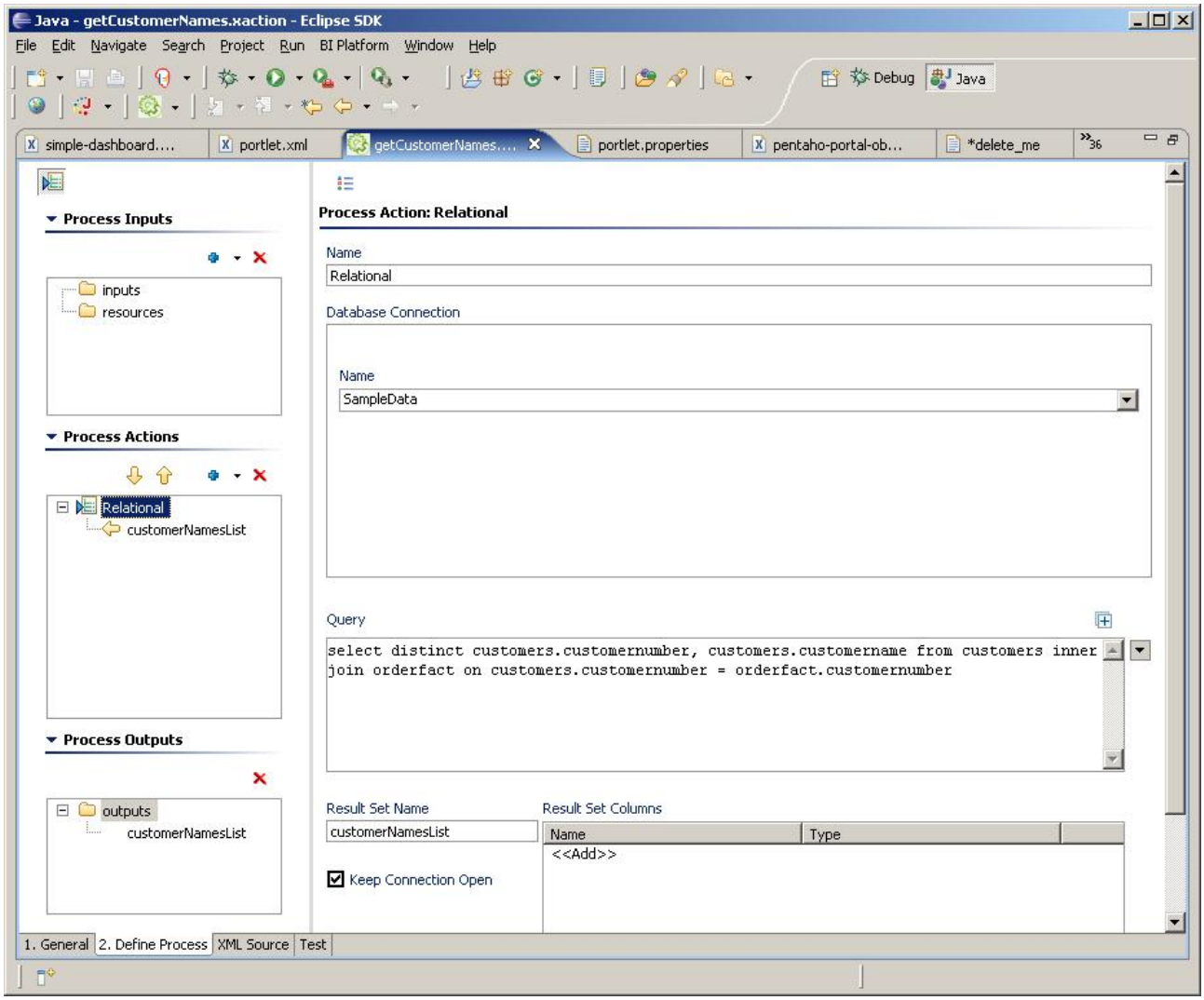
To configure the filter panel to get its LOV from a static list defined in the filter panel definition file

Static LOV is not yet available

Initializing a LOV with a static list may be available in a future release.

Create Action Sequences to Provide list-of-values (LOV) Data for the Filter Panel's Controls

In this example, I created the action sequence in `samples/simpleDashboard/getCustomerNames.xaction`. This Action Sequence contains a single Action that runs a SQL query to get customer names and customer numbers of customers who have made at least one order. Notice that the column names (`customername, customernumber`) map to the `<data-display>` and `<data-value>` elements in the filter panel definition file. Notice also that the Result Set Name (`customerNamesList`) in the action sequence maps to the `<data-output>` element.
The values in the Result Set will be displayed in the list control in the filter panel's form.



Create Action Sequences that are Associated with the other Portlets in the Page

In this simple example, there is only one other portlet in the page, the `SimplePieChartPortlet` portlet (implemented using the `ActionPortlet` class).
You'll recall the `SimplePieChartPortlet`'s preference specified that it should run the `samples/simpleDashboard/JFree_ChartComponent.xaction` Action Sequence. This Action Sequence responds to changes in parameters placed in the session by the filter panel, and provides the view of the business data.

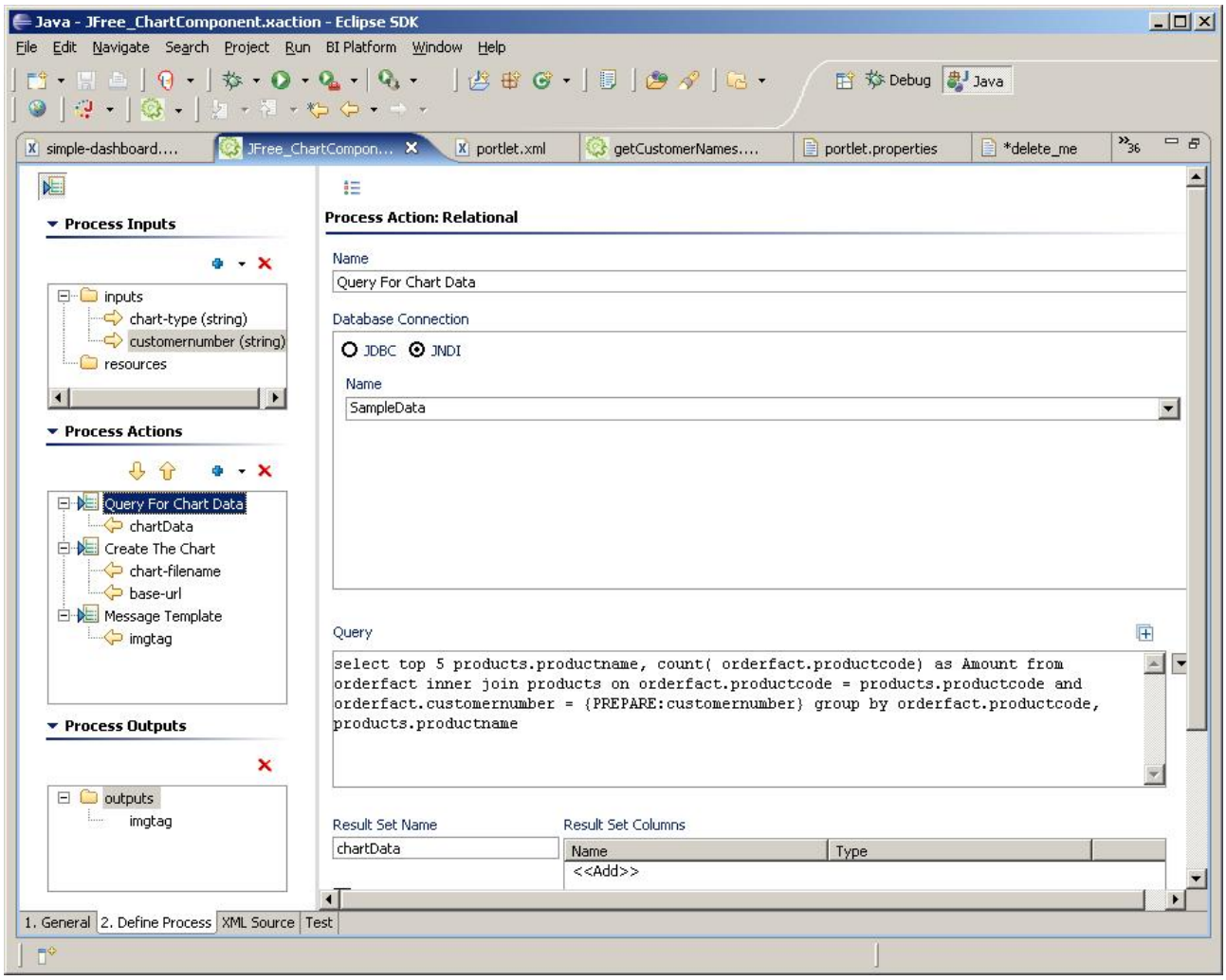
This Action Sequence contains:

1. A `Relational` action that gets that data that will be displayed in the pie chart
2. A `Pie Chart` action that uses the data to create a pie chart image on the server
3. A `Message Template` that creates an HTML fragment to pull the pie chart image into the portlet

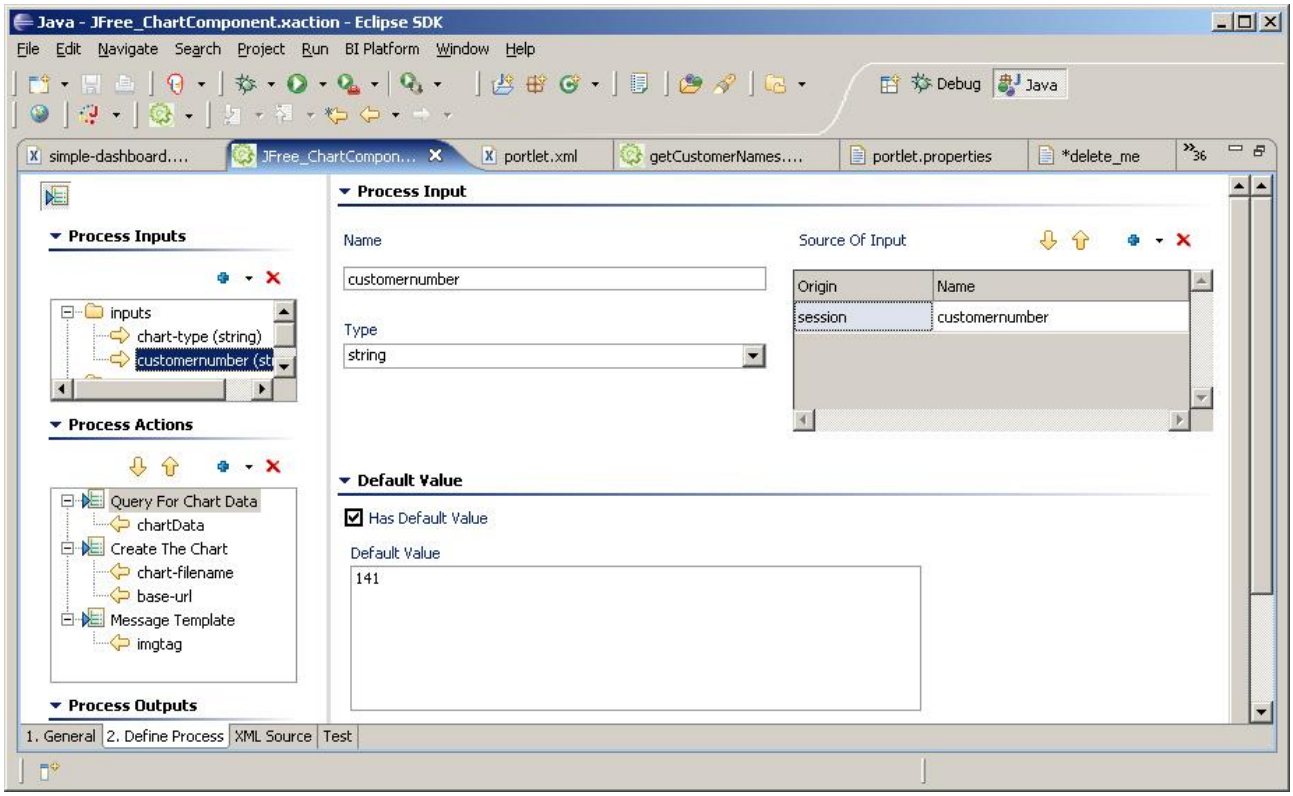
The Relational action runs a query against the SimpleData database to identify the top 5 most purchased products for a particular customer. In this example, I am using a pie chart. Typically for a pie chart you'll

want two columns of data. The first column identifies the label for the pie slice, the second column identifies the relative quantity of the pie slice. The number of rows will determine the number of slices.

In this example, the first column contains the product purchased, the second column contains the number of units purchased.



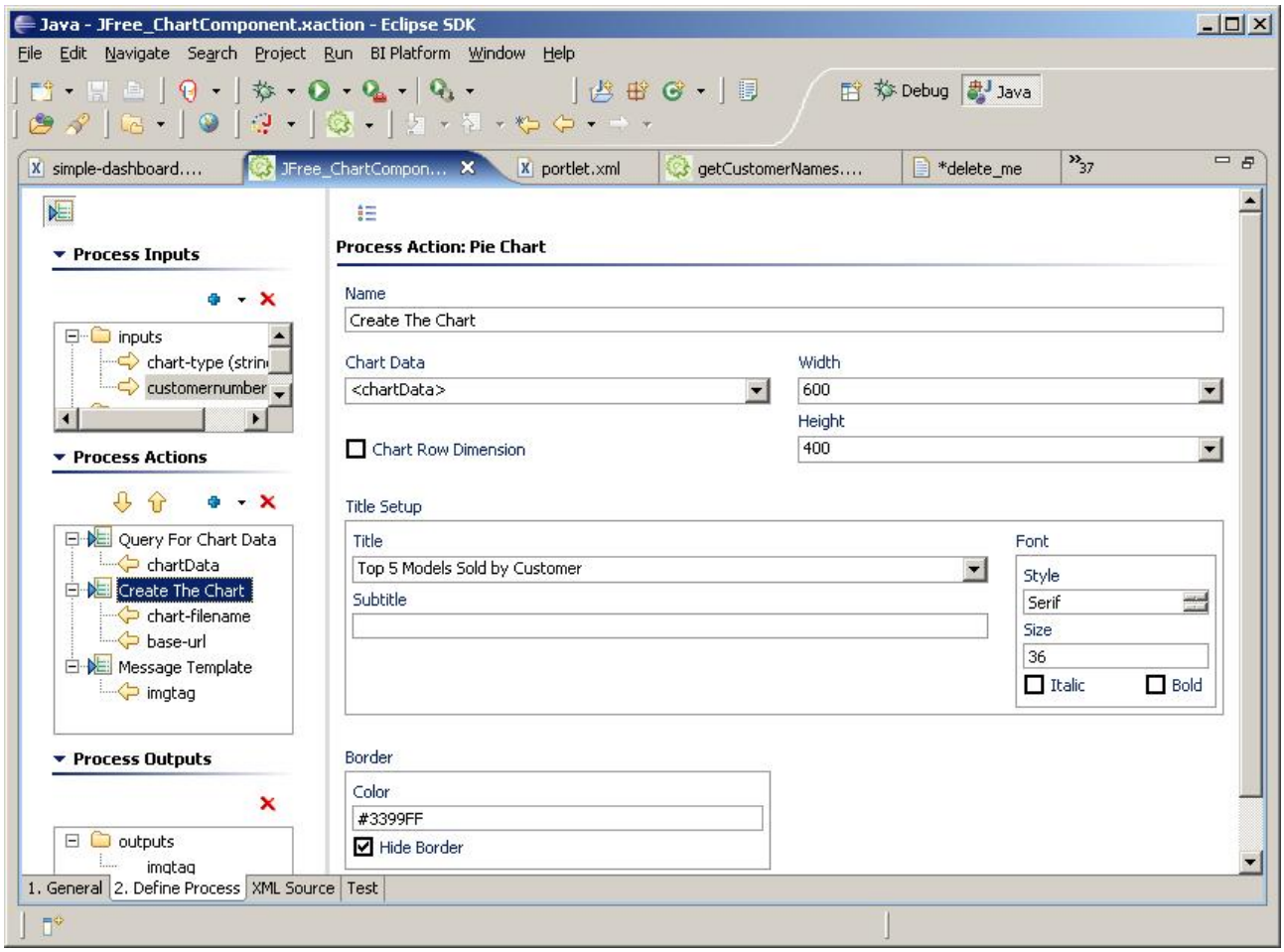
Note that the `customernumber` in the query comes from the session scoped input `customernumber`. This value is placed in the session by the filter panel.



The `Pie Chart` action generates a `.png` file on the server containing the pie chart image. Notice the `chart-filename` and `base-url` outputs. These outputs are created automatically by the `Pie Chart` action. `chart-filename` contains the name of the file containing the pie chart image. `base-url` contains the portal relative URL to the pie chart image file. These will be consumed by the `Message Template` action to create the HTML `<img>` tag that references the chart image on the server.

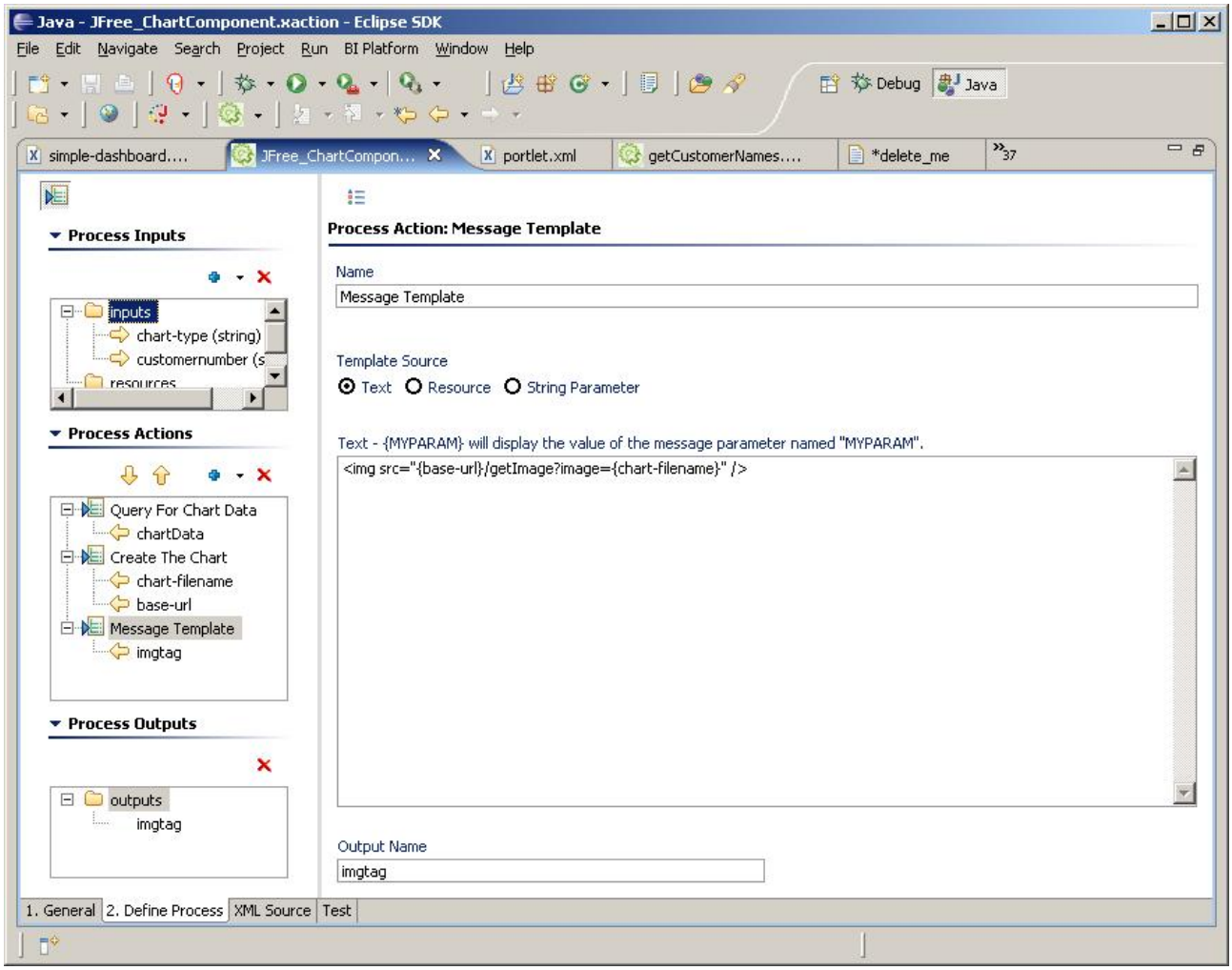
Default Behavior for the Pie Chart Won't Work in Portlet

The default behavior of the `Pie Chart` action is to generate a complete HTML page containing the chart image. This is appropriate for a JSP or a Servlet. However the HTML in a portlet consists of an HTML fragment, not a full HTML page.
The default behavior can be changed by specifying outputs for the `Pie Chart` action. Fortunately the `Pie Chart` action provides the `chart-filename` and `base-url` outputs. By specifying these outputs for the `Pie Chart` action, they will be available to any following actions in the Action Sequence.



The `Message Template` action creates an HTML `<img>` element that references the chart's image file on the server. The `Message Template` is able to construct an HTML `<img>` element with a valid `src` attribute by combining the `base-url` and `chart-filename` inputs from the `Pie Chart` action to reference the generated image file on the server. This action's output (`imgtag`) is an HTML fragment which is added to the Action Sequence's outputs.

The Action Sequence's output (i.e. the HTML fragment) is captured by the ActionPortlet. The ActionPortlet passes this HTML fragment on to the Portal to be aggregated with the HTML fragments generated by the other portlets in the page. The aggregated HTML content is then sent to the browser to render the portal dashboard.



I have included the XML for the `JFree_ChartComponent` Action Sequence for your convenience.

```
<?xml version="1.0" encoding="UTF-8"?>
<action-sequence>
  <name>JFree_ChartComponent.xaction</name>
  <title>Simple Chart</title>
  <version>1</version>
  <logging-level>DEBUG</logging-level>
  <documentation>
    <author>Steven Barkdull</author>
    <description>Creates a chart image based on a SQL query</description>
    <icon/>
    <help>You R going to need it.</help>
    <result-type>rule</result-type>
  </documentation>

  <inputs>
    <chart-type type="string">
      <default-value>.png</default-value>
      <sources>
        <request>type</request>
      </sources>
    </chart-type>
    <customernumber type="string">
      <sources>
        <session>customernumber</session>
      </sources>
      <default-value><![CDATA[141]]></default-value>
    </customernumber>
  </inputs>
```

```
<outputs>
  <imgtag type="string">
    <destinations>
      <response>content</response>
    </destinations>
  </imgtag>
</outputs>

<resources/>

<actions>
  <action-definition>
    <component-name>SQLLookupRule</component-name>
    <action-type>Query For Chart Data</action-type>
    <action-inputs>
      <customernumber type="string"/>
    </action-inputs>
    <action-outputs>
      <query-result type="result-set" mapping="chartData"/>
    </action-outputs>
    <component-definition>
      <source>sql</source>
      <live>true</live>
      <jndi>SampleData</jndi>
      <query><![CDATA[
         select top 5 products.productname, count( orderfact.productcode) as Amount
         from orderfact inner join products on orderfact.productcode = products.productcode
         and orderfact.customernumber = {PREPARE:customernumber}
         group by orderfact.productcode, products.productname]]></query>
    </component-definition>
  </action-definition>

  <action-definition>
    <component-name>ChartComponent</component-name>
    <action-type>Create The Chart</action-type>
    <action-inputs>
      <output-type type="string" mapping="chart-type"/>
      <chart-data type="result-set" mapping="chartData"/>
    </action-inputs>
    <action-outputs>
      <chart-filename type="string"/>
      <base-url type="string"/>
    </action-outputs>
    <component-definition>
      <by-row>false</by-row>
      <height>400</height>
      <width>600</width>
      <title>Top 5 Models Sold by Customer</title>
      <chart-attributes>
        <!--  this is the background for the whole image -->
        <!--  TODO support gradient and texture painting -->
        <chart-type>PieChart</chart-type>
        <title-position>TOP</title-position>
        <height/>
        <width/>
        <title/>
        <title-font>
          <font-family>Serif</font-family>
          <size>36</size>
          <is-bold>false</is-bold>
          <is-italic>false</is-italic>
        </title-font>
        <range-title>US Dollars</range-title>
        <chart-background type="color">#FFFFFF</chart-background>
        <plot-background-color>#FF0000</plot-background-color>
        <orientation>Horizontal</orientation>
        <is-3D>false</is-3D>
        <is-stacked>false</is-stacked>
        <category-label-rotation>0</category-label-rotation>
        <border-visible>false</border-visible>
        <border-paint>#3399FF</border-paint>
        <include-legend>true</include-legend>
      </chart-attributes>
    </component-definition>
  </action-definition>

  <action-definition>
```

```
        <component-name>TemplateComponent</component-name>
        <action-type>Message Template</action-type>
        <action-inputs>
          <base-url type="string"/>
          <chart-filename type="string"/>
        </action-inputs>
        <action-outputs>
          <output-message type="string" mapping="imgtag"/>
        </action-outputs>
        <component-definition>
          <template><![CDATA[<img src="{base-url}/getImage?image={chart-filename}"
/>]]></template>
        </component-definition>
      </action-definition>

    </actions>
  </action-sequence>
```

Create a JNDI Accessible Datasource to Provide Data to the Action Sequences.

In the PCI, datasources are specified by adding an XML file to the `server/default/deploy` folder of the PCI. The file's name must end with "-ds.xml".
This example uses the SampleData JNDI name to reference its datasource. SampleData's configuration file is called `sampledata-ds.xml`, and looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>SampleData</jndi-name>
    <connection-url>jdbc:hsqldb:hsql://localhost:9001/sampledata</connection-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>pentaho_user</user-name>
    <password>password</password>
    <check-valid-connection-sql>
      select count(*) from INFORMATION_SCHEMA.SYSTEM_SEQUENCES
    </check-valid-connection-sql>
  </local-tx-datasource>
</datasources>
```

Configuring JNDI Datasources for use in the Pentaho BI Platform has more information on creating JNDI datasources in the PCI.

Using a System Action to Improve the Performance of the Filter Panel

The filter panel's current configuration will retrieve the list-of-values (LOV) from the database every time the filter panel is rendered in the portal page. For LOV that rarely change, fetching the data with this frequency incurs unnecessary overhead on the database server.
We can eliminate this overhead by:

1. Configuring a System Action to retrieve the list of values from the database one time when the platform starts up, and place it in a global scoped parameter. This global scoped parameter is then available to any Action Sequence as an input.
2. Modifing the filter panel's definition to retrieve the LOV from the global scoped parameter, instead of running the getCustomerNames Action Sequence to get access to the LOV.

3. Modify the `getCustomerNames` Action Sequence to insure that it closes its database connection, and stores the result of the query in a disconnected result set.

Configuring the System Action

Briefly, a System Action is an Action Sequence that is configured to run either at system start time, or session start time. System Actions are configured in the repository's system configuration file, usually called `pentaho.xml`. You can learn the details of configuring System Actions in the article on System Actions.
Adding the following xml fragment to the system configuration file will cause the getCustomerNames.xaction to be run when the system starts up, and the output of that action sequence will be available to portlets as a global scoped parameter.

```
<pentaho-system>
   ...
   <system-actions>
      ...
      <org.pentaho.ui.portlet.PentahoPortletSession scope="global">
         samples/simpleDashboardUsingSystemAction/getCustomerNames.xaction
      </org.pentaho.ui.portlet.PentahoPortletSession>
      ...
   </system-actions>
   ...
</pentaho-system>
```

Modifing the Filter Panel

Modify the filter panel's definition by removing the `<data-solution>`, `<data-path>`, `<data-action>`, and `<data-output>` nodes, and replacing them with the `<global-attribute>` node. The text of the `<global-attribute>` node should be the same as the text of the `<data-output>` node in the original filter panel configuration file (i.e. the name of the output of the getCustomerNames Action Sequence).

```
<filters>
   <name>CustomerNamesFilter</name>
   <filter>
      <!-- Title text displayed in the filter panel -->
      <title>Customer</title>
      <!-- name and id of the HTML form control representing this filter -->
      <name>customernumber</name>
      <!-- type of control to display in the filter panel -->
      <type>list</type>
      <!-- name of global scoped parameter containing the LOV -->
      <global-attribute>customerNamesList</global-attribute>
      <!-- maps to the <option> element's text -->
      <data-display>customername</data-display>
      <!-- maps to the <option> element's value attribute in the HTML form -->
      <data-value>customernumber</data-value>
   </filter>
</filters>
```

Configuring the Filter Panel in Pre-1.6 Versions of the Platform

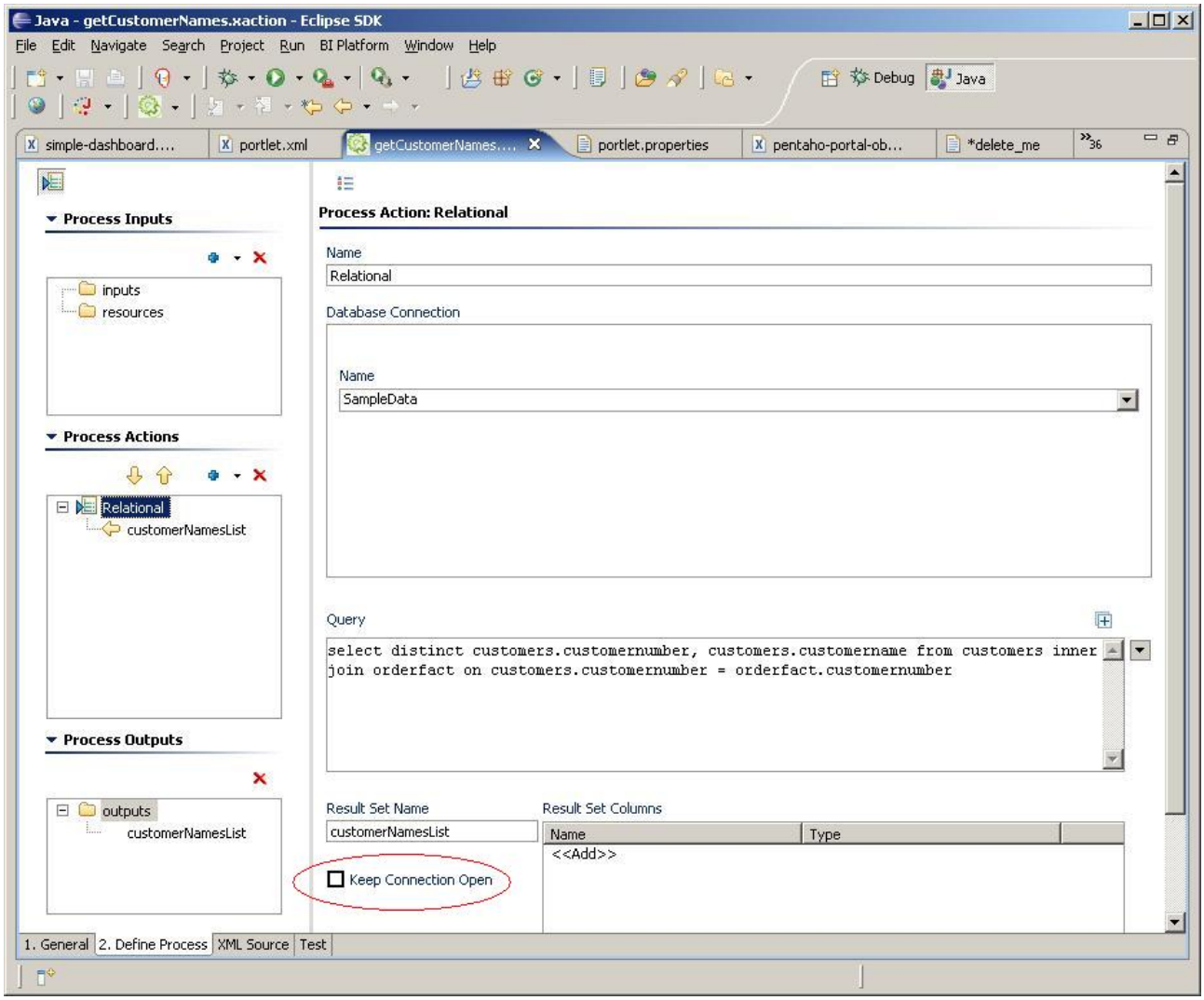
On pre-1.6 versions of the platform, instead of using using the `<global-attribute>` element to identify the name of the input, use the `<session-attribute>` element.

Modify the `getCustomerNames` Action Sequence to Insure that it Closes its Database Connection

By default in a `Relational` action the `Keep Connection Open` check box is checked. You will need to uncheck this checkbox.

What does "Keep Connections Open" mean?

If Keep Connections Open is not checked, when a Relational action runs a database query, it immediately stores the results of the query into a light-weight java object, and closes the connection to the database. This light-weight java object can be stored as a parameter in request, session or global scope. This makes the object available to other actions in this action sequence, and to other action sequences executing at another time. Often however, the size of a result is very large, making it expensive to copy the contents into a light-weight java object. It is also expensive to have both objects temporarily in memory at the same time. In this case, it makes sense to have "Keep Connections Open" checked. This will cause the result-set's connection to the database to be kept open. The result-set will be available to other actions in this action sequence. And a copy will NOT be made to a light-weight java object. In all cases, the connection to the database is closed at the conclusion of the action sequence. This suggests that a result set that was created with "Keep Connections Open" checked should never be placed in session or global scope, since the result set's connection will be closed at conclusion of the current action sequence, making it unusable.



You will need to restart your server for these changes to take effect.

Test the Portal-based Dashboard

Example Files

portlet.xml
portlet-instances.xml
pentaho-portal-object.xml
simple-dashboard.filterpanel.xml
JFree_ChartComponent.xaction
getCustomerNames.xaction
alternate_simple-dashboard.filterpanel.xml
pentaho.xml