WebSphere. Monitor

Version 6.0.2

Extending the WebSphere Business Monitor Dashboard

Contents

Figures

1 Introduction

1.1 The goal of this article

The goal of this article is to provide a best-practices design for a Java™ Specification Request (JSR) 168 portlet using Alphablox to extend the IBM WebSphere® Business Monitor v6.0.2 Dashboard. The article will provide a working portlet example, that a developer can use as a design pattern for developing a customized WebSphere Business Monitor Dashboard portlet.

1.2 The audience of the article

The audience for this article is any portlet developer wanting to extend the functionality of the WebSphere Business Monitor Dashboard by implementing his own portlet that uses data collected by Monitor and stored in its Datamart.

It is assumed that the reader is familiar with Portal and Alphablox components and administration, as well as the JSR 168 application programming interface (API). However, a brief introduction to the portlet JSR 168 API, AlphaBlox and IBM CubeViews™ is provided.

This article also requires the reader to have basic knowledge of WebSphere Portal Server and Monitor usage.

1.3 Disclaimers

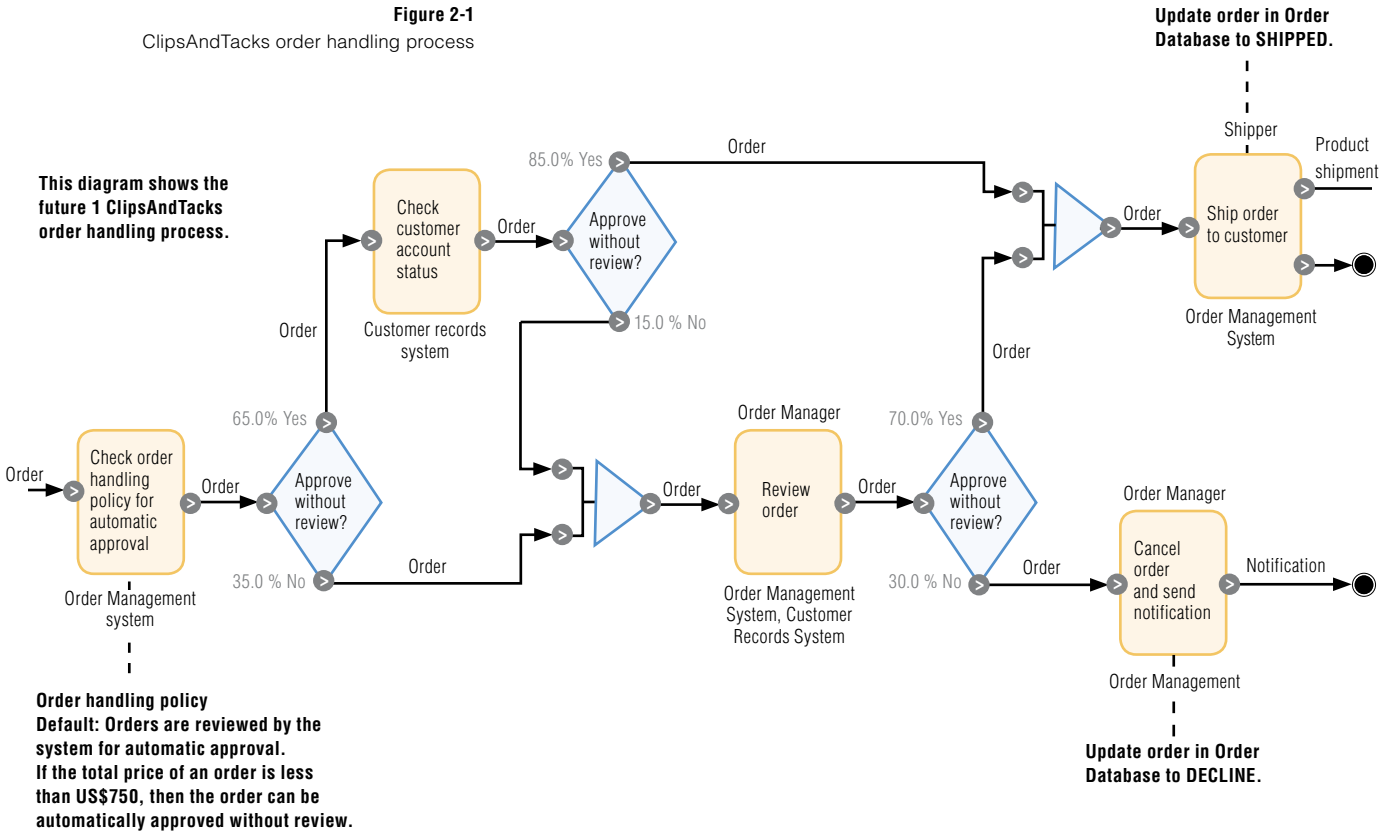
- *The sample portlet can only access the Datamart database using Alphablox APIs to retrieve cube metadata. Neither access to the content stored in the other Monitor database nor direct database access to the Datamart through raw SQL statement is supported.*
- *The sample portlet does not go into detail about error handling and user-preference validation.*

2 Sample scenario

2.1 Sample scenario—ClipsAndTacks

This document chooses the ClipsAndTacks scenario as the process scenario which will be monitored by Monitor and its instance data will be accessed by the sample portlet that is described in later chapters. Below diagram is the process diagram of the ClipsAndTacks scenario.

Figure 2-1
ClipsAndTacks order handling process



Flows of the ClipsAndTacks scenario process:

I. The customer visits the ClipsAndTacks Web site:
Enter the account number or create an account.
Enter the order information.
Submit the order.

II. Approve order or send for review, based on the following business rules:
If order is less than or equal to US$750, approve automatically
If order is over US$750, send for review

III. If the order is approved:
Check the customer's account status.

If the account is in good standing:
Send the order to the warehouse.
Issue a packing slip.
Record the order in the order records database.
Ship the product.

If the account is not in good standing:

IV. Or if the order is not approved:
Review the order manually.
Determine if the order is an acceptable credit risk.

If the order is an acceptable credit risk:
Send the order to the warehouse.
Issue a packing slip.
Record the order in the order records database.
Ship the product.

If the order is not an acceptable credit risk:
Cancel the order.
Send an e-mail notification to the customer.

A full explanation of the ClipsAndTacks scenario can be found in "Chapter 3. Case Study: ClipsAndTacks" in the Redbook, *Business Process Management: Modeling through Monitoring Using WebSphere V6 Products* (w3.itso.ibm.com/abstracts/sg247148.html?Open). This Redbook also describe the steps to install this process on a WebSphere Process Server runtime server.

2.2 Monitoring model

A monitoring model named OrderHandlingFuture1.mm is provided in the source code .zip file that accompanies this document.attached, It is built by the Monitoring Model Editor that is a plug-in to WebSphere Integration Developer. Please refer to the WebSphere Business Monitor product documentation (www-306.ibm.com/software/integration/wbimonitor/library/documentation.html) for how to build and deploy a monitoring model.

2.3 Test environment assumptions

This document assumes the following environments are prepared for testing the sample portlet:

1. *ClipsAndTacks process has been deployed on the WebSphere Process Server runtime server.*
2. *The monitoring model for the ClipsAndTacks process has been deployed on the Monitor server.*
3. *The Common Event Infrastructure (CEI) between the WebSphere Process Server runtime and the Monitor server has been configured successfully.*
4. *Some process instances have been completed and the instance data has been replicated to the Datamart database.*

2.4 Sample portlet functionality introduction

There are two reasons that users might want to write a custom portlet instead of just using the Monitor dashboard portlets:

- *They might want to implement a custom view to display and analyze data.*
- *They might want to access the Monitor database from their own application.*

The sample portlet described in this document is not going to extend a specific functionality that the production portlets do not include. Instead, the sample portlets will display the following benefits to any user who wants to write a custom portlet:

- *How to implement a portlet application with JSR 168 API.*
- *How to integrate and benefit from the Alphablox components in the portlet application.*
- *How to access and manipulate the history data that is collected by Monitor.*

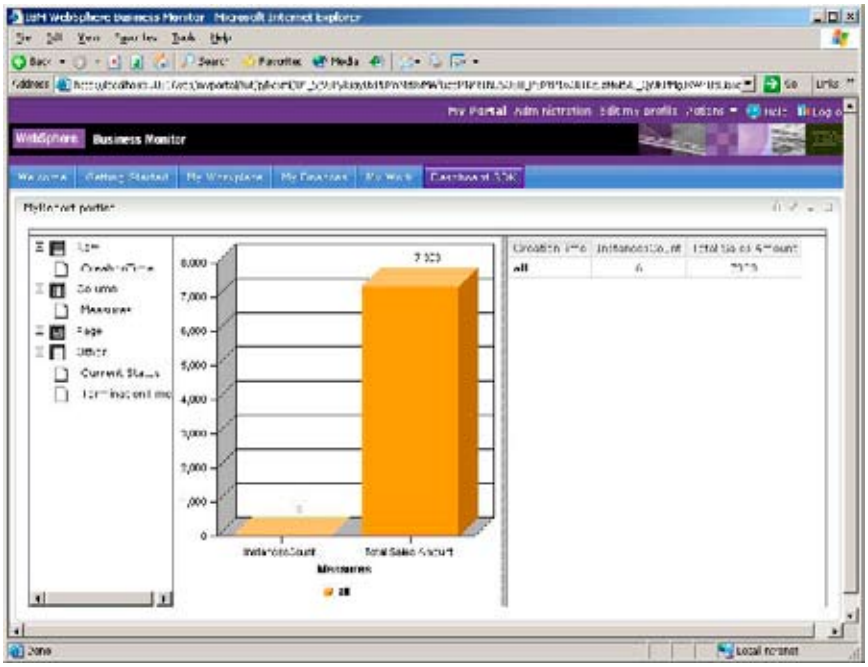
The sample portlet will use an Alphablox PresentBlox to perform the dynamic analysis on the history data of the Clips and Tacks process. This history data is organized as 3-types of forms: cube, measure and dimension. These cube, measure and dimension forms are defined in the monitoring model for the Clips and Tacks process. With the Alphablox API, the cube, measure and dimension data can be retrieved and presented in the PresentBlox component. Further analysis functionalities are then provided by the PresentBlox.

The sample portlet implements two portlet modes: the view mode and the configuration mode. The configuration mode shows all available cubes, measures and dimensions in the database. Users can select specific cubes, measures and dimensions that will be displayed and manipulated in the view mode. The view mode is responsible for showing the user selected data in a PresentBlox. Additional analysis functionalities are provided by the PresentBlox itself.

2.4.1 Portlet view mode

The following screen capture shows the view page of the sample portlet.

Figure 2-2
Portlet view mode



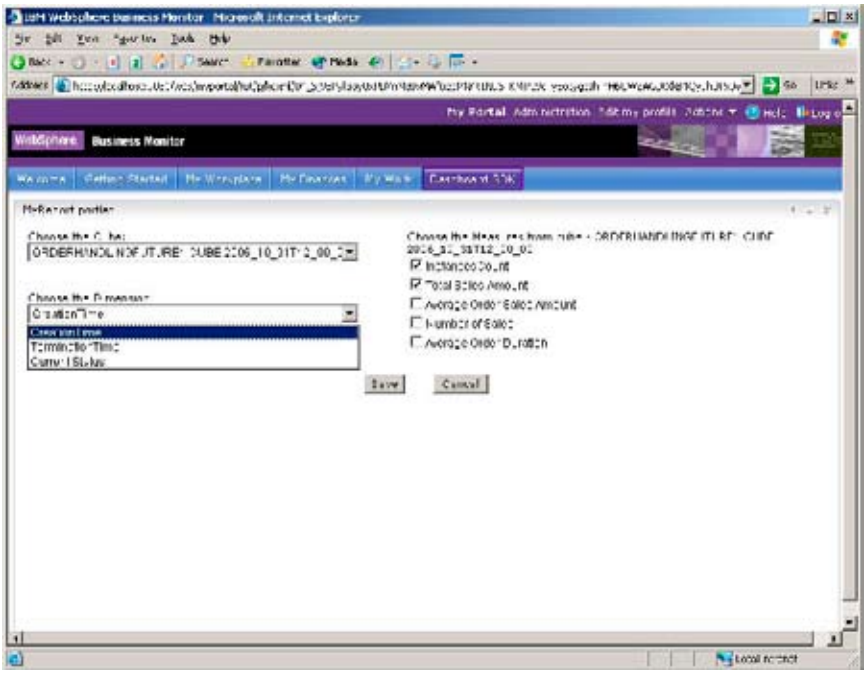
Notes:

- The portlet consists of an Alphablox component called *PresentBlox*.
- User-selected cubes, dimensions and measures display in the PresentBlox.
- Additional analysis functionality can be retrieved by right-clicking on any data item.
- A menu bar and toolbar will appear when the portlet is maximized, and disappear if the portlet is at normal size.

2.4.2 Portlet configuration mode

The following screen capture shows the configuration page of the sample portlet.

Figure 2-3
Portlet configuration mode



Notes:

- There are two drop-down lists. One is populated with all cubes in the database. Another one shows all the dimensions in a specified cube. The dimension list is determined by the selection of a cube.
- There is a list of check boxes showing all available measures, which are also determined by the selection of a cube.
- Changes on the cube list results in the dimensions and measures being displayed in this cube.
- After a user selects the preferred cube, dimension and measures, clicking **Save** will bring the portlet back to the view mode. And the PresentBlox in view mode is updated to reflect the new selections.
- Clicking **Cancel** just brings the portlet back to the view mode; no changes are updated in the PresentBlox.

3 Background knowledge

3.1 DB2 Alphablox introduction

3.1.1 Overview

IBM DB2® Alphablox provides the ability to rapidly create custom, Web-based applications that fit into the corporate infrastructure. Applications built with the DB2 Alphablox platform run in standard Web browsers, allowing real-time, highly customizable multidimensional analysis in a Web browser.

With the DB2 Alphablox platform you can:

- *Access and interact with data in multidimensional and relational databases*
- *Choose from a wide variety of charts to display data*
- *Interact with multidimensional data sources on different levels (for example, filter, drill down, and so on) to interactively display the exact view of the data preferred*
- *Access an intuitive user interface that helps make analysis of the data easy and powerful*
- *Access multiple data sources with a single application*

DB2 Alphablox provides a wide variety of APIs so developers can create custom applications. The DB2 Alphablox APIs are written in the Java™ programming language, and application developers can access them using Java that is issued on the server or through JavaScript that is interpreted in the browser.

The remainder of this chapter describes how DB2 Alphablox fits into a Java 2 Platform, Enterprise Edition (J2EE) environment, explains the components of DB2 Alphablox, and describes the architecture of DB2 Alphablox. For more information, refer to the DB2 Alphablox Information Center (publib.boulder.ibm.com/infocenter/ablxhelp/v8r4m0/index.jsp).

3.1.2 DB2 Alphablox components

The DB2 Alphablox provides an extensive library of modular, reusable components called Blox to help meet all analytic application design requirements for maximum usability. These components include the data access Blox, user interface Blox, form elements Blox, business logic Blox and analytic infrastructure Blox.

3.1.2.1 Data access Blox

Data access is managed through the Data Blox, the Stored Procedures Blox and the MDB Query Blox.

The Data Blox manages the connection between the user interface Blox and the appropriate data source. It also is responsible for submitting queries and retrieving result sets from a database. The syntax used for queries will vary

depending on the data source that is being accessed. DB2 Alphablox supports the following types of query strings:

- *Essbase report scripts for DB2 online analytical processing (OLAP) and Hyperion Essbase data sources*
- *Multidimensional Expressions (MDX) for Microsoft® SQL Server analysis services and DB2 Alphablox cubes (a multidimensional representation of relational data created with the DB2 Alphablox cube server)*
  Note: This type of query is the only one supported for WebSphere Business Monitor Dashboard extension.
- *SQL statements for relational data source*

The MDB Query Blox is useful for specifying data queries based on axes, dimensions and members without having to use data-specific query language. This Blox also makes it easy to replace query fragments using the program.

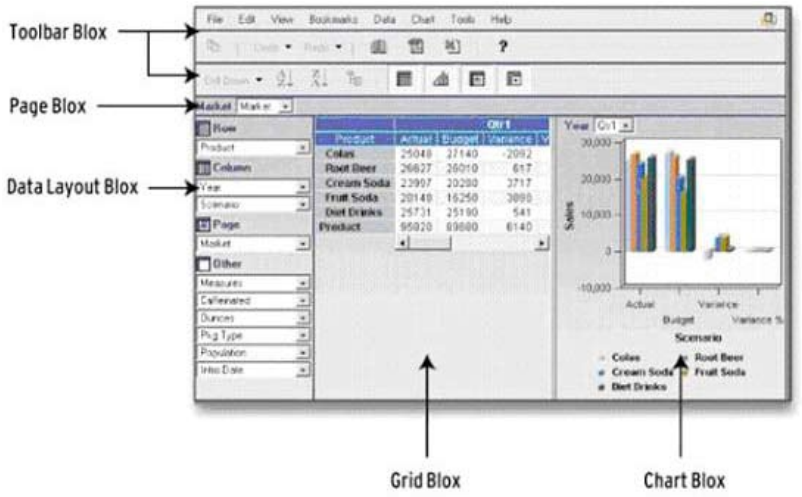
The data Blox and the MDB Query Blox are not used in this sample. For more information on Data access Blox, refer to the DB2 Alphablox Information Center (publib.boulder.ibm.com/infocenter/ablxhelp/v8r4m0/index.jsp).

3.1.2.2 User interface Blox

The user interface (UI) is a crucial factor in application usability. The user interface Blox provided by DB2 Alphablox is highly functional, interactive, and completely customizable. The user interface Blox in a DB2 Alphablox-enabled application includes several components:

- ***Grid Blox*** *provides a table of data and all the UI functions to manipulate the data in a multidimensional way. Users can drill up, drill down, sort, pivot, swap axes or choose to view the top n or bottom n members based on the values of a given data column.*
- ***Chart Blox****, which is used for advanced visualization of data, supports a large number of chart types including bar, line, pie, scatter and bubble charts, as well as bipolar and dual-axis charts.*
- ***Data Layout Blox*** *makes it easy for you to move and reorder dimensions across axes.*
- ***Page Blox*** *provides drop-down lists to manage the setting of slice dimensions*
- ***Toolbar Blox*** *offers easy access to common data-analysis functionality through the click of a button.*
- ***PresentBlox*** *combines the Grid Blox, the Chart Blox, the Data Layout Blox, the Toolbar Blox and the Page Blox into a single, well-orchestrated, interconnected user interface along with user toolbars and menus (see Figure 3-1). The PresentBlox also implements additional logic to interconnect the various underlying Blox. For example, in a PresentBlox, drilling down in the grid automatically updates both the grid and chart with the new data. These UI elements employ cutting-edge Dynamic HTML (DHTML) technology to provide a rich user experience including menu bars, right-click menus and custom layouts in a thin client (no need for Java, Microsoft ActiveX (or other browser plug-ins).*

Figure 3-1
Example of the PresentBlox

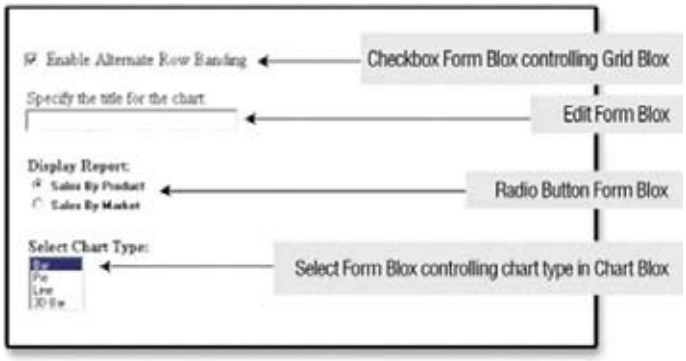


3.1.2.3 Form Blox

DB2 Alphablox provides several form Blox that can be very useful in developing custom analytic applications. Developers who are familiar with using native HTML form elements know that when the HTML page is refreshed, the user's current form element selections get reset unless they write additional code to maintain the same state across page refreshes.

All the Form Blox not only maintain the form elements' current state, freeing the developer from writing the extra code, but they also hook up with other components, such as JavaBeans, including the data access Blox and user interface Blox to provide the most commonly required functionality with minimal coding. For example, it is easy to link a Check box Form Blox to enable or disable the alternate row banding property in the Grid Blox and to create a Select Form Blox that controls the chart-type property in the Chart Blox.

Figure 3-2
Example of the Form Blox



The various form Blox provided by DB2 Alphablox include:

- Tree navigation element
  - Tree Form Blox
- For Basic HTML form elements
  - Checkbox Form Blox
  - Edit Form Blox
  - Radio Button Form Blox
  - Select Form Blox
- For MDB-specific HTML form elements
  - Data Source Select Form Blox
  - Cube Select Form Blox
  - Dimension Select Form Blox
  - Member Select Form Blox
  - Time Period Select Form Blox
  - Time Unit Select Form Blox

3.1.3 DB2 Alphablox programming model

With the DB2 Alphablox tag libraries, the JavaServer Pages (JSP) author does not have to know the low-level technical details behind the Blox components, but simply needs to know the syntax and functionality for each Blox. This process enables page authors with no Java experience to incorporate analytics seamlessly on an intranet or extranet using best-of-breed authoring tools.

Each Blox has a comprehensive set of properties that by using the tags could be set easily to custom values in the JSP pages. For example, setting the width property in the `<blox:present>` tag to 50 percent will make this PresentBlox occupy only half of the available horizontal space in the browser.

DB2 Alphablox also provides application developers with flexibility in customizing the user interface and adding their own business or application logic by exposing every Blox as a JavaBean and allowing programmatic access to those beans through a rich set of Java APIs. Moreover, DB2 Alphablox provides the Blox UI model – where application developers can register server-side handlers to handle user interaction events in the browser, such as mouse clicks and right-click actions. This Blox UI model offers developers the ability to customize the user interface behavior. For example, application developers can provide a different right-click menu depending on whether the user clicks on the row header, on the column header or on a data cell, and then apply any additional logic based on the current user's role, time of day or session state.

Application developers also can add their own HTML, CSS and JavaScript to the JSP pages to achieve the exact look and feel that they want. DB2 Alphablox also provides developers a unique and powerful ability to remotely invoke server-side Java APIs through JavaScript.

3.2 JSR168 Overview

JSR 168 is a specification from the Java Community Process for the standardization of portlets. The specification was developed to provide interoperability for running portlets on any vendor's implementation of the JSR 168 portlet container. For more information about the Java Portlet Specification, see the JSR 168 specification (developers.sun.com/prodtech/portalserver/reference/techart/jsr168/).

WebSphere Portal, starting with version 5.0.2.1, provides a runtime environment for both the IBM Portlet API, which is based on org.apache.jetspeed.portlet interfaces and JSR 168 compliant portlets.

3.2.1 JSR 168 portlet modes

Portlets will perform different tasks and create different output depending on the function they are currently performing. Modes enable the portlets to provide different function for the task that is required. JSR 168 supports three modes: view, edit, and help. JSR 168 also supports custom modes. At the time of this writing, IBM WebSphere Portal only supports the custom mode configuration. A portlet can change the mode programmatically in the processAction method. You can also specify the mode when creating an action or render link.

View

The view mode is used for displaying content reflecting the current state of the portlet. The view mode can include multiple screens that you can navigate. The doView() method of the GenericPortlet class is invoked for this mode. All portlets are required to support the view mode.

Edit

The edit mode is used for customizing the behavior of the portlet by modifying the PortletPreferences object. As with the view mode, the edit mode can contain multiple screens for navigation. The doEdit() method of the GenericPortlet class is invoked for this mode. Portlets are not required to support the edit mode.

Help

Help should be used to provide the user with information about the portlet. This could be generic or it could provide context-sensitive help. The doHelp() method of the GenericPortlet class is invoked for this mode. Portlets are not required to support the help mode.

Custom-portlet-mode

The configuration mode is used to globally update a portlet configuration. In configuration mode, an administrator can update the read-only PortletPreferences. Changes are reflected in all occurrences of the portlet on all pages. The doDispatch() method of the GenericPortlet class needs to be overridden to handle this custom mode.

Note: WebSphere Portal only supports the custom-portlet mode configuration.

3.2.2 Core JSR 168 object

This section describes the core object used when developing a JSR 168 portlet. Some important methods of each object are also discussed.

3.2.2.1 Interface javax.portlet.Portlet

All portlets must implement the javax.portlet.Portlet interface directly or by extending a class that implements this interface. A main method that is defined in this interface is the processAction method.

The void processAction(ActionRequest request, ActionResponse response) method

The processAction method is called in response to an action request. URLs generated by the portlet using RenderResponse.createActionURL() or the `<ActionURL JSP>` tag generate an action URL causing this method to be invoked. This method should be used to update the portlet's state based on the action-request parameters. Two objects are passed in: an ActionRequest and an ActionResponse.

The ActionRequest provides access to the parameters, window state, portlet mode, portlet context, portlet session and the PortletPreferences object. During the action request, the portlet can issue a redirect to a different URL.

While processing an action request, the portlet window state and the portlet mode can be changed. This change would be reflected in the next render phase. It should not be assumed that the portlet will change mode or window state because WebSphere Portal server can override the change. Changes to the window state and mode are made through the ActionResponse object.

3.2.2.2 Class javax.portlet.GenericPortlet

The GenericPortlet class implements the Portlet interface and does provide default implementations of the methods. All portlets should extend the GenericPortlet class rather that implementing the Portlet interface directly. The GenericPortlet implements many of the methods and reduces the workload when developing the portlet.

The void doDispatch(RenderRequest request, RenderResponse response) method

The doDispatch method is called from the render method. The doDispatch method determines the portlet mode and invokes the correct method. If using a custom mode, you should override the doDispatch method to test for your custom mode. If the request is not for your custom mode, then invoke the super. doDispatch method to allow the GenericPortlet do dispatch the correct method.

The void doEdit(RenderRequest request, RenderResponse response) method

When the portlet mode is edit, the doEdit method is invoked by the doDispatch method.

The void doHelp(RenderRequest request, RenderResponse response) method

When the portlet mode is help, the doHelp method is invoked by the doDispatch method.

The void doView(RenderRequest request, RenderResponse response) method

The doview method is invoked by the doDispatch method when the portlet mode is view.

The void processAction(ActionRequest request, ActionResponse response) method

The processAction method is invoked for all action requests for the portlet.

3.2.2.3 Interface javax.portlet.PortletURL

The PortletURL interface is used to create URLs that reference the portlet. There are two types of PortletURLs: ActionURL and RenderURL. By using the RenderResponse.createActionURL, RenderResponse.createRenderURL or the JSP tags, the portlet developer can create the different types of URLs. When a render URL is encountered, the render method is invoked and it in turn invokes the appropriate doXXX method. When an action URL is encountered, the processAction method is invoked. Only action URLs should be used for forms. The ActionURL is also used in the dashboard software development kit (SDK).

The void setParameter(java.lang.String name, java.lang.Stringvalue) method

This method is used to add parameters. Parameters added for a render URL are only available during the render request. Parameters added for an action URL are only available for the action request unless they are specifically added using the ActionResponse.setRenderParameter method.

The java.lang.String toString() method

This method returns a URL in the correct form for the portal server. Here is a code sample that illustrates how to use an action URL in a link:

```
PortletURL saveURL= renderResponse.createActionURL();
saveURL.setParameter("ACTION _ NAME","SAVE _ ACTION");
<a href="<%= saveURL.toString() %>">saveURL</a>
```

3.2.2.4 Interface javax.portlet.PortletRequest

The portlet request object contains information about the client request. The request also includes parameters, the portlet mode, the window state, session, and access to the portlet context. The PortletRequest interface defines commonly used functionality for the ActionRequest and RenderRequest methods.

Parameters that are received during an action request are not sent to the render request unless they are explicitly added using the setRenderParameters or setRenderParameters of the ActionResponse class. This can only be done in the processAction method.

If the render request follows an action request as part of the same request, the parameters sent in the render request will be the render parameters set during the action request.

A portlet can only see parameters in its own request. Parameters set for other portlets are not visible.

Request properties are used for portal-specific properties. These properties can include http headers.

3.2.2.5 Interface javax.portlet.ActionRequest

The ActionRequest interface extends the PortletRequest interface and is used during an action request. The ActionRequest object is passed into the processAction method. The ActionRequest object is in scope only during execution of the processAction method.

3.2.2.6 Interface javax.portlet.RenderRequest

The RenderRequest interface extends the PortletRequest. The RenderRequest interfacedoes not define any additional functionality. The RenderRequest object is scoped to the render method.

3.2.2.7 Interface javax.portlet.PortletResponse

The portlet response object contains information to be returned to the portlet during a request. Examples of this include redirection, portlet mode change, title, content, and so on. The portlet response object is passed into the render and the processAction methods. The portlet response object provides a way to add or update the response properties.

3.2.2.8 Interface javax.portlet.ActionResponse

The ActionResponse interface extends the PortletResponse interface. The ActionResponse is used during an action request and is passed into the processAction method. ActionResponse includes added functionality for changing the portlet mode, changing the portlet window, setting render parameters, and redirecting to another URL. The ActionResponse object is only in scope during the processAction method.

3.2.2.9 Interface javax.portlet.RenderResponse

The RenderResponse interface extends the PortletResponse interface. This object is passed to the render method. This object is used to set the title of the portlet and generate content by either obtaining a writer or delegating to a JSP or servlet. The scope of this object is only for the render method.

3.2.2.10 Interface javax.portlet.PortletPreferences

The PortletPreferences object is used to provide a customized view of the portlet to the user. Preferences are stored as name-value pairs. They can be either defined in the portlet deployment descriptor or programmatically. If the parameters are defined in the deployment descriptor, they have the option of being read-only. Read-only parameters can only be updated by the administrator while in configuration mode. The administrator can also modify the parameters using the administration portlets. If the parameters are added programmatically, they are not considered to be read-only. Users can modify parameters only while in the edit mode and only parameters that are not read-only. Changes made in configuration mode by the administrator affect all instances of the portlet on all pages.

3.2.3 Deployment descriptors

Deployment descriptors provide information to the server about the application. Two deployment descriptors are required for portlets: web.xml and portlet.xml. The web.xml (Web-deployment) descriptor is used to define all non-portlet resources. Compared to the IBM portlet API, JSR 168 does not define a servlet, so you do not have to define a servlet in the web.xml deployment descriptor. Remember that in JSR 168, portlets do not extend the HttpServlet interface like they do in the IBM API. Therefore, portlets are not servlets and are not required to go into the web.xml deployment descriptor.

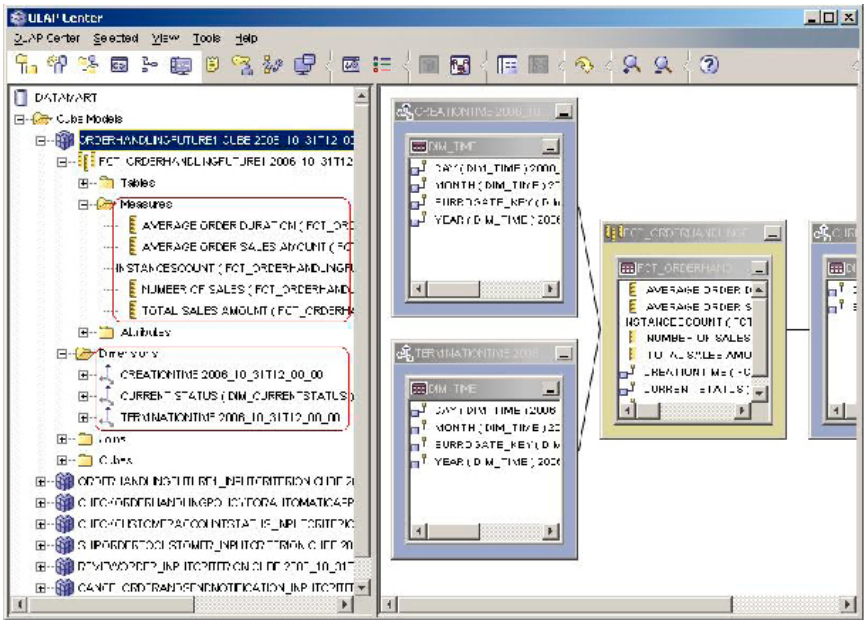
The portlet.xml deployment descriptor is used to define all portlet-related resources. You should use IBM Rational® Application Developer to create and modify your portlet.xml deployment descriptor. Rational Application Developer contains tools for inserting and removing elements and also for verifying that the portlet.xml deployment descriptor is valid.

3.3 DB2 CubeView and MDX query

DB2 Cube Views is used to create a multidimensional cube model on top of a relational DB2 database. Each cube model has its own dimension names and measure names. The cube model can then be used by DB2 Alphablox for further data analysis and reporting. In WebSphere Business Monitor, the contents in the Datamart database are exposed to portlet application through the Cube Views and Alphablox interface.

The file Model_cv.xml exported from the Monitor schemagen is used to tell DB2 Cube Views what dimensions and measures are to be exposed in the cube of a monitoring model. This file can be imported into the DB2 Cube Views by the DB2 OLAP GUI. After the import, you can also check what dimensions and measures are defined in a cube model from the DB2 OLAP GUI. The following screen capture displays the measures and dimensions for cube mode – "OrderHandlingFuture1 Cube 2006_10_31T12_00_00" – in the DB2 OLAP GUI.

Figure 3-3
Cube model of ClipsAndTacks process



Notes:

- There are seven cube models that are exposed for this process model.
- The cube model named “OrderHandlingFuture1 Cube 2006_10_31T12_00_00” is expanded in the DB2 OLAP GUI. You can see the measures and dimensions that are defined in it.
- There are four user-defined measures in this process model: “Average Order Duration,” “Average Order Sales Amount,” “Number of Sales,” “Total Sales Amount.” The other measure named “InstanceCount” is the predefined measurefor all process models by Monitor schemagen.
- Only the “Current Status” is the user-defined dimension in this process model. The other two dimensions: “CreationTime” and “TerminationTime” are the predefined dimensions for all process models by Monitor schemagen.

The sample portlet needs to retrieve the value of the measures from a selected cube model and perform the data analysis based on the user-selected dimensions. The language used for querying the multidimensional data source is MDX query language. For example, if you want to retrieve the value of the measures “InstancesCount” and “Total Sales Amount” from “OrderHandlingFuture1 Cube 2006_10_31T12_00_00” cube, and want to perform the data analysis based on the dimensions “Creation Time,” the MDX query statement for this purpose is:

```
select {[Measures].[InstancesCount], [Measures].[Total Sales
Amount]} on columns ,
[Location] on rows
from [OrderHandlingFuture1 Cube 2006 _ 10 _ 31T12 _ 00 _ 00]
```

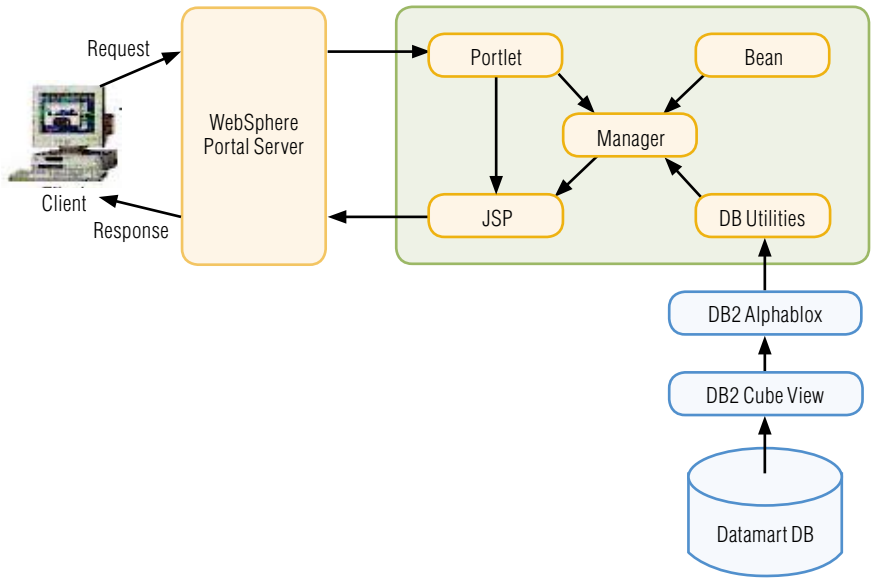
For a comprehensive explanation of the MDX language, see msdn2.microsoft.com/en-us/library/ms145506.aspx.

4 Architecture & Design

4.1 Architecture

The following diagram shows the architecture of the sample portlet.

Figure 4-1
Architecture of sample portlet



Notes:

- User's request (render or action) is accepted by IBM Websphare Portal Server and is forwarded to the portlet.
- The Java bean is responsible for storing user selection information in the configuration mode.
- The dbUtil(what is this?) is responsible for accessing database through DB2 Cube Views and DB2 Alphablox.
- The manager object bridges the gap between the portlet/JSP and bean/dbUtil.
- The portlet object invokes the JSP file to display the information it gets from the bean and dbUtil.
- JSP is responsible for the page rendering and response to the user through Websphere Portal Server.

4.2 Design

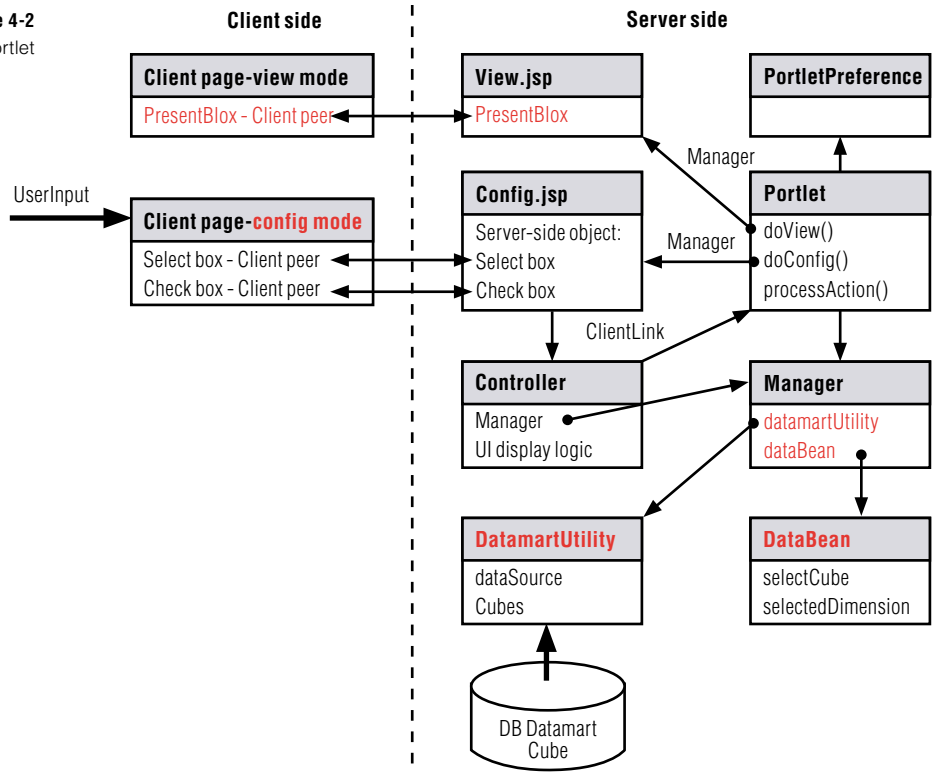
WebSphere Business Monitor Dashboard portlet makes rich use of the Alphablox component for database access, UI construction and data analysis.

Retrieving user-entered values in an Alphablox UI component from the configuration mode and passing them to a blox in view mode can be a challenge. The Alphablox server-side event handlers do not have access to the standard portlet-programming model. This means that when you perform an action on an Alphablox UI component, the portlet request and response, on which much of the portlet functionality is dependent, are not available.

Because of this challenge, a way is needed to bridge both the Alphablox and portlet-programming models. The introduction of the portlet manager class serves this purpose. The portlet manager can be reused between the view mode and the configuration mode. Its job is to shuttle information between the portlet and the Alphablox UI Controller.

The following diagram shows the class diagram of the sample portlet with the manager to bridge between Alphablox component and the portlet mechanism.

Figure 4-2
Design diagram of sample portlet



Notes:

- The portlet class holds a reference to the Manager. The manager has references to the Datamart utility, which is responsible for all database accesses, and to DataBean, which is used for storing the user-selection information.
- In the portlet-programming model, the manager object can be accessed in the JSP files.
- JSP includes Alphablox blox components for data rendering and collecting user selections. All the Alphablox components in both portlet modes can access the manager in the JSP, and then they get the access to the Datamart utility and DataBean accordingly.
- The controller class is responsible for constructing the UI with Alphablox components in configuration mode and handling the user-selection event triggered on the Alphablox components. It can also retrieve the manager reference from the JSP, so that it can store the user-selection information into the DataBean through the manager reference.
- The controller class could issue a portlet request by dispatching the ClientLink object, which is used to store a URL to the portlet itself. After dispatching the ClientLink object, the control goes back to the portlet request/response model and the processAction method () is invoked to process the request from the controller.
- The PortletPreference is a place to persist the portlet data (user selection) even after a portlet session is closed.

5 Development Steps

5.1 Prepare the development environment

The following tools are used for the sample portlet development:

- *Rational Application Developer V6.0 with Portal Tools*
- *IBM Alphablox tag library plug-in*

Notes: The standalone Portal tools plug-in is not available for WID or Eclipse. Without this plug-in, developer will have to create a portlet project from the scratch. Therefore, we highly recommend using Rational Application Developer as the portlet development tool.

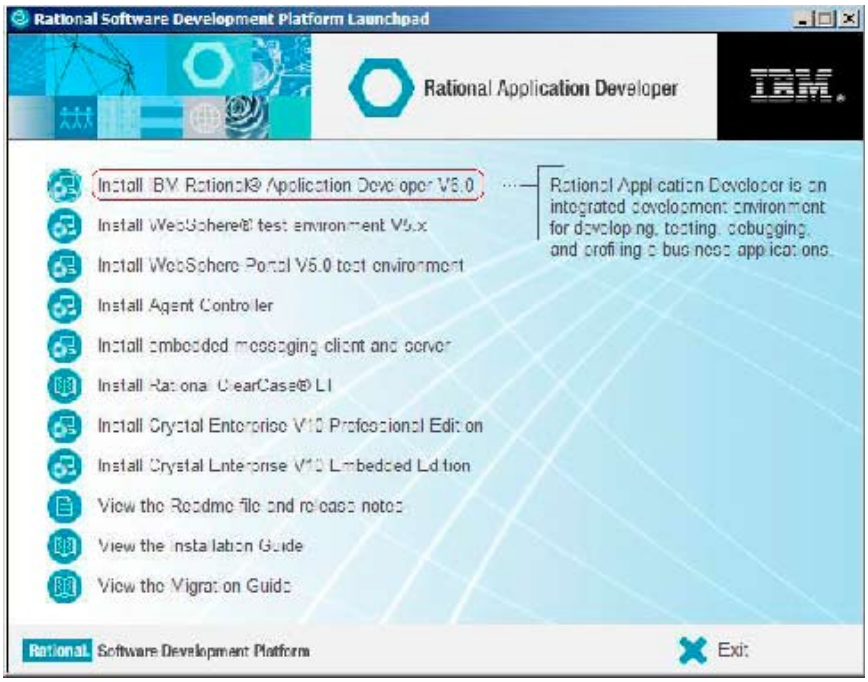
5.1.1 Installation of Rational Application Developer V6.0

Prerequisite: Rational Application Developer installation CD image is prepared.

Follow these steps to install Rational Application Developer V6.0:

1. *Run the launchpad.exe file from the CD image to display the Rational Software Development Platform Launchpad.*

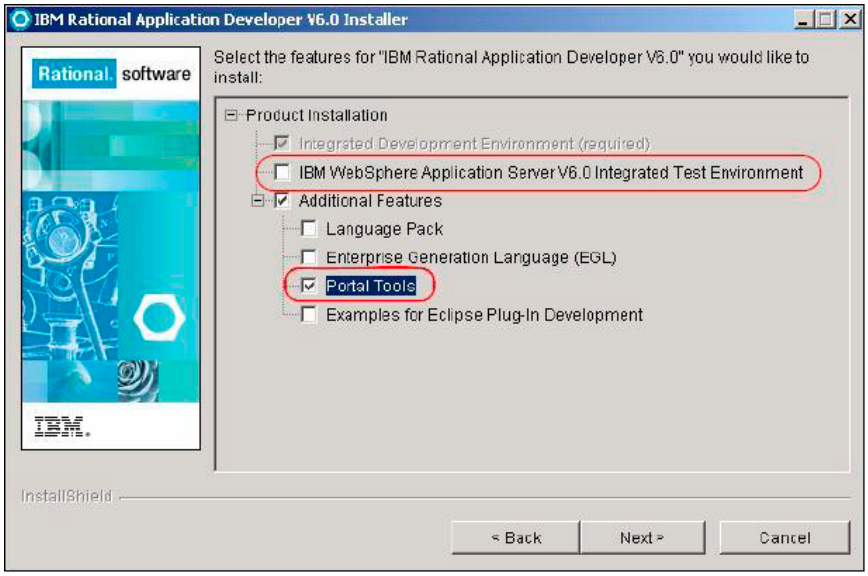
Figure 5-1
Launchpad for Rational Application Developer



2. *Select **Install IBM Rational Application Developer v6.0** (see Figure 5-1).*
3. *Click **Next** to continue after the installation program opens.*
4. *Accept the license agreement, and click **Next** to continue.*
5. *Accept the default installation path: **C:\Program Files\IBM\Rational\SDP\6.0**.*

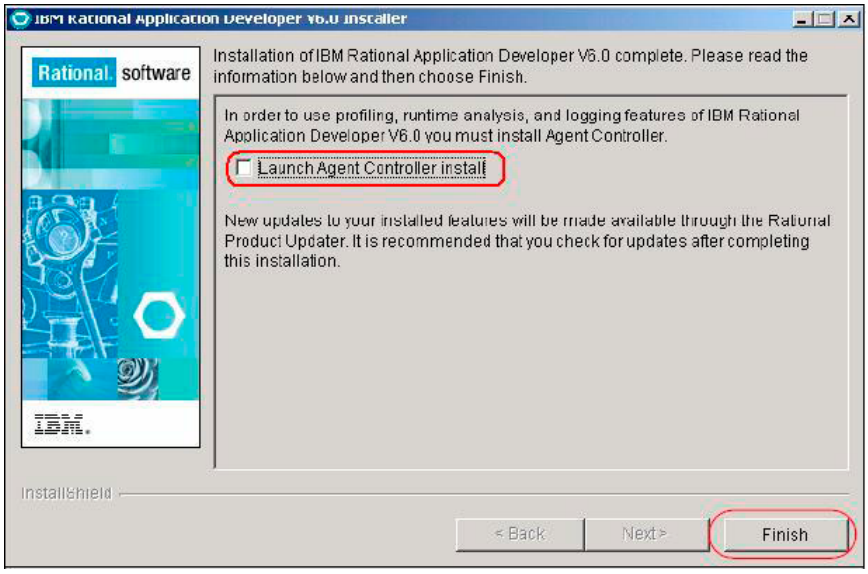
6. *In the features window, you can clear the IBM WebSphere Application Server V6.0 Integrated Test Environment check box. It is not required for portlet application development.*
7. *(Important) Select* ***Portal Tools*** *under the Additional Features heading. This option is required for portlet development (see Figure 5-2).*
8. *Click* ***Next*** *to continue.*

Figure 5-2
Launchpad option: Portal Tools



9. *In the summary information window, click* ***Next*** *to continue with the installation.*
10. *Click* ***Next*** *to continue after this installation is complete.*
11. *Clear the Launch Agent Controller install check-box option. Click* ***Finish*** *to complete this installation (see Figure 5-3).*

Figure 5-3
Launchpad: Finish

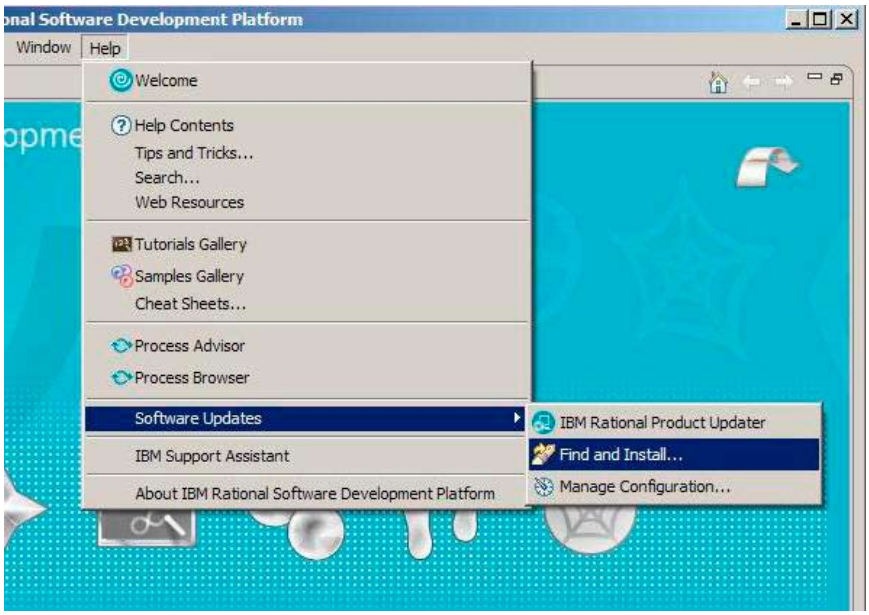


5.1.2 Installation of IBM Alphablox tag library plug-in

Because the sample portlet will make use of IBM Alphablox components, the IBM Alphablox tag library plug-in is required to be installed in Rational Application Developer V6.0.

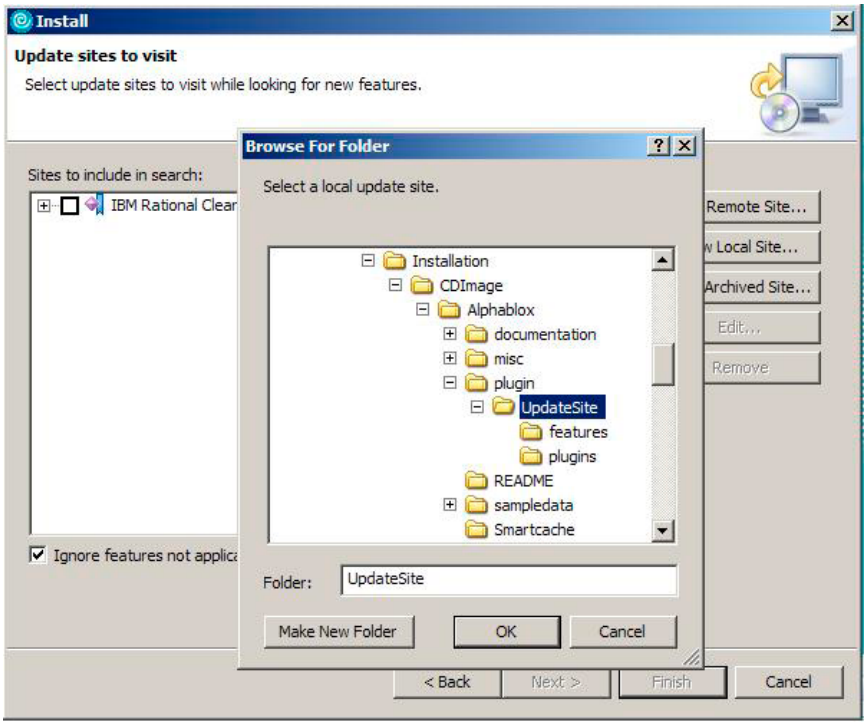
1. *Run Rational Application Developer V6.0 from Windows* ***Start→ All Programs→ IBM Rational→ IBM Rational Application Developer V6.0→ Rational Application Developer****.*
2. *Open the Feature Updates dialog from menu* ***Help→ Software Updates→ Find and Install*** *(see Figure 5-4).*

Figure 5-4
Installation of Alphablox tag library



3. *In the Features Updates dialog, select* ***Search for new features to install****.*
4. *Click* ***Next*** *to continue.*
5. *In the Update sites dialog, click* ***New Local Site...,*** *then browse to the Alphablox plug-in directory. The Alphablox plug-in directory is located on the Alphablox installation CD image. The full path might be different on your machine, but be sure to select the* ***UpdateSite*** *directory (see Figure 5-5).*

Figure 5-5
Location of Alphablox tag library



6. *Click **OK**, and then select the site you just added.*
7. *Click **Next** to continue.*
8. *Select **Install all the features in this site**.*
9. *Click **Finish** to complete the plug-in installation.*

Result: Now the development environment for portlet development is set up.

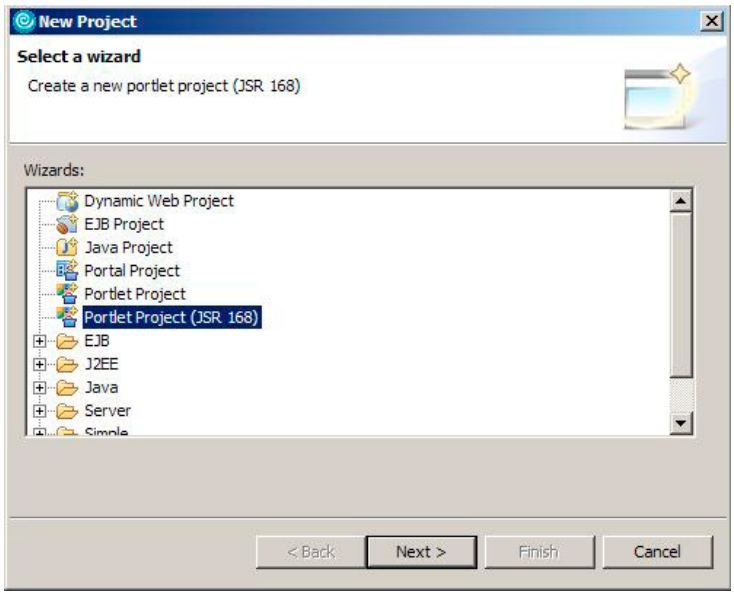
5.2 Implement the sample portlet in Rational Application Developer V6.0

5.2.1 Create a JSR-168 portlet project in Rational Application Developer

1. *Start Rational Application Developer on workspace C:\RadWs. Close the welcome page.*
2. *Click **File→ New→ Project** in the new project dialog, select a wizard for **Portlet Project (JSR 168)** (see Figure 5-6).*

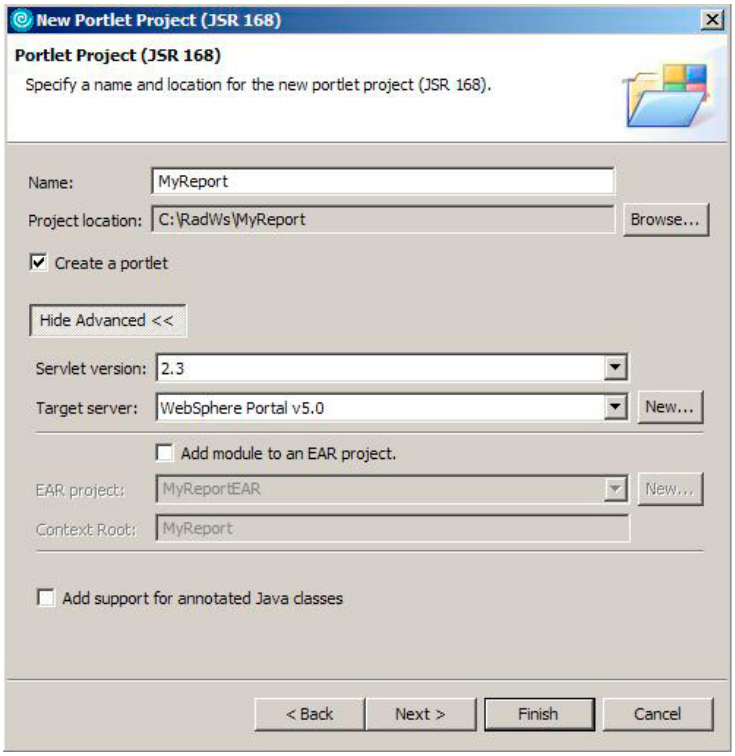
   Note: This wizard results from selecting the Portal Tools option during the Rational Application Developer V6.0 installation.

Figure 5-6
New portlet project: Step 1



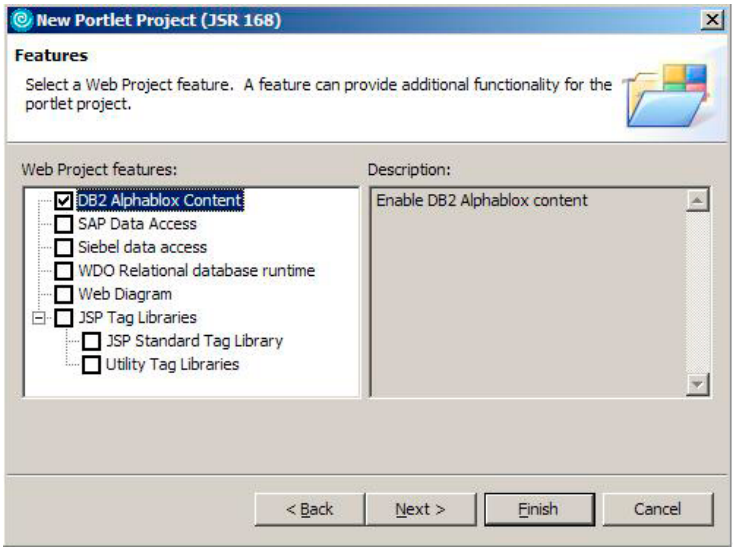
3. *Click **Next** to continue. The Confirm Enablement dialog opens.*
4. *Click **OK** to enable the portal development capability.*
5. *In the New Portlet Project (JSR168) wizard, name the new project* `MyReport`. *Click **Show Advanced**. Clear the check-box option **Add module to an EAR project** (see Figure 5-7).*
6. *Click **Next** to continue.*

Figure 5-7
New portlet project: Step 2



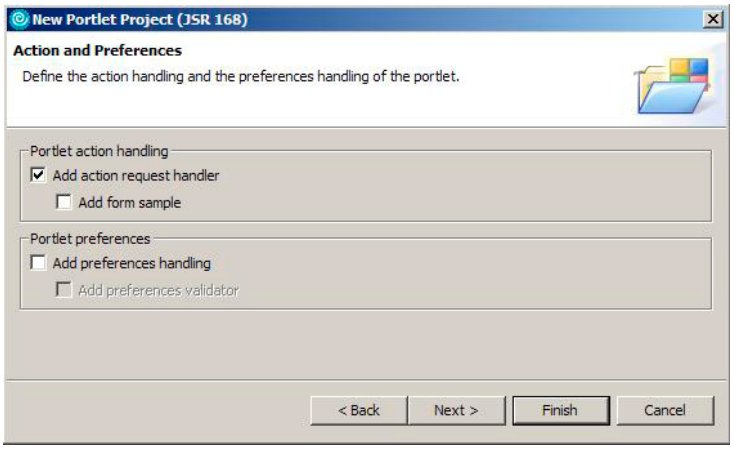
7. *In the Portlet Type dialog, select the type of new portlet as* ***Basic portlet (JSR168)****.*
8. *Click* ***Next*** *to continue.*
9. *In the Features dialog, clear the* ***Web Diagram*** *option check box and select the* ***DB2 Alphablox Content*** *option (see Figure 5-8). This option is required because our sample portlet uses the Alphablox component. It comes from the Alphablox tag library plug-in that is installed in Rational Application Developer V6.0.*
10. *Click* ***Next*** *to continue.*

Figure 5-8
New portlet project: Step 3



11. *In the DB2 Alphablox EAR File Locations dialog, leave the text field blank and click* ***Next****.*
12. *In the Portlet Settings dialog, click* ***Next*** *to continue.*
13. *In the Action and Preferences dialog, clear the* ***Add form*** *sample check box (see Figure 5-9).*
14. *Click* ***Next*** *to continue.*

Figure 5-9
New portlet project: Step 4



*15. In the Miscellaneous dialog, select **Add configuration mode**.*

*16. Click **Finish** to complete the new portlet project (JSR168) wizard.*

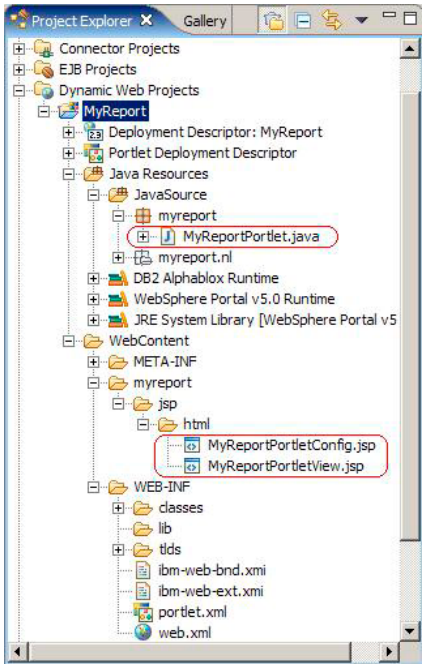
*17. When the Confirm Perspective Switch dialog opens, click **Yes** to confirm.*

Result: Now the wizard will create a basic JSR-168 portlet. You can add your code into it to implement your required functionality.

5.2.2 Review the newly created portlet

Before adding your own code into the newly created portlet, review the codes that wizard has created for you first. Expand the **MyReport** project in the Rational Application Developer Project Explorer as shown in Figure 5-10.

Figure 5-10
Project explorer



The key files to view are the portlet java file – MyReportPortlet.java and the two JSP files. The MyReportPortlet.java is the main Java class that handles the render and action request and response of the portlet. The two JSP files are responsible for presenting the portlet UI in configuration mode and view mode. Later you will add your own logic in these files to extend the portlet functionality.

Also, see the files portlet.xml and web.xml. They are the deployment-descriptor files of the portlet. To understand the content in these two files, refer to the JSR-168 portlet specification referenced at the end of this document.

5.2.3 The DatamartUtility class

You need to have a place to hold the database-access logic. For this purpose, there is a Java class named DatamartUtility.java. The complete source code of this class can be found in the attached source-code package. Here is a sample of the DatamartUtility.java class source code (also see Figure 5-XX shows).

```
public class DatamartUtility {

    private boolean isInitialized = false;  // Flag of initialization
    private String dataSource;              // Alphablox data source name
    private Cube[] cubes;                   // Cubes got from the data
                                              source

    /**
     * The constructor of datamart utility class.
     * In a normal Monitor installation, the Alphablox data source name
     * should be DATAMART _ Cube. We keep this name as a constant to
     * reduce hard code.
     */
    public DatamartUtility() {
        setDataSource(MyReportPortletConstants.DATA _ SOURCE);
    }

    /**
     * Load the metadata from the datablox, then get the cubes from the
     * metadata and keep a reference in this utility class.
     * @param dataBlox
     * @throws DataBloxCannotConnectException
     * @throws ServerBloxException
     */
    public void init(DataBlox dataBlox)
          throws DataBloxCannotConnectException, ServerBloxException {

        MDBMetaData mdbMetaData = (MDBMetaData) dataBlox.getMetaData();
        this.cubes = mdbMetaData.getCubes();
        setInitialized(true);
    }
}
```

Notes:

- The DatamartUtility class has a variable to hold the name of the Alphablox data source from which this portlet retrieves data. The data source is named DATAMART_Cube by default.
- The DatamartUtility class has a variable to hold the reference to cubes that are obtained through a datablox(. The cube object provides methods to access the dimensions and measures it contains.
- The init() method is called at the first portlet configuration-mode request. It retrieves the cubes from the database through a datablox component.
- The DatamartUtility class also provides methods for retrieving dimensions and measures from cubes.

5.2.4 The data bean class

The data bean class holds the user-selection information.

```
public class MyReportDataBean {

    // Saved user selection in the config mode
    private String selectedCube;
    private String selectedDimension;
    private HashSet selectedMeasures;

    // Unsaved user selection in the config mode
    private String selectedCubeTmp;
    private String selectedDimensionTmp;

    /**
     * The constructor of data bean class.
     */
    MyReportDataBean() {
        this.selectedMeasures = new HashSet();
    }

    /**
     * Check if this data bean has been configured.
     * @return boolean
     */
    public boolean isConfigured() {
        return ((selectedCube != null) && (selectedDimension != null)
&& (!selectedMeasures
                .isEmpty()));
    }
}
```

Notes:

- There are variables to hold the user selection in the configuration mode of the portlet. These variables are divided to two groups. One group is for holding the temporary selection before a user clicks Save. The other group is for holding the final selection after a user clicks Save.
- The data bean class provides a public method named isConfigured() to check if it is configured properly in the configuration mode.
- Another important item not shown in this code is the Getter and Setter methods for all class-member variables, which are required for this class. These methods can be generated by the utility in Rational Application Developer.

5.2.5 The portlet manager class

This section explains how the manager class bridges the gap between the portlet and the Alphablox UI component.

1. *Declare the manager class.*

   To pass information from the configuration mode to the view mode, the manager class should be able to be accessed in both modes. This object is achieved by persisting the manager class in the portlet session. Each time the view mode or configuration mode of the portlet is requested, the manager is copied from the portlet session to the attribute of the request. The following code shows the declaration of the manager class.

```
public class MyReportPortletManager implements Serializable {

    // Reference to the data bean.
    private transient MyReportDataBean dataBean;
    // Reference to the Datamart utility class.
    private transient DatamartUtility datamartUtility;
    // Flag to indicate if this manager requires initialization.
    private transient boolean isInitialized = false;
```

Notes:

- The manager class should implement the java.io.Serializable interface to support the persistence.
- sThe manager class holds references to the Datamart utility and DataBean. Through the manager class and these references, both the configuration and view modes of the portlet and the Alphablox UI component can access the Datamart utility and data bean conveniently.
- There is a special design concern for the cluster environment here. In a cluster environment, the manager object might be serialized and re-created in the session of another machine during the failover process. The reference to the DataBean will become invalid in the new Java virtual machine (JVM) because the DataBean object itself will be re-created also. The key word "transient" tells Java not to save the value of the data bean reference during the persist process. However, its value should be recalculated in the new JVM. A flag (isInitialized) is used to indicate if the manager is required to re-create the data bean object. A later section examines the initialization process of the manager object.

2. *Save the manager object and place it in a portlet session.*

   The following code shows the logic in the getPortletManager () method of the MyReportPortlet class, which is how the manager class is stored and pulled from the portlet session. Also, a new manager object is created if there is not one in the portlet session already. When the manager is not initialized, it is to maintain the session persistence in a cluster environment.

```
private MyReportPortletManager getPortletManager(PortletRequest
request,
        PortletResponse response) {
    // Pull the manager from the session
    MyReportPortletManager manager = (MyReportPortletManager)
           request
            .getPortletSession().getAttribute(
                    MyReportPortletConstants.MANAGER);

    // If no manager exists, create one and add it to the session
    if (manager == null) {
        manager = new MyReportPortletManager();
        manager.init(request, response);
        request.getPortletSession().setAttribute(
                MyReportPortletConstants.MANAGER, manager);
    } else if (!manager.isInitialized()) {
        // If the manager is failover from a cluster environment,
        // it might not be initialized yet, call the init()
              method now.
            manager.init(request, response);
    }
    return manager;
}
```

3) *Initialize the manager class.*

The following code shows how the manager is initialized. The major job of the initial process is to create the Datamart utility object and the data bean object, and then load the preconfigured information from the portlet preference.

```
public void init(PortletRequest request, PortletResponse response) {

    this.dataBean = new MyReportDataBean();
    this.datamartUtility = new DatamartUtility ();

    // Load the pre-configured information from the portlet
       preference
    // if exist.
    try {

        loadPreference(request);

        setInitialized(true);

    } catch (Exception e) {
        e.printStackTrace(System.out);
    }
}
```

4) *Retrieve the manager reference in the doView () and doCustomConfigure () methods.*

The following code describes how the manager class is accessed in the portlet's doView () method. The manager reference is pulled out from the portet's session by the getPortletManager () method and then the reference is stored in the portlet request's attribute. The doView () method finally invokes the JSP file to construct the page for the UI in view mode. With the manager's reference stored in the portlet request's attribute, the JSP code for the view mode can access the manager easily. The doCustomConfigure () method uses the same logic to access the manager class.

```
public void doView(RenderRequest request, RenderResponse response)
        throws PortletException, IOException {
    // Set the MIME type for the render response
    response.setContentType(request.getResponseContentType());

    // Get the portlet manager
    MyReportPortletManager manager = getPortletManager(request,
    response);

    // Put the portlet manager in the request
    request.setAttribute(MyReportPortletConstants.MANAGER,
    manager);

    // Invoke the JSP to render
    PortletRequestDispatcher rd = getPortletContext().get
    RequestDispatcher(
            getJspFilePath(request, VIEW _ JSP));
    rd.include(request, response);
}
```

5.2.6 Interaction 1: Initialize portlet view mode

In this section(these are not really chapters, but sections of chapters), you will initialize the portlet's view mode. The following code comes from MyReportPortletView.jsp file. This JSP file is invoked in the portlet's doView () method when a render request of view mode is issued.

```
<%@ page session="false" contentType="text/html" %>
<%@ page import="java.util.*,javax.portlet.*,myreport.*" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<%@taglib uri="bloxtld" prefix="blox" %>
<%@taglib uri="bloxuitld" prefix="bloxui" %>
<%@taglib uri="bloxportlettld" prefix="bloxportlet" %>

<portlet:defineObjects/>
<blox:header/>

<%
    // Retrieve the manager from request and then get the data bean
       from the manager
    MyReportPortletManager manager = (MyReportPortletManager)request
            .getAttribute(MyReportPortletConstants.MANAGER);

    MyReportDataBean dataBean = manager.getDataBean();
    if (!dataBean.isConfigured()) { %>
        This view has not been configured. Click the Configure icon on
the portlet toolbar to start the configuration. <%
        return;
}
%>
```

Notes:

- The first three lines of this JSP file are already created by the Rational Application Developer portlet wizard. They provide support for accessing the portlet's Java code and the portlet's predefined object.
- The three taglib directives instruct the JSP compiler to support the Alphablox-related tags.
- The `<blox:header/>` tag manages rendering of Alphablox blox on the pages.
- The `<portlet:defineobjects/>` tag manages the render request and render response of the portlet.
- The Java code part of this JSP file does the following things: Retrieves the reference of portlet manager from the render request's attribute; retrieves the reference of data bean from the manager; and checks if the data bean is configured properly and displays information text if it is not.

5.2.7 Interaction 2: Load configuration mode

As a standard way of using a portlet, clicking the "wrench" icon on the top-right side of the portlet window will put the portlet in Configuration mode.

Figure 5-11
Config icon

The request will be handled by the doCustomConfigure() method in the portlet class MyReportPortlet.java and it will invoke the MyReportPortletConfig.jsp to render the configuration mode page. The following snippet MyReportPortletConfig.jsp.

```
<%@ page session="false" contentType="text/html" %>
<%@ page import="java.util.*,javax.portlet.*,myreport.*"%>
<%@ page import="com.alphablox.blox.DataBlox" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<%@taglib uri="bloxtld" prefix="blox"%>
<%@taglib uri="bloxuitld" prefix="bloxui"%>
<%@taglib uri="bloxportlettld" prefix="bloxPortlet"%>

<portlet:defineObjects/>
<blox:header />

<%
        // Retrieve the manager from request
        MyReportPortletManager manager = (MyReportPortletManager)
        request
                .getAttribute(MyReportPortletConstants.MANAGER);

        DatamartUtility datamartUtility = manager.getDatamart
        Utility();
        String dataSource = datamartUtility.getDataSource();

        // Encode the blox name
        String dataBloxName = renderResponse.getNamespace()
                + MyReportPortletConstants.CONFIG _ DATA _ BLOX;
        String containerBloxName = renderResponse.getNamespace()
                + MyReportPortletConstants.CONFIG _ CONTAINER _ BLOX;
%>

<blox:data id="configDataBlox" bloxName="<%=dataBloxName%>"
        dataSourceName="<%=dataSource%>" >
    <%
        // Load the cube meta info into the data bean through
          data blox.
        if (!datamartUtility.isInitialized()) {
            datamartUtility.init(configDataBlox);
        }
    %>
</blox:data>

<blox:container id="configContainerBlox" bloxName="<%=containerBlox
Name%>"width="100%" >
    <%
        // Construct and display the config layout in this container
           blox.
        MyReportPortletConfigUI configUI = new MyReportPortlet
        ConfigUI(manager);
        configUI.display(configContainerBlox, (RenderRequest)
        renderRequest,
                    (RenderResponse)renderResponse);
    %>
</blox:container>
```

Notes:

- The taglib directives are imported in a similar way to the JSP file for view mode.
- A reference to the portlet manager is extracted from the request's attribute.
- A DataBlox is created for initializing the Datamart utility.

- The blox name should be prefixed with the portlet's namespace to make it unique. This is required because more than one instance of a portlet can exist on a page. If the blox names are not unique, the portlets have difficulty processing the duplicate names and errors occur.
- The MyReportPortletConfigUI class is introduced here. It is responsible for constructing the UI of the configuration mode page and handling the user-selection event in the page.
- This sample has the requirement that the UI components survive a refresh. This is especially important in the portal world because a user could choose to change the portlet window size, which refreshes the page and calls this JSP file again. The Java code that displays the configuration page is embedded in a container blox (inside of two blox tags). A characteristic of Alphablox blox tags is that the area within a blox tag is not refreshed when the JSP page is refreshed. This characteristic helps to retain the temporary selection values on the configuration page when a page refresh occurs.

The following code shows the declaration of the MyReportPortletConfigUI portlet class.

```
public class MyReportPortletConfigUI extends Controller {

    private MyReportPortletManager manager;  // Reference to
    portlet manager
    private ComponentContainer uiContainer;  // Container to hold
    UI components

    public MyReportPortletConfigUI(MyReportPortletManager manager) {
        this.manager = manager;
    }
}
```

Notes:

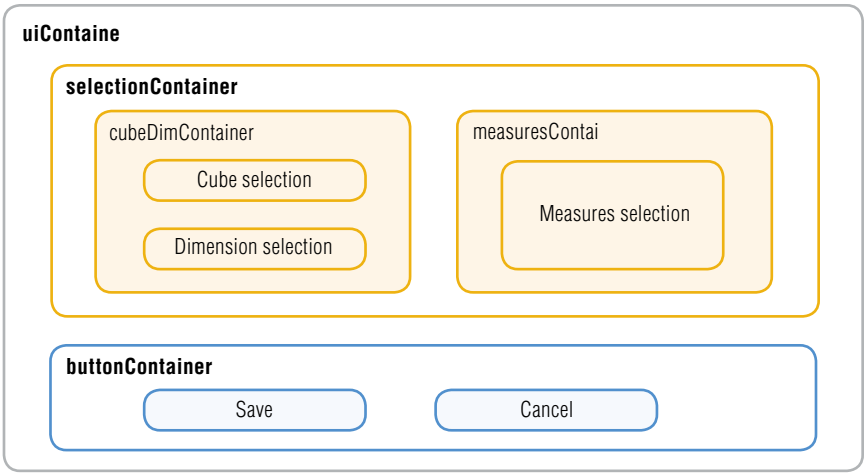
- Because you also want the MyReportPortletConfigUI class to handle the events coming from the Alphablox UI component, it is extended from the com.alppablox.blox.uimodel. core controller class.
- The MyReportPortletConfigUI class includes a reference to the portlet manager. The portlet manager bridges the event handler in this class and the data bean. Through this reference, the event handler can store the user-selection information it gets from the Alphablox UI component, into the data bean.

5.2.8 Interaction 3: Construct the UI in configuration mode

The display () method of the MyReportPortletConfigUI class is constructs the UI in the configuration mode. Though you can construct the UI in the JSP file by using the JSP tags for the Alphablox UI component, it is best to construct the UI with server-side Alphablox Java objectsto keep the JSP clean and simple. Through this method, you can also consolidate the UI components and associated event controllers in one class to keep the logic more straightforward.

The following diagram shows the logical layout of the configuration mode UI. All the UI components and component containers are made with server-side Alphablox Java objects.

Figure 5-12
XXXX



The following code shows the source for the display () method.

```
public void display(ContainerBlox containerBlox, RenderRequest
request,
        RenderResponse response) throws Exception {

    // Get the data Bean from portlet manager
    MyReportDataBean dataBean = manager.getDataBean();

    BloxModel bloxModel = containerBlox.getBloxModel();
    bloxModel.clear();

    // Prepare the component container and put it on container blox
    uiContainer = new ComponentContainer();
    uiContainer.setLayout(new VerticalLayout());
    uiContainer.setController(this);
    bloxModel.add(uiContainer);

    // Add the selection container and build the components on it.
    addSelectionContainer(dataBean);

    // Add the button container and build the buttons on it.
    addButtonContainer(dataBean, response);

}
```

Notes:

- The display() method constructs the UI on the container blox that is already created in the JSP.
- When constructing the UI, you need to fill the UI components with data obtained from the data bean. Therefore, a reference to the data bean is retrieved from the portlet manager.
- You can find much more-detailed steps to construct the UI in the addSelection Container() and addButtonContainer() methods. The logic for these two methods is straightforward. See the source code in the .zip of sample code that accompanies this document for reference.

5.2.9 Interaction 4: Handling events for cube or dimension user-selections

This section looks at how to handle the event when a user selects cubes or dimensions. The base class of the MyReportPortletConfigUI class is the Alphablox Controller class, which has provided many handler interfaces for different events. Among these interfaces, handleSelectionChangedEvent () is for handling the event when a selection has been changed in a drop-down list or a list blox. This method is the exact one that you need to implement. The following code shows the handleSelectionChangedEvent() method in the MyReportPortletConfigUI class.

```
public boolean handleSelectionChangedEvent(SelectionChangedEvent
event)
        throws Exception {

    // This method handle the selection change event on the
       DropDownList
    try {
        MyReportDataBean dataBean = manager.getDataBean();
        Component selectedComponent = event.getComponent();
        if (selectedComponent.getName().equals(
                MyReportPortletConstants.DROPDOWNLIST _ CUBE _ SELECT))
            // Selection is changed on cube select dropdownlist.

            DropDownList cubeList = (DropDownList) selectedComponent;
            DropDownList dimensionList = uiContainer
    .getDropDownList(MyReportPortletConstants.DROPDOWNLIST _
    DIMENSION _ SELECT);
        ComponentContainer measuresContainer = uiContainer
.getComponentContainer(MyReportPortletConstants.
            MEASURES _ CONTAINER);

            // Get the selected cube from the dropdownlist
            int selected = cubeList.getSelected();
            if (selected >= 0) {
                dataBean.setSelectedDimensionTmp(null);

                Cube selectedCube = (Cube) cubeList.
                getUserObject(selected);
                dataBean.setSelectedCubeTmp(selectedCube.getName());
            }
            populateDimensions(dimensionList);
            populateMeasures(measuresContainer);
        } else if (selectedComponent.getName().equals(
                MyReportPortletConstants.DROPDOWNLIST _ DIMENSION _
                SELECT)) {
            // Selection is changed on dimension select dropdownlist.

            DropDownList dimensionList = (DropDownList)
            selectedComponent;
            int selected = dimensionList.getSelected();

            // Get the selected dimension from the dropdownlist
            if (selected >= 0) {
                String selectedDimension = dimensionList.
                getLabel(selected);
                dataBean.setSelectedDimensionTmp(selectedDimension);
            }
        }
```

```
        } catch (Exception ex) {
            //TODO:
            ex.printStackTrace(System.out);
        }
        return false;
    }
```

Notes:

- Selections made before clicking the Save button are treated as temporary, because the user might cancel it. So the values obtained from the drop-down list are saved only in the xxxTmp variables in the DataBean.

5.2.10 Interaction 5: Handling events for Save or Cancel user-selections

This section explains how to handle the event of a user clicking the Save or Cancel button. A click event is triggered when the user clicks the button. The handleClickEvent() method of the Controller class is implemented in the MyReportPortletConfigUI class to handle this event. The following code shows the handleClickEvent () method.

```
public boolean handleClickEvent(ClickEvent event) throws
ModelException {
    // This method handle the click even on the save/cancel button

    MyReportDataBean dataBean = manager.getDataBean();
    Component selectedComponent = event.getComponent();
    if (selectedComponent.getName().equals(
            MyReportPortletConstants.SAVE _ BUTTON)) {
        // Save button is clicked

        // Save the temporary user selection on cube and dimension
        dataBean.setSelectedCube(dataBean.getSelectedCubeTmp());
        dataBean.setSelectedDimension(dataBean.getSelected
        DimensionTmp());

        // Save the user selection on measures.
        checkSelectedMeasures(dataBean);

        // Dispatch the URL
        ClientLink saveURL = (ClientLink) selectedComponent.get
        UserObject();
        uiContainer.getDispatcher().showBrowserWindow(saveURL);
    } else if (selectedComponent.getName().equals(
            MyReportPortletConstants.CANCEL _ BUTTON)) {
        // Cancel button is clicked

        // Restore the values of the temporary selection
        dataBean.setSelectedCubeTmp(dataBean.getSelectedCube());
        dataBean.setSelectedDimensionTmp(dataBean.getSelected
        Dimension());

        // Refresh the config page
        refreshConfig();

        // Dispatch the URL
        ClientLink cancelURL = (ClientLink) selectedComponent
                .getUserObject();
        uiContainer.getDispatcher().showBrowserWindow(cancelURL);
    }
    return false;
}
```

Notes:

- If a user clicks the Save button, the value saved in the data bean's xxxTmp variable should be copied to its corresponding xxx variable. On the other hand, if user clicks Cancel, you should restore the xxxTmp variables to their original values, and restore the configuration page according to the original selection values.
- The selection of measures is checked in this method (by calling the checkSeletedMeasures () method) instead of in the handleSelectionChangedEvent () method because you are only interested in the user's final, selected measure list when Save is clicked.
- Both the Save and Cancel buttons are attached to a user object, which is a ClientLink object, and is used to carry a URL that will be dispatched to the client.
- After dispatching the instruction to get the URL from the button's user object, control goes back to the portlet from the Alphablox event handler. This gives you a way to return to the portlet programming model from the Alphablox programming model.

To further understand the use of the ClientLink object, here is more detail into how the Save button and Cancel buttons are built. The populateButtons() method in the MyReportPortletConfigUI class is responsible for building these two buttons. It is called when a display () method is called.

```
private void populateButtons(ComponentContainer buttonContainer,
        RenderResponse response) throws ModelException {
    Button saveButton = new Button(MyReportPortletConstants.SAVE _
    BUTTON,
            MyReportPortletConstants.SAVE _ BUTTON _ TITLE);
    Button cancelButton = new Button(
            MyReportPortletConstants.CANCEL _ BUTTON,
            MyReportPortletConstants.CANCEL _ BUTTON _ TITLE);

    // Create an action URL from the portlet response, when a action
       URL is
    // opened, the portlet's processAction() is called.
    // The parameter set in the URL tells the processAction() it is a
    // Save action or a Cancel action.
    PortletURL saveURL = response.createActionURL();
    saveURL.setParameter(MyReportPortletConstants.ACTION _ NAME,
            MyReportPortletConstants.SAVE _ ACTION);

    PortletURL cancelURL = response.createActionURL();
    cancelURL.setParameter(MyReportPortletConstants.ACTION _ NAME,
            MyReportPortletConstants.CANCEL _ ACTION);

    // Store the action URL as ClientLink in the button's user object.
    // " _ self" instruct the browser open this URL in the current
       window.
    saveButton.setUserObject(new ClientLink(saveURL.toString(),
    " _ self"));
    cancelButton.setUserObject(new ClientLink(cancelURL.toString(),
    " _ self"));

    buttonContainer.add(saveButton);
    buttonContainer.add(new Spacer(0, 30));
    buttonContainer.add(cancelButton);
}
```

Notes:

- Each button is attached with a user object—the ClientLink—in which a URL is carried.
- The URL is an action-URL target to the portlet itself. When this action URL is opened, the portlet's processAction () method is called.
- The parameter set in the URL tells processAction () method if it is a "Save" action or a "Cancel" action.

Here is a look at how the portlet handles the action request. The following snippet shows the source code of the processAction () method of the MyReportPortlet.java class.

```
public void processAction(ActionRequest request, ActionResponse
response)
        throws PortletException, java.io.IOException {
    String actionName = request
            .getParameter(MyReportPortletConstants.ACTION _ NAME);

    // Get the portlet manager
    MyReportPortletManager manager = getPortletManager(request,
    response);

    // Go back to view mode.
    if (MyReportPortletConstants.SAVE _ ACTION.equals(actionName)) {

        // Save the data bean content into portlet preference
        manager.savePreference(request);

        response.setPortletMode(PortletMode.VIEW);
        response.setWindowState(WindowState.NORMAL);
    } else if (MyReportPortletConstants.CANCEL _ ACTION.
      equals(actionName)) {

        response.setPortletMode(PortletMode.VIEW);
        response.setWindowState(WindowState.NORMAL);
    }
}
```

Notes:

- If user clicked on the save button, the processAction () get a “Save” action. We tell manager save the selection values in data bean into the portlet’s preference. The value stored in portlet’s preference can be kept even the portlet session is closed. This is exactly what the configuration mode expects. If the action is a “Cancel” action, then the portlet just renders the view mode and does nothing else.
- The Response.setPortletMode (PortletMode.VIEW) method brings the portlet back to the view mode and a doView () method is called.

5.2.11 Interaction 6: Display the user-select values in view mode

The view mode is rendered with the user-selection values saved in the data bean. The doView () method calls MyReportPortletView.jsp to finish its rendering task. The following snippet shows the source code for displaying the PresentBlox component.

```
<%
    // Get data source and MDX query statement from data bean.
    String dataSource = manager.getDatamartUtility().get
     DataSource();
    String query = dataBean.generateMDXQuery();

    // Encode the blox name.
    String myPresentBloxName = renderResponse.getNamespace()
            + MyReportPortletConstants.VIEW _ PRESENT _ BLOX;

    // Default blox height
    String bloxHeight = “400”;
%>
```

```
<blox:present id="myPresentBlox" bloxName="<%=myPresentBloxName%>"
    width="100%" height="<%=bloxHeight%>" visible="false"
    dataLayoutAvailable="true" menubarVisible="false"
    toolbarVisible="false" chartFirst="true" >

    <blox:data dataSourceName="<%=dataSource%>"
query="<%=query%>"
        connectOnStartup="false" />

    <blox:chart dataTextDisplay="true" />

</blox:present>

<%
    // Update the query statement and result set to response the
       config changes.
    myPresentBlox.getDataBlox().setQuery(query);
    myPresentBlox.getDataBlox().updateResultSet();

    // The menubar and toolbar of the PresentBlox will be showed if
       the portlet
    // is maximized, otherwise they will be hidden.
    if (WindowState.MAXIMIZED.equals(renderRequest.get
    WindowState())) {
        myPresentBlox.setMenubarVisible(true);
        myPresentBlox.getToolbarBlox().setVisible(true);
        myPresentBlox.setHeight("100%");

    } else {
        myPresentBlox.setMenubarVisible(false);
        myPresentBlox.getToolbarBlox().setVisible(false);
        myPresentBlox.setHeight(bloxHeight);
    }

    // Display the PresentBlox.
    myPresentBlox.display(bloxRequest, bloxResponse, out);
%>
```

Notes:

- First the data source name and the MDX query are obtained from the data bean. The MDX query is generated according to the user-selection values in the data bean.
- A PresentBlox is added in the page by using the `<blox:present>` tag. The PresentBlox is a convenient way to perform the report and analysis tasks on multidimensional data sources by using a grid, chart and data-layout panel. The data source name and the query statement get from the data bean is passed to its nested data blox, which is responsible for the database access.
- The PresentBlox has a lot of options to control its layout. The data layout panel is opened, the menu bar and toolbar are disabled, and the chart panel is placed before the grid panel. Also the data values above each bar in the chart are displayed. For a full explanation of the PresentBlox options,see the Alphablox information center at publib.boulder.ibm.com/infocenter/ablxhelp/v8r4m0/index.jsp.
- The last part of the Java code outside of the blox tags is very important. A characteristic of the Alphablox blox tags is that the area within a blox tag will not be refreshed when the JSP page is refreshed. So if user makes changes in the configuration mode and returns to the view mode, though the data bean's query statement has been updated, the query statement of the PresentBlox is not updated unless you specify it again. Therefore, use the Java code outside of the blox tags to specify the query statement dynamically and an updateResultSet () tells the PresentBlox that the query result has been changed and the PresentBlox will update accordingly.
- The Java code also indicates that the menu bar and toolbar of the PresentBlox will appear if the portlet is maximized, but disappear if the portlet is at normal size. And the PresentBlox will make use of 100 percent of the window size if the portlet is maximized, but go back to a default height if the portlet is at normal size.

All the key concepts of the implementation of the sample portlet have now been explained. For the complete source code of the sample portlet, see the accompanying source code .zip file.

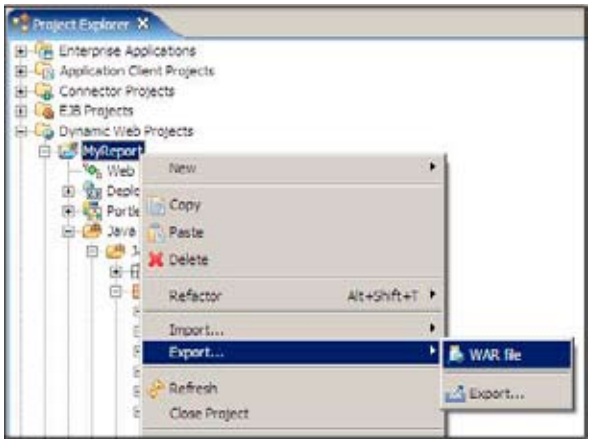
5.3 Deploy the sample portlet

This section describes the steps required to build the portlet Web archive (WAR) file and deploy it on the IBM Portal server.

5.3.1 Export the WAR file

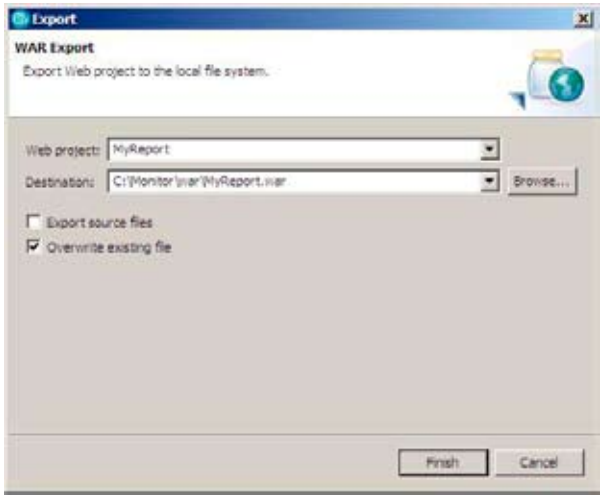
Rational Application Developer 6.0 has provided a convenient tool to build the WAR file for the portlet project. In the Project Explorer, right-click the portlet project name, and select **Export... → WAR file**, as shown in Figure 5-12.

Figure 5-13
Export WAR file: Step 1



A dialog for the export option displays (see Figure 5-13).

Figure 5-14
Export WAR file: Step 2



After specifying the project to be exported and the destination path, click Finish. The WAR file, MyReport.war is then created on the destination path.

5.3.2 Deploy the WAR file on portal server

Deploying the WAR file on the Portal server can be done by performing the following steps:

1. *Make sure that the Portal server is started. Open the Portal GUI in an Internet Explorer browser at localhost:9081/wps/portal.*
2. *Log in to the Portal GUI with the ID/password combination:* ***wpsadmin/wpsadmin****.*
3. *Click* ***Administration*** *at the top of the page, and then click* ***Portlet Management -> Web Modules****.*
4. *Click* ***Install****, and browse to the WAR file.*
5. *Click* ***Next****.*
6. *Click* ***Finish****.*

Result: The portlet is deployed on the Portal server.

6 References

- *JSR 168 portlet specification*
  (www.jcp.org/aboutJava/communityprocess/final/jsr168/)
- *Best Practices: Developing portlets using JSR168 and WebSphere Portal v5.02*
  (www-128.ibm.com/developerworks/websphere/library/techarticles/0403_hepper/0403_hepper.html)
- *Developing JSR168 Compliant Cooperative Portlets*
  (www-128.ibm.com/developerworks/websphere/library/techarticles/0412_roy/0412_roy.html)
- *IBM Rational Application Developer V6 Portlet Application Development and Portal Tools*
  (www.redbooks.ibm.com/abstracts/sg246681.html?Open)
- *IBM Alphablox infocenter*
  (publib.boulder.ibm.com/infocenter/ablxhelp/v8r4m0/index.jsp)
- *Multidimensional Expressions (MDX) Reference*
  (msdn2.microsoft.com/en-us/library/ms145506.aspx)

7 Appendix: Accompanying source code

A Rational Application Developer interchange compressed file, MyReport-interchange.zip, includes the source code for the MyReport portlet sample.

Note: Please import this interchange file with Rational Application Developer only.

8 Notices

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

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

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

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

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

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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

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

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

IBM Corporation
Department EZRA/ Building 502
4205 S MIAMI BLVD
Durham, NC 27703-9141
_U.S.A.

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

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

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

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

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
12-06