Building Composite, Role-Based Dashboards with WebSphere Dashboard Framework

Introduction to WebSphere Dashboard Framework

In-depth look at WebSphere Dashboard Framework builders

Leverage key features, such as Annotations and Alerts
Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (August 2007)

This edition applies to WebSphere Dashboard Framework Version 6.0.
## Contents

<table>
<thead>
<tr>
<th>Notices</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trademarks</td>
<td>viii</td>
</tr>
<tr>
<td></td>
<td>ix</td>
</tr>
<tr>
<td>Preface</td>
<td>ix</td>
</tr>
<tr>
<td>The team that wrote this paper</td>
<td>x</td>
</tr>
<tr>
<td>Become a published author</td>
<td>x</td>
</tr>
<tr>
<td>Comments welcome</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Chapter 1. Introduction to IBM WebSphere Dashboard Framework</td>
<td>1</td>
</tr>
<tr>
<td>1.1 What is WebSphere Dashboard Framework</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Features of WebSphere Dashboard Framework</td>
<td>4</td>
</tr>
<tr>
<td>1.3 How WebSphere Dashboard Framework fits into the IBM WebSphere Portlet Factory architecture</td>
<td>5</td>
</tr>
<tr>
<td>1.3.1 IBM WebSphere Portlet Factory</td>
<td>6</td>
</tr>
<tr>
<td>1.3.2 WebSphere Dashboard Framework</td>
<td>7</td>
</tr>
<tr>
<td>1.3.3 Lotus ActiveInsight</td>
<td>8</td>
</tr>
<tr>
<td>1.4 The benefit to the business</td>
<td>9</td>
</tr>
<tr>
<td>1.4.1 Why a dashboard</td>
<td>9</td>
</tr>
<tr>
<td>1.4.2 IBM Dashboard Accelerator</td>
<td>10</td>
</tr>
<tr>
<td>1.4.3 Why WebSphere Dashboard Framework</td>
<td>10</td>
</tr>
<tr>
<td>1.5 The example application</td>
<td>12</td>
</tr>
<tr>
<td>1.5.1 Technology used</td>
<td>13</td>
</tr>
<tr>
<td>1.6 Audience and goals of this IBM Redpaper</td>
<td>14</td>
</tr>
<tr>
<td>1.6.1 Skills required</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Chapter 2. Overview of portlet creation</td>
<td>15</td>
</tr>
<tr>
<td>2.1 Installing WebSphere Dashboard Framework</td>
<td>16</td>
</tr>
<tr>
<td>2.1.1 Planning the installation</td>
<td>16</td>
</tr>
<tr>
<td>2.1.2 Installing WebSphere Dashboard Framework with Eclipse</td>
<td>16</td>
</tr>
<tr>
<td>2.1.3 Notes about installing into Rational Products</td>
<td>19</td>
</tr>
<tr>
<td>2.2 WebSphere Dashboard Frameworks concepts</td>
<td>20</td>
</tr>
<tr>
<td>2.2.1 Accessing Help</td>
<td>20</td>
</tr>
<tr>
<td>2.2.2 WebSphere Portlet Factory concepts</td>
<td>21</td>
</tr>
<tr>
<td>2.3 Creating a WebSphere Dashboard Framework project</td>
<td>22</td>
</tr>
<tr>
<td>2.3.1 Step one: Setting the project name</td>
<td>23</td>
</tr>
<tr>
<td>2.3.2 Step two: Selecting the feature set</td>
<td>24</td>
</tr>
<tr>
<td>2.3.3 Step three: Java settings</td>
<td>26</td>
</tr>
<tr>
<td>2.3.4 Step four: Deployment configuration</td>
<td>27</td>
</tr>
<tr>
<td>2.3.5 Step five: Completion</td>
<td>30</td>
</tr>
<tr>
<td>2.4 Developing your first portlet using the framework</td>
<td>31</td>
</tr>
<tr>
<td>2.4.1 Step one: Add a model to the project</td>
<td>31</td>
</tr>
<tr>
<td>2.4.2 Step two: Exploring your model and the tooling</td>
<td>32</td>
</tr>
<tr>
<td>2.4.3 Project Explorer View</td>
<td>33</td>
</tr>
<tr>
<td>2.4.4 Step three: Adding the Excel data source to the builder</td>
<td>35</td>
</tr>
<tr>
<td>2.4.5 Step four: Displaying data</td>
<td>38</td>
</tr>
<tr>
<td>2.4.6 Step five: Running the model</td>
<td>39</td>
</tr>
<tr>
<td>2.4.7 Step six: Enhancing the model</td>
<td>43</td>
</tr>
<tr>
<td>2.4.8 Step seven: Adding the status indicator</td>
<td>45</td>
</tr>
<tr>
<td>2.4.9 Changing WebSphere Portlet Factory properties</td>
<td>47</td>
</tr>
</tbody>
</table>
4.3.19 Summary Row ................................................................. 114
4.3.20 Table Customizer ............................................................. 116
4.4 Customizing the look and feel ............................................... 116
  4.4.1 Creating a custom edit or configure page ......................... 116
  4.4.2 Customizing a threshold ............................................... 118
  4.4.3 Adding a Chart Customizer ........................................... 121
  4.4.4 Table Customizer ....................................................... 126

Chapter 5. Filtering ................................................................. 133
  5.1 Create a Query Filter ....................................................... 134
    5.1.1 Query Filter Form ................................................... 134
    5.1.2 Query Filter .......................................................... 135
  5.2 Enabling portlets to respond to filters ................................ 142
    5.2.1 Query Filter Observer .............................................. 142
  5.3 Profiling: Creating a role-driven filter ................................ 142

Chapter 6. Alerts ................................................................. 147
  6.1 Overview of alert capabilities .......................................... 148
  6.2 Configuration ............................................................... 149
    6.2.1 Step one: Selecting a persistence manager ..................... 149
    6.2.2 Step two: Adding alert features to your project ............ 151
    6.2.3 Step three: Setting up the database ............................ 151
    6.2.4 Step four: Setting up a data source ............................ 154
    6.2.5 Step five: Configuring your projects to use database persistence  .............................................. 154
    6.2.6 Step six: Creating an Alerts Scheduler project ............. 154
    6.2.7 Step seven: Configuring e-mail and SMS ....................... 155
  6.3 Configuring data for alerts ............................................... 156
    6.3.1 Alert data ............................................................ 157
    6.3.2 Alert Customizer .................................................... 158
  6.4 Creating an alert definition with the Manage Alerts portlet ..... 162
    6.4.1 Script-based alerts ................................................. 163
    6.4.2 Java-based alerts ................................................... 170
    6.4.3 Model-based alerts .................................................. 174

Chapter 7. Annotations .......................................................... 179
  7.1 Overview of annotations ................................................ 180
  7.2 Configuring Annotations persistence .................................. 181
    7.2.1 XML persistence ...................................................... 181
    7.2.2 Relational database persistence .................................. 181
  7.3 Annotating charts and tables ........................................... 183
    7.3.1 Annotations user interface ....................................... 183
    7.3.2 Commenting charts ................................................. 187
    7.3.3 Viewing and responding to chart comments ................... 188
    7.3.4 Commenting tables and table rows .............................. 190
    7.3.5 Viewing and responding to table and table row comments .. 191
  7.4 Enabling annotations ..................................................... 193
    7.4.1 Enabling Annotations in high-level dashboard builders ..... 193
    7.4.2 Using the Annotations builder in a model .................... 195
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5 Configuring alerting for annotations</td>
<td>198</td>
</tr>
<tr>
<td>7.6 Archiving annotations</td>
<td>204</td>
</tr>
<tr>
<td><strong>Appendix A. Additional material</strong></td>
<td></td>
</tr>
<tr>
<td>Locating the Web material</td>
<td>205</td>
</tr>
<tr>
<td>Installing the example application</td>
<td>205</td>
</tr>
<tr>
<td>Downloading</td>
<td>206</td>
</tr>
<tr>
<td>Create database and define data source</td>
<td>206</td>
</tr>
<tr>
<td>Deploy the Lotus Notes database</td>
<td>207</td>
</tr>
<tr>
<td>Creating the projects in WebSphere Dashboard Framework</td>
<td>208</td>
</tr>
<tr>
<td>Import WebSphere Portlet Factory archive</td>
<td>209</td>
</tr>
<tr>
<td>Configure project for Domino</td>
<td>210</td>
</tr>
<tr>
<td>Redeploy project</td>
<td>210</td>
</tr>
<tr>
<td>Populate the database</td>
<td>211</td>
</tr>
<tr>
<td>Configure e-mail alerts</td>
<td>211</td>
</tr>
<tr>
<td>Set up pages and portlet</td>
<td>212</td>
</tr>
<tr>
<td>System requirements for downloading the Web material</td>
<td>212</td>
</tr>
<tr>
<td>How to use the example application</td>
<td>212</td>
</tr>
</tbody>
</table>
Notices

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

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

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

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

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.

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.

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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
Trademarks

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

- Redbooks (logo)®
- ActiveInsight™
- Cloudscape™
- Domino®
- DB2®
- IBM®
- Lotus Notes®
- Lotus®
- Notes®
- Rational®
- Redbooks®
- Sametime®
- WebSphere®
- Workplace™
- Workplace Forms™

The following terms are trademarks of other companies:

- SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.
- Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.
- Snapshot, and the Network Appliance logo are trademarks or registered trademarks of Network Appliance, Inc. in the U.S. and other countries.
- Java, JavaServer, JavaServer Pages, JDBC, JRE, JSP, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- Excel, Microsoft, SQL Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Other company, product, or service names may be trademarks or service marks of others.
Preface

WebSphere® Dashboard Framework is a tool for rapidly building dashboard applications on top of a service-oriented architecture, including WebSphere Portal environments. Developers can quickly and easily leverage their company’s core assets and data by automatically assembling them into custom, high-value dashboards. Portlets created with WebSphere Dashboard Framework are dynamic, robust Java™ 2 Enterprise Edition (J2EE™) applications that react automatically to change. Business users can modify them in real time to meet changing business requirements without requiring any coding, duplicating, or versioning of assets. By eliminating the need to code all of these implementations and their variations, WebSphere Dashboard Framework simplifies the development, deployment, and change management processes.

This IBM® Redpaper provides an overview of the options and techniques for developing applications using the WebSphere Dashboard Framework. We provide an overview of the technology and give you a step-by-step introduction into developing your first WebSphere Dashboard Framework. We then look deeper into the product with the use of an example application based on a fictitious company.

Throughout the IBM Redpaper, we also point out tips and tricks as well as some best practices to help you achieve the best outcome for your organization in the quickest possible time.

The team that wrote this paper

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Cambridge Center.

Philip Monson is a Project Leader at the ITSO Lotus® Center in Cambridge, MA. Phil has been with Lotus/IBM for 17 years, joining the company when the early versions of Notes were rolled out for internal use. He has served in management, technical, and consulting roles in the IT, Sales, and Development organizations.

Sam Alexander is a developer for the WebSphere Portlet Factory development team at IBM, specializing in Portlet Factory’s back-end integration builders. He holds a Master’s degree in Computer Science from Boston University’s Metropolitan College and a Bachelor’s degree in Business Management from the University of North Carolina at Wilmington.

Lea Medhurst is a member of the Lotus Technical Pre-Sales team in Australia. He has been working for IBM for two years, joining IBM after an extensive period at an IBM Business Partner. Lea has been responsible for implementing, administering, and developing applications within the Lotus and WebSphere Brands. Lea originally started developing using Lotus Notes® and has since moved on to J2EE and in particular WebSphere Portal and its associated development tools. Lea has been working with WebSphere Portal since V4.0 and is a co-author of Portalizing Domino Applications, SG24-6466.

Christine Miner is a Senior Consultant at Ascendant Technology with over eight years experience in application design and development, specializing in WebSphere Portal and WebSphere Portlet Factory. Christine has experience across the full life cycle of a project,
including architecture, design, development, implementation, and testing. Recently, Christine became an IBM Certified Application Developer for IBM WebSphere Portlet Factory V6.0. The Web site for Ascendant Technology is:

http://www.atech.com

Thanks to the following people for their contributions to this project:

Rebecca Buisan, Product Line Manager, IBM Software Group
Jason Cornell, Domain Engineer, IBM Software Group
Michael C. Frank, WDF Development Manager, IBM Software Group
Luciano Silva, Chief Programmer - WDF, IBM Software Group
Dong Fei Wu, Software Engineer, IBM Software Group
Fan He, Software Engineer, IBM Software Group
Zhao-hui Nie, Software Engineer, IBM Software Group
Rui Shen, Software Engineer, IBM Software Group
Qing Guo, Software Engineer, IBM Software Group
Liang Liang, Software Engineer, IBM Software Group

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and client satisfaction. As a bonus, you will develop a network of contacts in IBM development labs and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at: ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks® publications in one of the following ways:

- Use the online Contact us review IBM Redbooks publications form found at:
  ibm.com/redbooks
- Send your comments in an e-mail to:
  redbooks@us.ibm.com
Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400
Introduction to IBM WebSphere Dashboard Framework

This chapter provides an overview of WebSphere Dashboard Framework. This chapter introduces you to the features and concepts of the WebSphere Dashboard Framework and discusses the value to the business of implementing the framework. This chapter describes the following topics specifically:

- What is WebSphere Dashboard Framework
- Features of WebSphere Dashboard Framework
- How WebSphere Dashboard Framework fits into the IBM WebSphere Portal family
- The benefits of WebSphere Dashboard Framework to the business
- The example application developed for this IBM Redpaper
- Audience and goals of this IBM Redpaper
1.1 What is WebSphere Dashboard Framework

IBM WebSphere Dashboard Framework is a powerful and flexible tool for rapidly building role-based dashboards on top of a service-oriented architecture (SOA). WebSphere Dashboard Framework augments the capabilities of IBM WebSphere Portlet Factory, adding dashboard-specific features, such as a robust alerting module, high fidelity charting, flexible filtering technology, and a set of dashboard-specific design components called builders.

WebSphere Dashboard Framework provides the tooling that allows for the creation of dashboard applications with very little programming effort. These dashboards can be built and customized very quickly to suit the exacting needs of the business in a fraction of the effort usually required with traditional development techniques while still providing the features and richness to allow the creation of powerful user interfaces. Figure 1-1 on page 3 is an example of an application built with WebSphere Dashboard Framework.
These dashboards can be deployed to a number of application server environments whether they are a Java Specification Request 168 (JSR168) portal server, such as WebSphere Portal Server, or a Java 2 Enterprise Edition (J2EE) compliant server, such as WebSphere Application Server.
1.2 Features of WebSphere Dashboard Framework

WebSphere Dashboard Framework provides the following features.

**Dashboard builders**
The product provides a series of reusable software automation components known as builders. These builders simplify the creation of applications by allowing the developer to specify parameters into a wizard-like interface. The builders automate the most common dashboard design patterns and use cases. When many builders are used together, complex applications can be created. The builders can then be reused or profiled to create many applications in an extremely short amount of time.

An example of a builder is the Web Chart builder that allows the display of several different types of charts upon specification of inputs into the builders. This builder can then be reused many times to create variation or new charts by simply changing the input parameters into the builder.

**Business user configuration and personalization**
Along with the builders that are provided with IBM WebSphere Portlet Factory, it is easy to extend applications to allow business users to change the configuration of applications that are produced. This allows the developer to build great flexibility into the application and provide better value to the user by allowing the user to configure the application to the user’s exact requirements. This feature is possible with the base WebSphere Portlet Factory; however, the WebSphere Dashboard Framework expands this capability by providing additional high-level builders, such as the Table Customizer builder discussed in section 4.3.20, “Table Customizer” on page 116.

For example, using the charting example in Figure 1-1 on page 3, this feature makes it possible to provide the user with the ability to specify whether the user wants to see the data presented in a pie graph or a line graph.

**High fidelity charting**
WebSphere Dashboard Framework enables the creation of dynamic, configurable charts. The product includes support for a wide range of chart types (for example, Pie, Dial/Gauge, Bar, and Line) and file formats (for example, SWF, PNG, JPG, and BMP).

**Alerting**
Alerts are a feature of WebSphere Dashboard Framework that allows for the configuration of real-time events based on specified inputs. This allows business users to take actions quickly based on an event that might be happening within the business. An example of alerting can be a sales manager who has responsibility for numerous salespeople who are dispersed around a geography. If one region within that geography is falling short of its sales target, a threshold might be crossed that triggers an alert that notifies the sales manager so that the sales manager can take immediate action. Instead of hunting through numerous reports, the sales manager is able to take immediate action and correct the issue in real time, leading to a better business outcome.

The first piece of functionality that makes up this feature is the Alerts engine. The engine generates alerts that are based on configuration thresholds that have been set. The second component is the administration portlet that allows the administrator to define and control the alerts. The third component is the My Alerts portlet, which is one of the methods for alerting the user. This portlet is configurable based on role and other user preferences.
Alerts can also be escalated, for example, if a manager has received an alert, but the alert has not been addressed for a predefined amount of time because the manager is out on vacation. The alert can be escalated to additional people.

**Filtering**
Filtering allows the user to see only the data the user wants dependent on a value used to filter that data. An example might be a sales manager who covers many different regions and sees sales data for all of the sales manager’s regions in the sales manager’s dashboard. With WebSphere Dashboard Framework, it is possible to provide the sales manager with a filter that allows the sales manager to select the southern region data and only see data that is pertinent to that region. Filtering works cooperatively with the Dynamic Profiling technology built into WebSphere Dashboard Framework.

**Annotations**
The new Annotations feature in WebSphere Dashboard Framework allows users to create, view, and respond to comments on charts, tables, or table rows. When creating a comment, a user can specify privacy settings, add one or more attachments, and dynamically generate alerts notifying selected users of the comment. All of this can be accomplished by business users within a very intuitive portlet interface. Also, included within the Annotations framework is a portlet that allows administrators to archive or delete comments.

### 1.3 How WebSphere Dashboard Framework fits into the IBM WebSphere Portlet Factory architecture

WebSphere Dashboard Framework is one of a family of products that all accelerate the value of WebSphere Portal. All of these products dramatically decrease the time to market of creating an application while still adhering to strong architectural standards, such as a service-orientated architecture.

IBM Dashboard Accelerator provides flexible tools and a framework for assembling dynamic dashboards and scorecards, linking strategic objectives to real-time performance information. With no additional charge to access a new dashboard key performance indicator (KPI) catalog, clients can easily download portlets and assemble them into dashboard applications. For example, a company looking to quickly deploy a sales dashboard finds relevant portlets, such as Top Opportunities, Customer List, and Customer Satisfaction already developed.

Each product uses the preceding product as a foundation for providing greater functionality as shown in Figure 1-2 on page 6.
1.3.1 IBM WebSphere Portlet Factory

IBM WebSphere Portlet Factory supercharges WebSphere Portal with rapid application development (RAD) tools and technology that accelerate portal deployments by simplifying the creation, customization, maintenance, and deployment process for portlets. WebSphere Portlet Factory’s ease-of-use and advanced development features dramatically streamline the entire portlet development process, enabling developers to deliver adaptive, robust portlets in a fraction of the time and at a fraction of the cost it requires today. WebSphere Portlet Factory is the base product on which the other products are built. Using WebSphere Portlet Factory, it is possible to:

- Quickly create applications that integrate with existing company assets, such as IBM Lotus Domino®, SAP®, PeopleSoft®, Siebel®, and databases to build SOA compliant applications that are reusable.
- Create portlets that are either native WebSphere or JSR168 as well as standalone Web applications and Web services, without requiring coding.
- Improve the time to market for applications by not only speeding up the development process but also using the unique profiling abilities that allow many variations to an application without duplication.

Several of the new features in Version 6.0.1 of the product are:

- A service-oriented architecture (SOA) entry point that helps make it easy to deliver SOA-based portal applications
- New builders to easily integrate IBM Workplace™ Forms, Portal Content Repository, and Directory Searching
- New enhancements that help developer productivity and success, including support for refactoring and simplified deployments
Comprehensive Web 2.0 support, including AJAX and REST

See Figure 1-3 for an example of the WebSphere Portlet Factory tooling. The example below shows generated Java code produced by a builder within a model.

Figure 1-3   Example WebSphere Portlet Factory application in the development tool

1.3.2 WebSphere Dashboard Framework

WebSphere Dashboard Framework is built upon WebSphere Portlet Factory and is the focus of this paper. It has all of the features outlined in section 1.2, “Features of WebSphere Dashboard Framework” on page 4 as well as these new features introduced in the latest version:

- Microsoft® Excel® enhanced integration: Enhancements allow for integration of complex Excel spreadsheets for direct display as dashboard graphs and tables.
- Maps and legends: Easily create geographic maps of data by country, state, and so forth and add or customize legends to maps, graphs, and tables.
- Open Document Spreadsheet (ODS) export: Support provided to export directly from dashboard portlets in ODS.
- Annotations: New collaborative capability supports making a comment on the portlet or a row in the table for review by others.
Figure 1-4 shows an example of a WebSphere Dashboard Framework application. The example shown is the Banking Dashboard KPI, which is available for download from:

http://www.lotus.com/dashboardcatalog

![Figure 1-4](image)

1.3.3 Lotus ActiveInsight

IBM Lotus ActiveInsight™, which includes WebSphere Dashboard Framework, creates a performance management solution that delivers real-time key performance metrics dynamically to users through role-based, actionable, intuitive dashboards and scorecards. It helps improve performance by reaching performance data wherever it resides and empowering users to collaborate and initiate actions to address issues at the point they arise.

Lotus ActiveInsight builds upon the features of WebSphere Dashboard Framework. It provides a prepackaged jump-start dashboard deployment that includes packages for sales and executives. The pre-built solution can then be modified to suit the exact needs of the organization. ActiveInsight has all of the features of WebSphere Dashboard Framework and also:

- Delivers ready to use composite, role-based views of real-time performance information without the need for development
- Provides built-in collaborative services to help rapidly resolve actual or projected plan exceptions
- Reaches performance data wherever it resides while leveraging existing technology investments by offering powerful integration tools and a world-class SOA environment

With these features, ActiveInsight allows an organization to achieve its organizational goals by quickly identifying any problem areas within the organization.
See Figure 1-5 for an example of an ActiveInsight application.

Figure 1-5  An example of ActiveInsight

1.4 The benefit to the business

The ability for an organization to have a clear view of key performance indicators is recognized as a vital contributor to the performance of any organization. Today, many organizations struggle with an overwhelming amount of data from various information systems. Dashboards have been around for a long time. When they were first introduced, dashboards provided users with information culled from numerous disparate systems and presented it in one location. Although useful, the data was static and often became outdated quickly.

1.4.1 Why a dashboard

Businesses are realizing that many dashboards still have limitations, such as static data and an inability to react to and update information in a timely manner. It can be difficult for businesses to find and consolidate information from disparate sources within their organizations and to enable dashboards to provide value beyond the executive level.

For dashboards to have a positive impact on a business, companies must make them active so the dashboard can empower the business to measure things that its users can actually affect. In other words, dashboards need to allow users to view the information that they need in order to make effective business decisions and to take appropriate action.

Through active dashboards, businesses can implement an action plan and assign ownership for each key performance indicator in case that key performance indicator falls out of its
target range. An active dashboard also facilitates collaboration, for example, by enabling users to initiate chat sessions, view participants’ online or offline status, and send e-mail.

1.4.2 IBM Dashboard Accelerator

The IBM Dashboard Accelerator solution addresses the business challenges head-on that we listed in section 1.4.1, “Why a dashboard” on page 9.

With the announcement of IBM Accelerators for WebSphere Portal (see Figure 1-6), IBM offers integrated packages that easily snap onto the Portal to help deliver business value and faster time-to-value by addressing specific project needs.

This includes IBM Dashboard Accelerator and its real-time, active, performance dashboards. The IBM Dashboard Accelerator includes the following components:

- Lotus ActiveInsight
  Software that helps organizations to define and communicate goals, view real-time metrics, and take action through composite, role-based dashboards and scorecards. ActiveInsight includes both a pre-built scorecard application, as well as WebSphere Dashboard Framework RAD tooling.

- IBM WebSphere Dashboard Framework
  Flexible tools and a framework for building dynamic dashboards and scorecards that link strategic objectives to real-time performance information.

- Dashboard KPI Catalog
  Available as a branch of the WebSphere Portal catalog, this component contains portlets that represent industry vertical or horizontal best practice KPIs. With access to this catalog, clients can easily download portlets and then quickly assemble them into meaningful dashboard applications. This catalog is available for no charge to Lotus ActiveInsight and WebSphere Dashboard Framework clients. See:
  
  http://www.lotus.com/dashboardcatalog

![Figure 1-6  IBM Dashboard Accelerator](image)

1.4.3 Why WebSphere Dashboard Framework

WebSphere Dashboard Framework benefits an organization by addressing all of the key points discussed in 1.4.1, “Why a dashboard” on page 9.
Help lower dashboard development and maintenance costs
IBM WebSphere Dashboard Framework is designed to simplify and speed up the entire dashboard development cycle from data integration, to user interface development, to deployment, and all of the iterations in between. By leveraging this framework, you have the potential to help increase developer productivity up to 15 times, dramatically reducing the costs of your dashboard efforts.

Facilitate improved decision-making with consolidated, timely data
With WebSphere Dashboard Framework, in conjunction with IBM WebSphere Portlet Factory, you can easily consolidate data and processes from multiple back-end systems into unified dashboard solutions with a consistent and intuitive interface. Dashboards built with WebSphere Dashboard Framework can access data from both historical and operational data sources, including spreadsheets, data warehouses, enterprise applications (SAP, PeopleSoft, Siebel, and IBM Lotus Domino), and Web services. By having access to information in a timely and dynamic manner, business users can make better informed decisions that can result in improvements in overall business performance.

Speed up problem resolution by enabling action from the dashboard
Unlike traditional dashboards, which are often nothing more than relatively static electronic reports, dashboards powered by WebSphere Dashboard Framework facilitate problem resolution by enabling action right from the dashboard. If the dashboard illuminates a potential problem or opportunity, business users can quickly react without having to pick up the phone or jump to one or more different systems. Supported actions include the ability to collaborate, kick off workflows, update data within back-end systems, and dynamically create new alerts, all from within the dashboard interface.

Help improve employee productivity through tailored information and alerts
With WebSphere Dashboard Framework and WebSphere Portlet Factory technology, your company can dynamically generate highly tailored, role-based dashboards that display the proper functionality, alerts, and data to meet users’ specific needs and requirements. As a result, you can avoid the high maintenance costs and scalability challenges that people typically face when creating customized dashboards by using traditional techniques.

Leverage existing infrastructures with standards-based tooling
WebSphere Dashboard Framework was built from the ground up to support the creation of dashboards that are based on a service-oriented architecture (SOA) and adhere to open standards, including J2EE, XML, and JSR168. In addition, because WebSphere Dashboard Framework provides native support for the market-leading WebSphere Portal, you can quickly deploy dashboards that leverage the powerful services inherent in the portal, including built-in collaboration, workflow, single sign-on, portlet communication, and user group management.

Increase dashboard flexibility by enabling user configuration
WebSphere Dashboard Framework provides the ability to very easily create robust, browser-based configuration wizards that enable nontechnical users to dynamically personalize and configure their dashboard portlets without having to involve your IT department. Users can change any aspect of the portlet that was exposed by IT when setting up the basic configuration, which can include the thresholds, the data fields to display, and the format of the data (for example, the type of chart). This capability to extend portlet configuration to business users helps to increase the flexibility of your dashboards while reducing IT maintenance costs.
1.5 The example application

Throughout this IBM Redpaper, we provide an application that gives you concrete examples that you can reference at any time. See Figure 1-7 on page 13. You can find instructions for how to download and install the example application in Appendix A, “Additional material” on page 205. Chapter 2, “Overview of portlet creation” on page 15 walks you step-by-step through creating your first WebSphere Dashboard Framework application; however, the remaining chapters in the book discuss the features of WebSphere Dashboard Framework in greater depth. The example application in this IBM Redpaper is provided as a reference for these later chapters.

The example application used throughout this IBM Redpaper is based on a fictitious sales organization. The company is a worldwide global sales organization operating in the American, Asian, and European regions. The company has a sales manager for each of the regions as well as a global sales manager. The organization has several business challenges:

- Information contained within different systems
  They have data stored in Domino and DB2® as well as information in Microsoft Excel, which makes it difficult to pull the data together into a coherent set of business data.

- No single view of key sales data
  The sales team often has to refer to different reports from various systems, which occasionally leads to reports being missed in key review meetings.

- No role-based reporting
  Everyone within sales sees all of the data for all of the regions, which makes it difficult for the regional managers to focus on their data only.

- Data is difficult to interpret
  All data is presented in reports that are native to the systems in use. This means that there is no graphical representation of the data, which leads to mistakes interpreting the data.

- No effective way to collaborate on the sales data
  The team is spread out in many time zones and does not have a way to effectively communicate about the sales data within the context that they view it.

- No real-time notification of important issues
  Certain systems produce reports monthly, so the sales force is only alerted to issues once a month. The sales force needs earlier notification in order to act earlier in the sales cycle on these issues.

The organization chose WebSphere Dashboard Framework to address these issues. Throughout Chapters 3 to 7, we explain the key builders that the developers used to create the example application, as shown in Figure 1-7 on page 13, and explain the concepts that were used to solve these business issues.

The example application is available for you to install with all of the source code that we used. It might be beneficial to install the application and refer to it as you read the chapters.
1.5.1 Technology used

We, the authors of this IBM Redpaper, used the following technologies to create the example application:

- WebSphere Dashboard Framework 6.0
- Eclipse Software Developers Kit (SDK) 3.2
- WebSphere Portal Server 6.0.1
- IBM DB2 V9.1
- Lotus Domino 8.0 Beta 2
- Microsoft Excel 2002 Service Pack 3
1.6 Audience and goals of this IBM Redpaper

This IBM Redpaper aims to provide relevant information to a wide variety of users. In the first few chapters, we provide information that is beneficial to an audience that might consider implementing WebSphere Dashboard Framework in their organization. Executives, Line of Business managers, and users can gain a high-level understanding of the features and functionality provided by the framework.

In the later chapters, we provide an in-depth look into specific areas of the technology. These chapters are targeted toward the implementers of the technology. You will discover the technology is an easy to use modeling framework that lends itself toward script developers or people who want to develop applications but do not wish to learn an in-depth programming language. It is also an aim of this IBM Redpaper to appeal to programmers and help them understand the simplicity and power of the product.

After reading this IBM Redpaper, we hope you:
- Understand the features and functionality of WebSphere Dashboard Framework
- Understand the value of a dashboard and the use of a framework
- Are able to use and understand the various dashboard builders
- Get an understanding of best practices for building dashboards with WebSphere Dashboard Framework

1.6.1 Skills required

WebSphere Dashboard Framework is a rapid development environment that is a code generator. The code generated is J2EE compliant, but you do not necessarily need to know J2EE, because WebSphere Dashboard Framework generates the code. The primary skills you need to get the most from this IBM Redpaper are:
- An understanding of application development methodologies
- Familiarity with WebSphere Portal administration

It is beneficial but not mandatory to understand:
- J2EE development
- Eclipse integrated development environments (IDEs)
- Knowledge of relational database concepts
- XML
Overview of portlet creation

This chapter provides information to help you create applications with WebSphere Dashboard Framework. We outline how to get the product installed and then how to create your first portlet with WebSphere Dashboard Framework. We also discuss the various development options and provide guidance about the most productive ways to work with WebSphere Dashboard Framework. This chapter is an introduction to WebSphere Dashboard Framework and is intended for the developer, who is not familiar with the product or Eclipse.

This chapter contains the following sections:
- Installing WebSphere Dashboard Framework
- Creating a WebSphere Dashboard Framework project
- Developing your first portlet using the framework
- Deploying to other environments
2.1 Installing WebSphere Dashboard Framework

This section describes the procedure to install WebSphere Dashboard Framework and its available options.

2.1.1 Planning the installation

There are three options when you install the tooling. The option that you choose depends on your circumstances. The options are:

- Install into Rational® Application Developer
  This is an excellent option if you want to continue using the features that Rational Application Developer offers with WebSphere Dashboard Framework. You can install into either Version 6 or 7 of Rational Application Developer. You can also use Rational Software Architect and Rational Web Developer.

- Install into Eclipse
  Because the tooling for WebSphere Dashboard Framework is a set of Eclipse plug-ins and features, you can install into an existing installation of Eclipse. The supported versions for WebSphere Dashboard Framework 6.0 are Eclipse 3.1 and 3.2 (SDK versions only).

- Install with Eclipse
  Provided with WebSphere Dashboard Framework is Version 3.2 of Eclipse SDK, which allows you to install it along with WebSphere Dashboard Framework.

For the purpose of this IBM Redpaper, we chose to install WebSphere Dashboard Framework along with Eclipse as part of the installation. The rest of this section explains how we performed that installation.

2.1.2 Installing WebSphere Dashboard Framework with Eclipse

The installation process is straightforward:

1. To begin, select install.bat from your installation source. After the installation splash window displays, select your desired language and click OK. Then the Introduction window appears as shown in Figure 2-1 on page 17.
2. Accept the license agreement and click Next. The following window lets you specify the location where you want to install the WebSphere Dashboard Framework as shown in Figure 2-2.

3. Next, select whether you want to use an existing Eclipse installation, existing Rational Application Developer, or install Eclipse along with WebSphere Dashboard Framework. In this example, we install along with Eclipse as shown in Figure 2-3 on page 18. Click Next.
4. If you install with Eclipse, the next window asks you to specify where you want to install Eclipse. After that, a summary window appears. Review that window and select **Install**. After just a few minutes, you get a confirmation window where you can select **Done**.

To launch WebSphere Dashboard Framework, start your Eclipse installation. On a Windows® machine, you start eclipse.exe from wherever you installed Eclipse. After Eclipse is running, click **Window → Open Perspective → Other** and select the **WebSphere Portlet Factory** perspective as shown in Figure 2-4 on page 19.
After the WebSphere Portlet Factory perspective opens, you are ready to work with WebSphere Dashboard Framework.

2.1.3 Notes about installing into Rational Products

**Important:** Occasionally, the WebSphere Dashboard Framework installer does not recognize the installation location of Rational Application Developer. This situation is described in TechNote 1257267 at:


The solution is to instead indicate that you want to install WebSphere Dashboard Framework into an existing Eclipse installation and then point it to the Eclipse instance that Rational Application Developer uses (for example, C:\Program Files\IBM\SDP70).

Another method is to install WebSphere Dashboard Framework with Eclipse and then add the WebSphere Dashboard Framework directory as an extension point in Rational Application Developer. To do this in Rational Application Developer, go to **Help → Software Updates → Manage Configurations**. When you are in the Product Configuration window, click **Add an**
Extension Location and select the Eclipse location that you just installed. In the case shown in Figure 2-5, it is C:\IBM\WPF\WDFDesigner\eclipse.

![Product Configuration](image1.png)

**Figure 2-5  Selecting WebSphere Dashboard Framework as an extension point in Rational Application Developer 7**

## 2.2 WebSphere Dashboard Framework concepts

In this section, we briefly look at the key concepts behind WebSphere Dashboard Framework. WebSphere Dashboard Framework is an extension of WebSphere Portlet Factory; all of the same concepts apply.

### 2.2.1 Accessing Help

The product Help documentation is an excellent place to start. WebSphere Dashboard Framework takes advantage of the Eclipse Help system that is built in the Integrated Development Environment (IDE). To get to the Help from Eclipse, go to **Help → Help Contents** to see a list of books in the left navigator bar.
There are two particular books that are relevant to this product:

- **WebSphere Portlet Factory**
  This book describes all of the key concepts of the products as well as the base builders in the product, such as Domino Data Access builder and the AJAX builders. In the WebSphere Portlet Factory Help book, there is a section Reference → Builder Call Help, which is especially useful for accessing Help regarding the inputs for well over 100 builders.

- **WebSphere Dashboard Framework**
  This book discusses the specific features of WebSphere Dashboard Framework. There are excellent samples and tutorials listed in the Help and the Dashboard builder catalog is especially useful.

Another way to use the Help function is when you are within a builder, press F1 to bring up the Help for that builder in context on the right. This is an excellent way to learn exactly which inputs are required for the builder with which you are currently working.

### 2.2.2 WebSphere Portlet Factory concepts

Portlets created with the WebSphere Portlet Factory follow a standard Java 2 Enterprise Edition (J2EE) model-view-controller design. In order to simplify the task of writing custom portlet Java and JavaServer™ Pages™ (JSP™) code, WebSphere Portlet Factory introduces the developer to a few key objects to use to create a portlet application:

- **Builders**
  A *builder* is the core building block that automates design and development tasks performed by developers. Simply put, a builder is a collection of Java classes and XML documents that represents a specific pattern or high-level component of a Web application. A builder provides a wizard-like user interface (UI) for gathering configuration information from the developer, as well as the code for rendering the pattern or context-aware elements within the Web application. A simple builder might add a button to a JSP page, while another builder might render a Domino view. Builders can analyze the application and perform tasks in the context of what previous builders have created. For example, a page navigation control builder can reference a set of JSP pages and create a set of navigational controls relevant to the context of those pages. If a page changes, then the navigational controls update automatically in a ripple effect that can cascade through the entire application.

- **Models**
  A *model* is a sequenced collection of builders that generates the application components representing the behavior, structure, data, and presentation of the portlet application. Internally, a model is simply an XML file containing a series of calls to builders. Each model can be turned into a portlet or can be run as a standalone J2EE application.

- **Profiles**
  A *profile* contains a set of inputs that varies the way a portlet behaves. Profile settings can be edited after a portlet is deployed by clicking the configuration icon for the portlet. A profile feeds values into builders based on user identity or other contextual information (such as the day of the week). Using profiles, you can automatically generate different variations of a generic portlet (from the same model) for different users or situations. Configured profiles are extremely easy to implement with WebSphere Portlet Factory.

- **Regeneration**
  When a model is regenerated, each builder in the model executes in sequence and creates pieces of your portlet, such as JSP pages or Java methods. During regeneration,
profiles can feed different inputs to builders based on the user or situation, automatically creating custom portlets dynamically or "on the fly." There is a negligible performance impact associated with regeneration. Less than one percent of the total processing resources in a typical "regen-enabled" execution environment are spent performing regeneration, because generated objects are cached at optimal levels of granularity, as are sessions. Figure 2-6 is a representation of WebSphere Dashboard Framework concepts.

![Figure 2-6 Dashboard Framework concepts](image)

### 2.3 Creating a WebSphere Dashboard Framework project

The first step in any WebSphere Dashboard Framework application is to create the project within your chosen IDE. Here again, you have a couple of choices about how you want to set up the testing of the applications created in your IDE. There are a couple of choices depending on your environment:

- **Deploy for testing to a local WebSphere Portal server**
  
  If you have an instance of WebSphere Portal on your local machine, you have the option to deploy and test straight to that instance.

- **Deploy for testing to a remote WebSphere Portal server**
  
  If you do not have the resources to run a WebSphere Portal server locally, you can configure your IDE to use the remote WebSphere Portal by mapping a drive to that server.
The following steps create your first WebSphere Dashboard Framework project using a local WebSphere Portal server as the unit test server. At this point, we keep it simple, but when you become more advanced, you configure your environment for features, such as alerts and annotations. Refer either to the Help or following chapters if you want to set up these features now.

**Tip:** We, the authors of this IBM Redpaper, found it much more productive to test to a local WebSphere Portal environment rather than a remote server.

### 2.3.1 Step one: Setting the project name

First, ensure that you are in the WebSphere Portlet Factory perspective by checking the title bar of your IDE. The title bar shows something similar to WebSphere Portlet Factory - Eclipse SDK. If you are not in that perspective, click **Windows → Open Perspective → Other** and then select **WebSphere Portlet Factory**.

Next, select **File → New → WebSphere Portlet Factory Project**. This launches the project creation wizard. Give your project a name. In the case shown in Figure 2-7 on page 24, we call our project **Sales**.

**Tip:** You can also just right-click in the Project Explorer view and select **New** from there or **Shift+Alt+N** on a Windows machine.
2.3.2 Step two: Selecting the feature set

After you select **Next**, the dialog appears for you to select the feature sets that you want to have in your project. By default, all of the basic WebSphere Portlet Factory builders, which include all of the builders that you require to do tasks, such as SQL calls, are there for you to use.

The dialog allows you to add additional functionality to your project, including:

- **Charting**
  
  WebSphere Dashboard Framework includes a license of the GreenPoint Web Charting engine for producing high fidelity charting. This includes charts such as pie and bar types. You can also select this option under the WebSphere Dashboard Framework category.

- **Dojo Ajax**
  
  These builders automate the task of creating Dojo Ajax components. Dojo Ajax is an open source project that is a framework for creating intuitive and attractive user interfaces. See: [http://www.dojotoolkit.org/](http://www.dojotoolkit.org/)

  This functionality includes builders, such as:
  
  - Dojo Drag and Drop Source and Target builder
– Dojo Enable builder
– Dojo Inline Edit builder
– Dojo Tooltip builder

**Note:** While the Dojo builders are really part of the base WebSphere Portlet Factory product and beyond the scope of this IBM Redpaper, these builders are really useful in creating a fantastic user interface and we encourage you to explore this great functionality.

- **Integration Extensions**
  These integration extensions allow integration into a great number of data sources that you might have within your organization. You can then leverage that data to build components. These builders include:
  – Content Model Extension
  – Excel Extension
  – Lotus Collaboration Extension
  – PeopleSoft Extension
  – SAP Extension
  – Siebel Extension
  – Workplace Forms™ Extension

- **National Language Extension**
  These builders allow you to create applications that can support multiple languages and locales.

- **Tutorials and Samples**
  WebSphere Portlet Factory includes a number of sample applications. This is a great way to learn the functionality that is included within the product. Selecting this feature installs the samples and tutorials as outlined and documented in the product Help.

- **WebSphere Dashboard Framework**
  These features make up the core of the WebSphere Dashboard Framework and include:
  – Alerts Module Example shows you how to incorporate the alerting module.
  – Alerts Scheduler consists of builders to enable the alerting scheduler.
  – Dashboard Framework includes builders that form the basis of the framework. See Chapter 4, “Displaying data” on page 79.
  – GreenPoint Web Charts contain builders to create high fidelity charting.
  – Sample Applications are available, which are documented in the Help files.
  – Tutorial solutions are documented in the Help files.

**Note:** The best practice is to only select the Feature Sets that you require for your application. This helps reduce the size of the Web archive (WAR) file that needs to be deployed and generally makes building and deploying your project much faster.
In the example application that we build in the next section, we import an Excel spreadsheet and develop a graph from that data. As shown in Figure 2-8, the feature sets required are the Excel Integration Extension, GreenPoint Web Chart, and the Dashboard Framework.

![Figure 2-8 Selecting the Feature Sets for the project](image)

### 2.3.3 Step three: Java settings

In this dialog in Figure 2-9 on page 27, you can set any additional Java settings, such as adding any additional JAR files or classpaths that you need. Generally, we find there is nothing to change in this dialog when working with WebSphere Dashboard Framework, which is the case in this example, so simply click **Next**.
2.3.4 Step four: Deployment configuration

There are two parts to this step of the wizard. This step allows you to set up your test environment for this project.

**Application server deployment configuration**

It is possible to create applications using WebSphere Dashboard Framework purely to run on a J2EE server without WebSphere Portal. If you just wanted to deploy to a J2EE server, this section allows that configuration. If you develop applications for a WebSphere Portal environment, it is also a lot quicker during the initial building of an application to test on the application server rather than deploy to a full WebSphere Portal server after each iteration of your development. After you have the component built and then require the features of WebSphere Portal Server, such as inter-portlet communication, you can then test your work on the WebSphere Portal server.

To configure the Application Server, click **Add**. After the New Deployment Configuration window displays, give your configuration a name and description and specify the server type as shown in Figure 2-10 on page 29. For all examples in this IBM Redpaper, we use WebSphere Portal Version 6.0.1 and Version 6.0.17 for the WebSphere Application Server on which the Portal runs.
After we selected **WebSphere Application Server 6.x**, the wizard presents us with a list of configuration inputs:

- **Installed Applications Dir**
  This is the full path to the directory where applications are installed on the application server. For WebSphere Application Server, this is the directory, including the cell name, underneath the profile where your applications are deployed. In our case, which was a standard portal installation, it is: 
  `C:\IBM\WebSphere\profiles\wp_profile\installedApps\IBM-L`

- **Server Host**
  This is the host name where your application server is installed. In our case, it is `localhost`.

- **Server Port**
  This is the port on which your application server is listening. In our case, this is the default port of `10038`.

- **Auto Deploy**
  Because WebSphere Dashboard Framework is an extremely rapid application development environment, it is extremely useful to have the tool automatically deploy your application so that you can see how your changes affect your component immediately. By checking automatic deployment, the tool deploys your changes as soon as you save your component.

- **WebSphere Application Server for deployment**
  With Auto Deploy enabled, this is the WebSphere Application Server where the changes are deployed. In our case, it is `WebSphere_Portal`.

**Tip:** If you have a locally installed WebSphere Portal server, it is useful to select `WebSphere_Portal` instead of `server1`; therefore, you do not have to start `server1` as well as WebSphere Portal on your local machine.

- **Admin User**
  This is the administrator user for your WebSphere Application Server.

- **Admin Password**
  This is the administrator password for your WebSphere Application Server.

After you complete all the inputs and select **Test Server Connection**, ensure that you get the “Test connection succeeded” message. Figure 2-10 on page 29 shows the inputs that we used.
WebSphere Portal server deployment configuration

Configuring the WebSphere Portal server is very similar to the application server. After you click **Add**, the wizard presents you with the New Deployment Configuration window.

The configuration inputs for portal are:

- **Portlet API**
  
  This allows you to create portlets to either the native WebSphere Portal API or the standards-based Java Specification Request (JSR) 168 API. Unless you have a really good reason to use native API, we recommend that you use the Java Standard.

  **Important**: If you plan to use alerting, you must use the WebSphere Portal API for the scheduler project, because it contains features that are required by alerts that the JSR standard does not currently have. See Chapter 6, “Alerts” on page 147 for more details.

- **WP Root**

  This is the path on the file system to the root of your WebSphere Portal server. If you are planning to use a remote portal server, you need to map a drive to the file system on the remote server.

- **Auto Deploy**

  This automatically deploys the changes, which you make to your component, to the WebSphere Portal server. Enabling Auto Deploy can really boost developer productivity.

- **JRE™ Home**

  When WebSphere Dashboard Framework deploys the required components to the WebSphere Portal server, it uses tools, such as XML Access, that you do not see. To do
Building Composite, Role-Based Dashboards with WebSphere Dashboard Framework

To get this successfully, the tooling requires a Java Runtime Environment (JRE). This input needs to point to the bin directory of the JRE that you want to use.

- **Admin URL**
  This is the full URL to the configuration interface of your WebSphere Portal server. Use the default value as a guide and change the server name as needed.

- **Admin User**
  This is the administrative user for the WebSphere Portal server.

- **Admin Password**
  This is the password for the administrative user for the WebSphere Portal server.

After you complete the inputs, click **Test Server Connection** to ensure that you get the “Test connection succeeded” message.

Figure 2-11 displays the inputs that we used in our configuration where the WebSphere Portal server was local to the tooling.

![New Deployment Configuration](image)

**Figure 2-11** Inputs for the WebSphere Portal server configurations

**Important:** To get the most out of the tooling, it is important that the server configuration functions correctly. If you have issues, spending time here to get the issues sorted saves a lot of time and frustration as you move forward with the tool.

### 2.3.5 Step five: Completion

After you click **Finish**, the tooling creates your project. Along the way, it asks for a few inputs.
First, the tooling asks if you want to add the jars to your project. The features that you selected influence how many jars need to be added to your project. Just select **Yes** when presented with the dialog to automatically have the jars added.

Next, the tooling asks if you want to deploy the project now if you selected to automatically deploy. This deploys the necessary artifacts to the WebSphere Application server and the WebSphere Portal server to save time later. To deploy now, just select **Yes**.

After the questions are complete, the readme files display. You can read these readme files and then close them. Now, you are ready to create applications with WebSphere Dashboard Framework.

### 2.4 Developing your first portlet using the framework

After you create a project, the next step is to develop a component. This section guides you through developing your first component with WebSphere Dashboard Framework and helps you become familiar with the tooling.

This application is fairly simple with effective results as displayed in Figure 2-12. We take a common business tool, a Microsoft Excel spreadsheet, and add value to the data contained within the spreadsheet by adding a graph and status indicators to the data. We describe more complex components later in this IBM Redpaper.

![Sales Actuals](image)

**Figure 2-12**  Finished application

#### 2.4.1 Step one: Add a model to the project

The first thing is to create a model, which is a container for holding all of our builders for this component. To add a new model, first ensure that you are in the WebSphere Portlet Factory perspective, then select **File → New → WebSphere Portlet Factory Model**, which starts the model creation wizard. Ensure that your project is highlighted. In our case, it is the Sales project, and click **Next**.
The next step in the model creation wizard gives you the option of starting with preconfigured models to use as starting points. In our case, we start from scratch. So, we highlight the Empty model as shown in Figure 2-13 and click Next.

![Select Model](image)

Figure 2-13   Selecting the Empty model

Next, give your model a name. In our case, we call our model SalesActuals and click Next to create the model.

**Tip:** The name that you give your model must be a valid Java class identifier:

- All names must begin with a letter of the alphabet, an underscore, or a dollar sign ($).
- After the initial letter, the name can contain letters and numbers but no spaces or special characters.
- The name can be any length, but long names are discouraged.
- Names are case sensitive.
- You cannot use a Java keyword.

The tooling warns you if you choose a name that breaks these rules.

### 2.4.2 Step two: Exploring your model and the tooling

If you are in the WebSphere Portlet Factory perspective within your IDE, you have an interface similar to Figure 2-14 on page 33.
There are four areas to the user interface:

- Project Explorer View
- Model Editor View
- Outline View
- Other Views

2.4.3 Project Explorer View

Just like any other perspective, this view is used to navigate around the artifacts that together make up the project. There are many artifacts with which you become familiar, such as properties files and configuration files, but they are out of the scope of this section of the IBM Redpaper.

After you create a model (in 2.4.4, “Step three: Adding the Excel data source to the builder” on page 35), this is the view where you can find and open that model. The default location for your model is `<ProjectName>/WebContent/WEB-INF/models`, so in our case, the location is Sales/WebContent/WEB-INF/models.

Starting with Version 6.0 of WebSphere Dashboard Framework, there is a simple shortcut that you can use to locate models. You can locate models in `<ProjectName>/models`, which makes using the tooling even simpler. You can edit your model in either place, because they are exact copies of each other.
Model Editor View
This is the view where you add the inputs to your builders and see the results of the generation process. At the bottom of the view, you notice four tabs:

► WebApp Tree
  This is where you can view the application and the code that is generated. This interface is informational only and cannot be edited.

► Model XML
  Here you can inspect all of the XML artifacts produced by the model. This interface is informational only and cannot be edited.

► Builder Call Editor
  This is the main interface to the builders that make up the model. Here, you can have multiple builders open and enter the inputs into each of those builders.

► WebApp Diagram
  This is a graphical representation of the page flow of the model. This display can be useful for understanding how the pages relate to each other. This illustration is informational and cannot be edited, but you can double-click on an object to go to the builder that represents that object.

Outline View
The outline view shows the list of builders that you have in this model. This is also the area in which you add builders to the model. Across the top of the view, there are five icons. The two icons on the right are standard minimize and maximize icons. The other three icons are:

► Generate the Current Model button
  This icon allows you at any time to regenerate the code that your model represents. This is useful when the inputs in one builder rely on the output of other builders.

► Add a builder button
  Clicking this icon allows you to add builders to your model.

► Drop-down menu builder button
  This icon provides a list of useful tools for working with your model, such as insert, sorting, and filtering.

Figure 2-15 on page 35 shows the location of these icons.
2.4.4 Step three: Adding the Excel data source to the builder

The first step is to import the Excel spreadsheet that you want to use into the project.

In the Project Explorer View, highlight the Sales\WebContent\WEB-INF directory. Right-click and select Import → Import as shown in Figure 2-16 on page 36.

**Requirement:** The imported Excel workbook must be in the .XLS format. It must have been created using Microsoft Excel Versions 97 through 2003.
In the General category of the select dialog, select the file system and select **Next**. In the next window, browse to the location of your .xls, select it, and click **Finish**.

Now, we are ready to add a builder to the project. Select the add builder button (refer to “Outline View” on page 34) and the Builder Palette displays as shown in Figure 2-17 on page 37.

There are two ways to navigate through the Builder Palette. The first way is the All category, which displays a list of all of the builders available to the project highlighted in the box on the right. If you highlight the first entry on the right and press the E key, you see all of the builders that start with the letter “E”.

The second method is to display a reduced set of builders on the right depending on the category that you choose on the left.
Select the Excel Import builder and click OK.

The builder then shows in the model editor view of your IDE. Now, we need to complete the inputs for this builder:

- Name: Enter SalesSpreadsheet.

**Important:** Do not use spaces in your builder names.

- File to Import: Use the ellipses (...) button at the end of the field. In the choose reference dialog, select choose file, expand the WEB-INF directory, and select the Excel spreadsheet that you just imported.
- Generate Schema: Ensure that On Design Regen is selected.
- Content Selection Method: Select Automatic (let builder find the content).
- Sheet: Select Sheet1.
- Has Header Row: Ensure this is checked.
- Preserve Empty Rows: Ensure this is unchecked.
- Preserve Cell Formatting: Ensure this is unchecked.

At the completion of this step, your builder looks similar to Figure 2-18 on page 38. After you specify all of the inputs, click OK.

**Tip:** At any time within a builder, you can select F1 to get contextual Help.
2.4.5 Step four: Displaying data

To display data from the Excel spreadsheet, we use one of the WebSphere Dashboard Framework builders called the Summary and Drilldown builder. We keep it simple at this stage, but for more information, see Chapter 4, “Displaying data” on page 79.

To add the Summary and Drilldown builder, click the Add builder button from the outline view and select Summary and Drilldown from the list of available builders.

Complete the following inputs.

Under the Properties section of the builder, complete the following inputs. See Figure 2-19 for the completed properties:

- Name: Give your builder a name. In our case, it is displayData.
- Initialization Action: Click the ellipses (…) button at the end of the field, and from the resulting dialog, select DataServices → SalesSpreadsheet → execute.

Next, under the Summary Chart Properties section, we need to tell the builder what to display in the graph. See Figure 2-20 on page 39 for the completed inputs.
Complete the following inputs:

- **Column for X-Axis**: From the drop-down list, select Region.
- **Column Selection**: Not all of the information in the spreadsheet makes sense in a graph, so change this to Include Specific Columns.
- **Column for Y-Axis**: To select the specific columns, click on the field, and a drop-down list appears. Select Forecast_Sales and in the field below, select Actual_Sales.

![Summary Chart Properties](image)

**Figure 2-20  Completing the Summary Chart Properties section**

### 2.4.6 Step five: Running the model

First, you test the model.

**Testing a model**

With just these few inputs, it is now possible to run the model. To test a model, select Run → Run. The Create, manage, and run configurations window appears. Right-click WebSphere Portlet Factory Model and click New configuration. Accept the defaults and select Run. See Figure 2-21 on page 40 for an example.
The model then runs as a standalone application on the configured WebSphere Application server. The next time that you want to run the model, highlight the model and just click **Run** (this assumes that the last project that you ran was the WebSphere Dashboard Framework project).

**Note:** Creating the run configuration is only required the first time.

You see something similar to Figure 2-22 on page 41.
Running a model from the WebSphere Portlet Factory Designer makes it easy to iteratively add or configure a builder and then quickly test the changes to confirm that the change had the intended effect. This action tests most aspects of the application with a few exceptions, such as Click-to-Action or Sametime® awareness, which you must test from within WebSphere Portal.

**Deploying the model as a portlet**

Turning a model into a portlet is a fairly straightforward task. Follow these steps:

1. Simply add the Portlet Adapter builder. Locate the Portlet Adapter builder by selecting Portlet Integration as the Category Name within the Builder Palette dialog and then selecting Portlet Adapter.

   Give the Portlet Adapter builder a name, for example, in our case, SalesActualsPortlet. Give your portlet a title, for example, in our case, it is Sales Actuals. Save the model. See Figure 2-23.

   After you add this builder, you are ready to deploy your model as a WebSphere Portlet. This is quite simple. All of the WebSphere Portlet Factory portlets are actually concrete portlets.
that reference a single abstract portlet. The difference among portlets is simply the model that the concrete portlet references.

2. The next step is to rebuild the WAR. This becomes the WAR that is deployed and is the WAR that was defined earlier upon project creation. To rebuild the portlet WAR, right-click the Web project (in our case, it is called Sales) and select Build Portlet Factory WAR → Build Portlet WAR. This updates the WAR file on the file system in the specified path.

   **Note:** If you have enabled Auto Deploy, building the WAR is only required if your model makes changes to artifacts, such as the portlet.xml. Activities, such as adding a new model to a project, requires that you build the WAR. See the product Help for more details.

3. Using the browser, open the home page for the IBM WebSphere Portal. This URL is similar to:
   
   `http://hostname:10038/wps/portal`

4. Log in to the portal as a portal administrator.

5. Select **Administration** from the Launch drop-down menu and then select **Manage Pages**. Navigate to the page where you want to deploy the portlet and click the **edit** icon.

6. If you specified automatic deployment (Auto Deploy) for your WebSphere Portlet WAR, the rebuild process automatically updates the deployed portlet WAR for you. This means that your Sales Automation portlet has already been exposed within WebSphere Portal, and all that remains for you to do is to add it to a page (see Figure 2-24).

![Figure 2-24  Adding the portlet to the page](image)

7. The portlet looks and functions in much the same way as it did in your test environment as displayed in Figure 2-25 on page 43.
If you have automatic deployment enabled every time that you make a change to your model, simply log out and log back in to WebSphere Portal, and the changes automatically reflect in the portlet.

2.4.7 Step six: Enhancing the model

So far, we have created an extremely simple application but still with business value. If you have clicked Table when testing the application, you might have noticed the formatting is not very good. We can also make the graph a bit more appealing.

Change the Summary and Drilldown builder

The steps to change the Summary and Drilldown builder are:

1. Open the **Summary and Drilldown builder**. Under the section titled Summary Chart Properties, change the **Show Chart as 3D** field to **3D**.

   In the section Action Menu Options, check **Enable Export to Excel** and check **Enable Print Page**. See Figure 2-26 on page 44 for completed inputs.
2. Add a Rich Data Definition builder to the model. Rich Data Definition (RDD) is a powerful builder that lets you select a schema in your model irrespective of any data source behind the schema. For each field in the selected schema, you can specify labels, formatting, and validation just by using a single instance of this builder. We use RDD to quickly change the labels that display in your portlet, therefore, enhancing its look and feel.

Enter the following inputs (leaving all of the other fields as default fields):

Under the properties section:

- **Schema:** SalesSpreadsheet_Schema

In the Data Definition Editor, click on each of the fields and set the following:

- **Forecast_Sales:** Set Label as **Forecast Sales** and set Base Data Definition as **base_Currency**.
- **Actual_Sales:** Set Label as **Actual Sales** and Base Definition as **base_Currency**.
- **Percentage_Achieved_of_Forecast:** Set Label as **Percentage** and Base Data Definition as **base_Currency**.

Save and close the builder.

**Tip:** The Rich Data Definition builder is extremely powerful. We recommend that you explore this builder more thoroughly in the product Help.

3. Rerun your model either by using the Run icon in the IDE, or logging out of WebSphere Portal and logging back in to WebSphere Portal. You see significant differences as shown in Figure 2-27 on page 45.
2.4.8 Step seven: Adding the status indicator

In our example, there is only a small amount of data, but in a real environment, the table might contain many entries. It is easier for the user to identify problems in the actual sales figures against the forecast if the user has an indicator to the problem:

1. Add a Status Indicator builder to your model.
2. Complete the following inputs.

   Under the Properties section:
   - Name: problems
   - Field: This is the name of the field from which you want to drive your status indicator. In our case, it is the Percentage_Achieved_of_Forecast field. Use the ellipses (…) to select:
     \[\text{[displayDataTable]}\text{displayData_table/ExcelContent/Row/Percentage_Achieved_of_Forecast}\].

   Under the Indicator Options section:
   - Site Application: Apply style or background color to table row (repeated items only)
   - Status Options: This is where you give your status a name and specify what happens:
     - First row of the table: Set Status to Actual < Forecasts, set Short Description to Actual is under 80% of Forecast, and set the color to Red.
     - Second row of the table: Set Status to Actual > Forecasts, set Short Description to Actual are greater than Forecast, and set the color to Green.

   Under the Indicator Logic section:
   - Indicator Logic: Compare values to thresholds
   - Thresholds: This is where you specify the logic for the status:
     - First row of the table where status is Actual < Forecasts: Set Comparison to < and set the Threshold to .8.
     - Second row of the table where status is Actual > Forecasts: Set Comparison to > and set the Threshold to 1.

See Figure 2-28 on page 46 for an image of the completed builder.
3. Save and run your model. The table now looks similar to Figure 2-29.
2.4.9 Changing WebSphere Portlet Factory properties

So far, we have seen that this is a very rapid development environment that can create some compelling user interfaces to business data. Hopefully by this stage, you have started to experiment to see how your changes are reflected in the models.

At this stage, you might want to add more capability to the tooling. This is where the properties of the project can help you out.

Right-click on the root of the project. In our case, we selected Sales → Properties → WebSphere Portlet Factory.

Here, you can see where the tool is deploying its artifacts as well as toggling automatic deployment on and off. You can also add features to your project by going to the Feature Info and simply selecting what you want and then clicking OK. See Figure 2-30.

![Figure 2-30 Adding a feature to the project](image)

2.4.10 Troubleshooting

While troubleshooting techniques are really part of WebSphere Portlet Factory and beyond the scope of this IBM Redpaper, it is worthwhile sharing the following information from the Help information:

- Look for builder errors and warning messages in Designer. If there are errors shown in the Problems view of Designer when you edit a model, track these errors down and resolve them, because these errors can result in execution errors.
- Disable builders to temporarily simplify a model and isolate the problem. As in any development environment, simplifying an application to isolate a problem can make
debugging easier. In Designer, you can disable builders by selected them in the Outline view, then right-clicking one of the selected builders, and choosing Disable.

- Write data to system output to look at values during execution. There are a couple of ways to do this in a model:
  - In an Action List, you can use the SystemOut action in the Special category to output values. For WebSphere Portal, this system output is written to the WebSphere Portal server's log/SystemOut.log file.
  - With the Debug Tracing builder, you can see the value of certain data at a particular point during model execution. Specify the Action to trace and for Additional debug output, select the value that you want to see. This output is written to the WEB-INF/logs/debugTracing.txt file of the deployed WAR.

- Look for errors in the event.log file of the deployed WAR. When a runtime error is detected during model execution, the details of the error are logged in the WEB-INF/logs/event.log file. The exception details might help you determine the root cause of the error.

- Run with system tracing to see exactly which actions execute. System tracing can be used to see all of the high-level model actions during execution. System tracing also gives you a quick look at the performance of all of the actions, so that you can often quickly spot a performance issue. You can enable system tracing for a model in the Run configuration dialog on the Tracing tab. The system tracing output is written to the system output file (for WebSphere Portal, this is the server's log/SystemOut.log file).

- Use Java level debugging. To set up Java debugging (with single step, breakpoints, and so on), you can use the remote debugging capabilities of Eclipse. See information in WebSphere Portlet Factory Help for instructions to set Java level debugging up.

2.5 Deploying to other environments

After you develop your application, deploy that application to other servers, such as your integrated test environment or production.

2.5.1 Deploying the simple application

Deploying applications, such as the application that we have developed in this chapter, is straightforward:

1. If you have automatic deployment enabled for your project, you can locate the WAR file that represents your project in the installableApps folder of the portal server. In our case, it is C:\IBM\WebSphere\PortalServer\installableApps\Sales.WAR.

2. You can copy this WAR file to any other servers and deploy it just like any other WAR file.

2.5.2 Deploying more complex applications

In reality, a WebSphere Dashboard Framework project usually consists of a series of portlets at a minimum and commonly a few portal pages. It is worth considering this at the start of the project, because it can save time as well as help you avoid mistakes.

The best tool for creating more complex applications is the WebSphere Portals XML Access. The best XML Access reference is the WebSphere Portal Information Center, which includes several sample XML Access files. If you cannot figure out how to do something, such as capture a specific layout, you can use the IBM WebSphere Portal administration tools to create a page the way that you want it and then use an export XML Access script to see how...
the layout is represented in the script. It can be tempting to use these administration tools to do all of your configuration and then export the configuration to edit, but you do not end up with a very manageable output. As a result, it is better to create the scripts manually. Although you can segment the scripts in any way you want, it is more efficient and manageable if you break the scripts into smaller files (especially because XML is so verbose.)

The types of attributes to consider deploying in this manner are:

- **Portlets:**
  - Permissions
  - Cloned portlets
  - Default configuration data
  - Locale specific information

- **Pages:**
  - Create the hierarchy
  - Page permissions
  - Page and portlet layout
  - Apply any themes or skins that are required
  - Set local information

All of these tasks can be automated if you use tools, such as the Maven Cruise Control plug-in, to control your deployments.

**Note:** WebSphere Dashboard Framework can use profiles and property files to automatically adjust to different environments. See the product Help for more details.

### 2.6 Moving forward with best practices

The application developed in this chapter was very simple, and we did not always adhere to best practices when developing more complex applications. Now that you understand how to get a simple application going, let us look at some best practices moving forward.

#### 2.6.1 WebSphere Portlet Factory best practices

First, these are generic best practices when using WebSphere Portlet Factory:

- Use a service provider/consumer model architecture.

  We recommend developing your dashboard applications using the Service Provider/Service Consumer pattern, which is what we do in our more complex example application later. WebSphere Dashboard Framework is an excellent participant in a service-oriented architecture. Using this method ensures that the same flexibility that applies to an SOA design applies to your WebSphere Dashboard Framework application. We discuss this in more detail in Chapter 3, “Retrieving data” on page 53. The concept of consuming that data makes up part of Chapter 4, “Displaying data” on page 79. Also, see the product Help for WebSphere Portlet Factory and the tutorial about creating data driven portlets for more information.
- Use the highest level builder available for the job.
  Many builders contain the functionality of many of the lower level builders; for example, the Domino View and Form builder contains the same functionality that is contained within the Domino Data Access builder and other builders. It is more efficient to use the high-level builder.
- Keep model size under 50 builders.
  Keeping the model smaller has many benefits, such as faster code regeneration time, and makes debugging your model much simpler. If you find yourself adding many builders, consider investigating the use of the imported model builder.
- Do not write a lot of code in Method builder; use the Linked Java Object (LJO) instead.
  This performs much better as a Method builder. Also, the Linked Java Object keeps your code much tidier and easier to read.
- Use Rich Data Definition to simplify and centralize field formatting, validation, and UI.
  This is an extremely powerful builder, which provides a consistent method of formatting your applications. Also, if you need to change the way that something is formatted, you can change it in one place.
- Use the Comment builder to mark and describe sections of a model.
  This makes it easier for others to understand how you put your model together. It also might remind you in years to come why you did something a certain way.
- Use the Imported Model to include common elements needed by multiple models, such as event declarations.
  This allows you to get reuse and consistency in the applications that you create.
- Use the Localized Resource builder to get automatic localization of resource bundles and imported pages.
  These resource bundles and imported pages are optimized to give the best performance when using different locales.
- Use the Run with system tracing option to examine program flow and to look for performance issues.
  This helps you to identify the cause of a performance issue.

### 2.6.2 WebSphere Dashboard Framework best practices

The best practices when using WebSphere Dashboard Framework are:
- Make the dashboards more than just scorecards.
  Back up your graphs and other visual tools with data.
- Provide customized views.
  If it makes sense, give your users the ability to customize the data exactly how they want to look at it.
- Build in custom indicators and alerts.
  Give the users indicators that make sense to them or even the ability to specify the thresholds themselves.
- Make sure you can drill down into a problem or issue.
  Rather than the user just viewing high-level data, make sure the user can get down to the data that is the root cause of the problem.
- Make them actionable.
  Give the users the business tools that they require to act on any problems that the data uncovers. At the very least, give the users a link to the business tool that they need.

- Manage expectations. Bad performance of the key business objectives is not solved by a dashboard.
  Investing money in a dashboard cannot fix the issues with the business. A dashboard simply highlights the issues in the business; a dashboard cannot fix them.

- Make sure there is consensus over key performance indicators (KPIs).
  Business leaders within the organization might have different opinions about what the correct KPIs are for the business. This not only leads to rework of the dashboard; it can also lead to misinterpretation of the data that is displayed.
Retrieving data

One of the business values of WebSphere Dashboard Framework is the ability to rapidly create dashboard applications that present valuable, relevant information. But what about the data driving these dashboard applications? This chapter answers that question.

This chapter explores the following topics:

- Creating service provider models using WebSphere Portlet Factory's rich set of integration builders to retrieve data from DB2, Lotus Domino, and Microsoft Excel
- Best practices for creating provider models
3.1 Retrieving data from relational databases

It is a common use case to build dashboard applications that are driven by data that is stored in a relational database. To illustrate this use case, we developed several provider models to retrieve this data from DB2 so that it can be used by consumer models containing dashboard builders.

In the following sections, we present an overview of our database and configuration. We also show you how we built our OfficeSalesKPIsProvider.model, which uses Portlet Factory’s SQL builders to retrieve data from DB2.

3.1.1 Creating and configuring the database

WebSphere Dashboard Framework includes an SQL script that creates various tables with sample data in a relational database. For simplicity, we used these scripts to create the sample data that we used in parts of our application.

After creating our database using the DB2 Command Center, we used the DB2 Command Editor to run the SQL script, sampleapp_schema.sql, which is located in the WEB-INF/solutions/dashboard/common/nodeploy/data folder. Figure 3-1 shows several of the tables that we created that contain fictitious data for employees, offices, sales key performance indicators (KPIs), customers, and customer orders.

After our database was configured, we configure a data source for the application server.

3.1.2 Configuring the JDBC provider and data source

To make our database available to our dashboard application, we created a new JDBC™ provider and data source using the WebSphere Administrative Console. Next, we briefly describe our configuration for WebSphere Portal Server 6.x and the underlying WebSphere
Application Server Version 6. If you use 5.x, refer to the product Help for the procedure to create a data source for that platform.

Open the WebSphere Application Server administration console. With WebSphere Portal running, open the WebSphere Application Server administration console using the following Web site:

https://<fully_qualified_domain_name>:10039/admin

Log in to the administration console with your WebSphere Application Server administrative user ID and password.

Creating the J2C Authentication Data

After you log in, navigate to the console. The first step is to create a user credential for our database user. Select Security → Global security, and then select Authentication → JAAS Configuration → J2C Authentication data. See Figure 3-2.

![Figure 3-2 Entering the J2C Authentication data](image)

In Figure 3-3 on page 56, click New and enter a name for your data in the Alias field. Enter the User ID and Password and then press OK.
Building Composite, Role-Based Dashboards with WebSphere Dashboard Framework

Figure 3-3  Defining the J2C authentication data

Be sure to save your changes to the master configuration.

Creating the JDBC provider

From the navigation menu, we selected Resources → JDBC providers → New. We set the appropriate values for DB2 as shown in Figure 3-4 and clicked Next.

Figure 3-4  Creating a new JDBC provider for DB2

In Figure 3-5 on page 57, we configured the Class path field to the path of the SQL resource provider classes in our environment. We also set the Native library path field to the path of our SQL resource provider native libraries in our environment.
Creating the data source

After we configured the JDBC provider, we created a new data source. As shown in Figure 3-6 on page 58, we named the data source and provided a JNDI name.

Tip: For the class paths, we used C:\IBM\SQLIB\java, but the best practice for WebSphere Application Server is to create a WebSphere variable for the path to the DB2 JDBC driver rather than hardcoding it into the JDBC providers page. Refer to the WebSphere Application Server Information Center for more details.
Figure 3-6  Specifying the JNDI name and other properties for the new data source

**Important:** In theory, you can use whatever JNDI name that you want, but all of the sample models that come with WebSphere Dashboard Framework use this JNDI name in their builders, so it is much simpler to use this JNDI name.

In Figure 3-7 on page 59, for the Component-managed authentication alias, we selected the J2C Authentication data authentication entry that we had created earlier.

And last, we specified our Database name and Server name.
Chapter 3. Retrieving data

Now that we have completed the configuration, we create our provider models to interact with the SQL data.

### 3.1.3 SQL DataSource builder

We used an SQL DataSource builder in our models to obtain database connections through JNDI. As shown in Figure 3-8 on page 60, we clicked **Fetch DataSource Names** to populate the DataSource Name input with a list of DataSources from which to choose. Then, we selected the DataSource that we created in 3.1.2, “Configuring the JDBC provider and data source”.

In 3.1.4, “SQL Call builder”, we reference this DataSource builder in the SQL Call builder.
3.1.4 SQL Call builder

Key performance indicator (KPI) data is stored in the SA_SALEKPI3 table in the database. Figure 3-9 shows a sample of the data that is stored in this table.

<table>
<thead>
<tr>
<th>OFFICE_ID</th>
<th>MARKET</th>
<th>TARGET_KPI</th>
<th>ACTUAL_KPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Revenue</td>
<td>2,200,000</td>
<td>1,000,000</td>
</tr>
<tr>
<td></td>
<td>Profit</td>
<td>480,000</td>
<td>380,000</td>
</tr>
<tr>
<td></td>
<td>Average Order Size</td>
<td>27,000</td>
<td>27,000</td>
</tr>
<tr>
<td></td>
<td>New Customers</td>
<td>122</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Market Share</td>
<td>0.12</td>
<td>0.15</td>
</tr>
<tr>
<td>1001</td>
<td>Revenue</td>
<td>2,900,000</td>
<td>2,910,000</td>
</tr>
<tr>
<td></td>
<td>Profit</td>
<td>426,000</td>
<td>480,000</td>
</tr>
<tr>
<td></td>
<td>Average Order Size</td>
<td>32,000</td>
<td>32,000</td>
</tr>
<tr>
<td></td>
<td>New Customers</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>Market Share</td>
<td>0.05</td>
<td>0.1</td>
</tr>
<tr>
<td>2002</td>
<td>Revenue</td>
<td>1,800,000</td>
<td>1,780,000</td>
</tr>
<tr>
<td></td>
<td>Profit</td>
<td>300,000</td>
<td>460,000</td>
</tr>
<tr>
<td></td>
<td>Average Order Size</td>
<td>22,000</td>
<td>30,000</td>
</tr>
<tr>
<td></td>
<td>New Customers</td>
<td>110</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Market Share</td>
<td>0.13</td>
<td>0.12</td>
</tr>
<tr>
<td>2010</td>
<td>Revenue</td>
<td>2,000,000</td>
<td>1,900,000</td>
</tr>
<tr>
<td></td>
<td>Profit</td>
<td>370,000</td>
<td>430,000</td>
</tr>
<tr>
<td></td>
<td>Average Order Size</td>
<td>22,000</td>
<td>20,000</td>
</tr>
<tr>
<td></td>
<td>New Customers</td>
<td>95</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Market Share</td>
<td>0.12</td>
<td>0.1</td>
</tr>
<tr>
<td>2001</td>
<td>Revenue</td>
<td>1,500,000</td>
<td>500,000</td>
</tr>
<tr>
<td></td>
<td>Profit</td>
<td>320,000</td>
<td>230,000</td>
</tr>
<tr>
<td></td>
<td>Average Order Size</td>
<td>22,000</td>
<td>25,000</td>
</tr>
<tr>
<td></td>
<td>New Customers</td>
<td>95</td>
<td>102</td>
</tr>
</tbody>
</table>

We only needed to retrieve the Revenue KPI for a specific office. Therefore, we needed to connect to the database and execute an SQL select statement. To accomplish this task, we used Portlet Factory's SQL Call builder.

The SQL Call builder allows you to build SQL statements to run in your application. If the SQL statement returns results, the builder allows you to transform the results into XML. The builder also contains a tool to help you write your SQL.
The following sections show how we specified a JDBC data source and utilized the Database Explorer tool to build our SQL.

**Specifying a data source**
In addition to the name of the builder, the first required input is the SQL DataSource, which allows you to select the JDBC data source to use when executing the SQL. As shown in Figure 3-10, clicking **Fetch DataSource Names** automatically populates this list with a list of the data sources, which are defined on the application server, from which you can choose.

![Figure 3-10](image)

Figure 3-10  Clicking Fetch DataSource Names populates the SQL DataSource drop-down list

Alternatively, you can override this input by specifying a SQL DataSource builder for the SQL DataSource input in the Builder Override Definition section, as shown in Figure 3-11. This is what we did.

![Figure 3-11](image)

Figure 3-11  Overriding the SQL DataSource in the SQL Call builder

As shown in Figure 3-12, when SQDataSource input in the Builder Override Definition section contains a value, then the SQL DataSource input at the top of the builder changes to a read-only prompt indicating that the data source has been overridden.

![Figure 3-12](image)

Figure 3-12  SQL DataSource has been overridden in the Builder Override Definition section

After the data source was configured, we began constructing our SQL Statement using the SQL Call builder's Database Explorer.

**Using the Database Explorer**
To assist us in writing the SQL, we used the Database Explorer section in the builder. The Database Explorer section in the builder is a useful tool for locating tables or stored procedures and automatically generating a base SQL that you can modify.
To locate the table that we wanted, we used Schema Pattern and Entity Pattern inputs to filter the search results for available tables. As the Help describes, both inputs support wildcard characters. For example, all of the tables in our database use the DEMO schema. Also, there are several tables pertaining to customers, each of which starts with the string SA_CUSTOMER. We entered DEMO in the Schema Pattern input and SA_CUST% in the Entity Pattern table, and then we clicked Fetch Entity Names. The Entity Name drop-down list then displayed all of the table names that begin with SA_CUST. As Figure 3-13 shows, four tables were identified that begin with SA_CUST.

You can also use the Database Explorer to generate sample Select, Insert, Update, or Delete statements for the selected table. You can then customize this base SQL before you use it in the builder. Using Database Explorer this way can be extremely helpful, especially if your column names are hard to remember or hard to spell.

For example, in the OfficeSalesKPIProvider provider model, we needed to generate a Select statement that, when given an office id, retrieves only Revenue KPIs for the office. We also needed to perform a join to retrieve the city name from another table and add new columns to the results. Because this is a Select operation, we chose Select for Statement Type. Then, we clicked Generate Sample SQL. The builder generated a base select statement as shown in Figure 3-14.

Tip: Use the Database Explorer section of the builder to help you:

- Easily locate tables and stored procedures
- Automatically generate sample, base SQL statements that you can use as a starting point

![Database Explorer](image)
As shown in Figure 3-15, in the Sample SQL text area, we modified the base SQL to perform a join to retrieve the office city and created the new calculated columns: ONTARGET and BELOW. We also added a WHERE clause to retrieve only Revenue KPI rows and constrained the results based on an Office ID parameter.

![Sample SQL](image)

Figure 3-15  The Sample SQL field can be used to modify the generated SQL

After we completed our SQL, we clicked **Apply Sample SQL**, which copied the SQL to the SQL Statement text area. As shown in Figure 3-16, the builder automatically created an entry in the Parameters table for OFFICE_ID, along with a Web app variable that can be used to set the parameter.

![SQL Statement](image)

Figure 3-16  Clicking Apply Sample SQL in the Database Explorer section copies your SQL to the SQL Statement input and automatically creates any input parameters

### 3.2 OfficeSalesKPIS service definition

We used the Service Definition builder to define a service called **OfficeSalesKPIS**.

To later test the services of this provider, we enabled **Add Testing Support** and **Generate Main** in the Testing Support section as shown in Figure 3-17.

![Testing Support](image)

Figure 3-17  Enabling testing support in the Service Definition builder

To see how to test your service operations, see 3.6.1, “Testing provider models”.
3.2.1 getKPIByOffice service operation

We then added a Service Operation builder to our provider model, as shown in Figure 3-18. We named our operation getKPIByOffice. The Service Operation calls the execute action of the SQL Call data service that we created earlier.

For Operation Inputs and Operation Results, we selected **Use structure from called action**, which instructs the builder to automatically create an input variable (an office ID) and an output variable.

Because we enabled testing support in the model’s Service Definition builder, we can easily verify the results of this service operation. For more information, see 3.6.1, “Testing provider models”.

3.3 Retrieving data from Lotus Domino

In our dashboard application, customer order data is stored in a Lotus Domino database. This section describes how we created a Service Provider model, SAOrdersProvider.model, to retrieve customer order information from Domino.

As a general overview, we used Portlet Factory’s Domino Data Access integration builder to access the Domino data. We created a service definition and a service operation called getOrdersByCustomerId, which accepts a single argument of the customer ID. This customer ID is used to perform a field level search on the Domino view only for orders for the supplied customer ID. Finally, order data is returned to the caller. Figure 3-19 on page 65 shows the builder calls that we used in our model.
We describe in detail how we retrieve customer order information from Domino in the following sections. We begin by briefly describing our Domino database and configuration.

### 3.3.1 Domino database for customer orders

To demonstrate integration with Lotus Domino, we created a Domino database to store customer order information for our application. As seen in Figure 3-20, each order has a unique order ID, customer ID, status, date ordered, quantity, and amount.

![Orders view displaying order information for all customers](image)

Our provider model performs field level searches to find orders for a specific customer ID. To support field level searches in the database, the database is full-text indexed.

After we created the database, we configured the Domino server and database to be accessible to our provider model.

### 3.3.2 Configuring your Domino server and Domino database

The Domino integration builders use the Domino Java APIs, which communicate with Domino by using Common Object Request Broker Architecture (CORBA). Therefore, your Domino server must be running the Domino Internet Inter-ORB Protocol (DIIOP) task. Also, the connecting user must have access control list (ACL) access permissions to the database.

For additional information on configuration, see the Help file for the Domino Data Access and Domino View & Form builders.

At this point, we are ready to create the provider model.
3.3.3 Domino Data Access builder

First, we added a Domino Data Access builder to our provider model, which contains data services for reading view data, creating documents, retrieving documents, updating documents, and deleting documents. These services define variables for their inputs and outputs.

To build schemas at design time and run time, and retrieve data at run time, the builder requires connection and credential information, including the server name, user name, and password.

This information can be supplied in several ways, which is described in the builder Help. For our application, we created a properties file based on the example provided in `WEB-INF\config\domino_config\default_domino_server.properties`. Example 3-1 provides the contents of our properties file.

For the ServerName property, we specified the fully qualified host name of the machine where Domino is running. We also appended the DIIOP port number so that the interoperable object reference (IOR) can be retrieved. We then specified a user name and password for accessing the database.

**Example 3-1 Properties file specifying Domino connection and credential information**

```
ServerName=M51E.cam.itso.ibm.com:63148
UserName=System Admin
Password=xxxxx
```

Then, we selected this properties file for the Properties File input. For Runtime Credentials, we chose to use the same information that is specified in the properties file, as shown in Figure 3-21.

With connection and credential information configured, we clicked Get Databases and views to populate the Database Name input with a list of choices and then selected our database, `SA_Orders.nsf`.

Next, as shown in Figure 3-22 on page 67, we checked Enable View Support input and selected the Orders view from the Domino database.
As shown in Figure 3-20 on page 65, the Domino view contains orders for all customers. However, our application only needs orders for a specific customer ID. Therefore, we used the Search String input to perform a field level search on the view. We set the Search String input to a variable; in the following sections, we show how we build the search string and populate this variable.

![Domino Data Access](image)

Figure 3-22  Specifying the View Name and Search String inputs in the Domino Data Access builder

### 3.4 getSearchString method

In the previous section, we set the Search String to a variable, which needs to be set to a valid Domino search string. Our search string needs to perform a field level search on the view for only documents where the value for the customerId field in the Domino document matches a specific customer ID. So, we added a getSearchString method builder shown in Figure 3-23 on page 68.
Method builder builds a search string to the string that can be used to search on a particular field.

The Method builder accepts a customer ID as a string argument. Then, it builds and returns a Domino field level search string.

For example, if using customer ID 201, getSearchString returns (FIELD customerID=201).

This method is used by the getOrdersByCustomerID action list, which we describe in the next section.

3.4.1 getOrdersByCustomerID action list

We then created an action list to retrieve only the orders for a particular customer, as shown in Figure 3-24 on page 69.

Accepting a single argument of a customer ID, the action list first calls the getSearchString method to set the searchString variable. It then calls the Domino Data Access builder's readTable data service. Finally, the action list returns the results of the readTable service call.
Figure 3-24  An Action List to set the search string variable, call the Domino Data Access builder’s readTable service operation, and return the results

This action list becomes the target action of our service operation described in 3.4.3, “getOrdersByCustomerId service operation”.

3.4.2 OrdersService service definition

To create a data service that can be used by the consumers in our dashboard application, we added a Service Definition builder to our provider model and named it OrdersService as shown in Figure 3-25.

Figure 3-25  The Service Definition builder UI

We knew that we were going to add a service operation to retrieve customer order records from Domino (discussed in 3.4.3, “getOrdersByCustomerId service operation”). And, we wanted the ability to test the service operations before using them in the consumer application. Therefore, we enabled Add Testing Support and Generate Main in the Testing Support section of the Service Definition builder as shown in Figure 3-26 on page 70.
To see how to test your service operations, see section 3.6.1, “Testing provider models”.

### 3.4.3 getOrdersByCustomerId service operation

After we configured the Service Definition builder, we added a Service Operation builder called getOrdersByCustomerId as shown in Figure 3-27.

Because this service operation is part of the service definition that we have just created, we selected OrdersService for the Data Service input.

For the Action To Call input, we selected getOrdersByCustomerIDAction from the list described in 3.4.1, “getOrdersByCustomerID action list”.

The Operation Inputs and Operation Results sections, which are shown in Figure 3-28, allow you to specify how to handle the service operation input and output.
Specifying the Operation Inputs
Because our action list accepts a single string argument of the customer ID, we selected Use structure from called action for the Input Structure Handling input in Figure 3-28 on page 70. When this option is selected, Portlet Factory automatically creates input variables that allow consumer models to set the operation inputs.

Specifying the Operation Results
If the Action To Call input is set to Data Service, you can simply specify Use structure from called action for the Result Structure Handling input, and the builder automatically determines the output's schema type. However, when the Action to Call is an action list or method, as it is in our case, you must select Specify result schema. In other words, in the Result Schema builder input, we need to manually select a schema to use for our operation’s XML result variable.

In 3.4.1, “getOrdersByCustomerID action list”, remember that the action list called the Domino Data Access builder’s readTable data service, which stores results in a schema-typed XML variable. The action list then returns this results variable. So, we needed to determine the schema used for this results variable so that we can select it for the Result Schema input.

We used the WebApp Tree tab to determine the schema used for the readTable results variable. As indicated by the arrows in Figure 3-29, we selected the data service created by the Domino Data Access builder. Then, in the right pane, we noted that readTable operation creates and stores its results in a variable called SA_Order_results.

![Figure 3-29](image)

Figure 3-29  Selecting a Data Service from the WebApp Tree displays the operations, their input variables, and their output variables

Then, in the WebApp Tree’s variable section, we selected the SA_Order_results variable. In the right pane, we see that the Schema Element is ViewData. This is shown in Figure 3-30 on page 72.
Accordingly, in Figure 3-28 on page 70 in the Results Schema input, we select SA_ORDER/ViewData.

At this point, we had a working provider model to return customer order data from Domino.

### 3.5 Retrieving data from Excel

In our dashboard application, sales opportunity data is stored in a Microsoft Excel spreadsheet as shown in Figure 3-31.

![Excel spreadsheet containing Sales Opportunity data](image)

This section illustrates how we created a provider model, SalesOpportunities.model, to serve sales opportunity data to our consumer models. Figure 3-32, "Builder call list for SalesOpportunities.model" shows the builder call list.
Like the previous provider models, this model uses the Service Definition and Service Operation builders. The Service Definition builder defines a service called SalesOpps. The Service Operation defines a data service named getSalesOpps. When called by a consumer, getSalesOpps returns an XML representation of spreadsheet data.

The Service Operation calls an action from the Excel Import builder, which is described in section 3.5.1, “Excel Import”.

3.5.1 Excel Import

The Excel Import allows you to import the content of an Excel workbook into your application. Similar to other integration builders, the builder transforms the data into a schema-typed XML document and stores the XML data in a variable.

Before using the builder, the Excel file must be copied into your project. Although there is no restriction about the location of the file in the project, we recommend that you store the file in a subfolder of WEB-INF, so that it can be accessed in a controlled manner.

As shown in the builder UI in Figure 3-33 on page 74, after the Excel file is copied into the project, it can be selected in the File To Import input.

The Content Selection Method input allows you to specify which content from the workbook to import. Selecting Automatic, as we did in our application, selects all cells in the smallest rectangular range for all defined content from the workbook sheet specified in the Sheet input. Alternatively, this input allows you to manually specify a range of cells or specify a named range from the workbook.
When using the Excel Import builder, remember these restrictions:

- Rich text is not supported. Importing a spreadsheet with cells that contain rich text causes a runtime exception.
- Cells that contain Boolean formulas are not supported.

**Performance considerations**

The Excel Import builder is not designed for accessing spreadsheets containing large datasets. If your data set is significantly large, we recommend migrating your data to a relational database and then using the SQL integration builders.

Also, we recommend enabling caching in the getSalesOpps service operation. Caching is explained in 3.6.2, “Caching”.

**Tip:** For performance reasons, do not use Excel Import to access spreadsheets containing large data sets. Instead, consider migrating the data to a relational database and use the SQL integration builders.

### 3.6 Provider model best practices

This section presents several best practices that we used when developing our provider models.

#### 3.6.1 Testing provider models

As we did in our application, we recommend that you enable testing support in your provider model's Service Definition builder, as shown in Figure 3-34 on page 75.
Selecting **Add Testing Support** creates several test pages:

- An index page that allows you to select a service operation to test
- A input page that allows you to enter any inputs for the service operation
- A view page that allows you to view the results of the service call

Enabling **Generate Main** generates a main method that allows you to run the provider model to test your service operations.

For example, Figure 3-35 shows how we tested the `getOrdersByCustomerId` service operation (discussed in 3.4.3, “`getOrdersByCustomerId` service operation”). Running the model presents the index page with each defined service operation. After selecting the `getOrdersByCustomerId` service operation, the input page allows us to specify our input to the operation. When you click **Submit Query**, the view page shows us that our service returned the expected results.
3.6.2 Caching

To improve application performance, we recommend caching service calls and caching the schema.

Caching service calls

The Service Operation builder allows you to cache the results of the service call, as shown in Figure 3-36. When this setting is enabled, all of the users in a specific profile who run the model's action receive the cached output. The Refresh Interval (secs) allows you to set the cache expiration period.

![Figure 3-36 Enabling caching in the Service Operation builder](image)

Caching is appropriate for data that changes infrequently. For example, our OfficeSalesKPISProvider model contains a Service Operation getKPIByOffice, which uses SQL Call builder to fetch data from DB2. Because this data changes infrequently, it is not necessary to make a trip to the database each time that the operation is called.

This input uses the functionality of the Cache Control builder. For additional information about caching the output of actions, see the Cache Control builder Help.

Tip: For data that changes infrequently, caching a service call or action that returns data can improve performance.

Schema caching

Many integration builders contain an input that allows you to cache the schemas generated by the builder. When enabled, the builder uses a cached copy of the schema rather than rebuilding it. This can improve regeneration performance at design time when editing the model.

For example, the Domino Data Access builder generates a schema for the selected Domino view and form. In our Domino database that contains the customer order information, we knew these Domino elements do not change. Therefore, we selected **Enabled Schema Caching** input, shown in Figure 3-37, to improve regeneration performance.

![Figure 3-37 Many integration builders allow you to cache the generated schemas](image)

Tip: For improved performance, enable schema caching when the schema remains static.
3.6.3 Data manipulation

Portlet Factory allows you to use transform builders or the IXml interface to perform transformations on XML data. For example, after retrieving results from a data service, a transform builder or an LJO method using the IXml interface can be used to manipulate the data.

For improved performance with large data sets, we recommend performing data transformation at the data source level when possible.

This is illustrated in the provider model OfficeSalesKPISProvider.model located in models/data. In SQL Call builder, our SQL select statement, shown in Figure 3-38, is responsible for creating and calculating the value of the ONTARGET and BELOW columns.

```sql
select
    A.OFFICE_CITY,
    A.TARGET_KPI,
    A.ACTUAL_KPI,
    (A.TARGET_KPI * .8) as ONTARGET,
    (A.TARGET_KPI * .7) as BELOW
from
    DEMO.SA_SALEKPI A, DEMO.SA_SALEOFFICES B
where
    A.MATRIX like 'Revenue' and A.OFFICE_ID=B.OFFICE_ID and A.OFFICE_ID = ?
```

*Figure 3-38  Calculating ONTARGET and BELOW fields at the data source level using SQL*

**Note:** For large data sets, perform data manipulation at the data source level when possible.
Displaying data

This chapter introduces you to all of the builders of the WebSphere Dashboard Framework. It also briefly discusses best practices to consider before and during development. This chapter describes:

- Best practices for displaying data
- Why WebSphere Dashboard Factory enhances WebSphere Portlet Factory
- Customizing the look and feel of your portlets
4.1 Displaying data

By now, you are familiar with how to get data. For more information regarding getting data, refer to 3.1, “Retrieving data from relational databases” on page 54. This chapter discusses displaying the data using the builders offered by WebSphere Dashboard Framework. Of course, additional builders, such as Service Consumers and Portlet Adapters, need to be used when creating your consumer model. These additional builders are provided as part of the base product, WebSphere Portlet Factory. The Service Consumer builder is integral to creating the models that we used in our example application. Our models consume data from DB2, Domino, and Excel service providers.

The example application consists of the following portlets and the WebSphere Dashboard Framework builders:

- **Offices Query Filter:**
  - Query Filter Form
  - Query Filter

- **Sales Opportunity for Selected Office:**
  - Query Filter Observer
  - Record List and Detail
  - Status Indicator

- **Office Sales Key Performance Indicators (KPIs):**
  - Status Page
  - Query Filter Observer

- **Customer Orders:**
  - Table Customizer
  - Query Filter Observer
  - Summary and Drilldown
  - ODS Export
  - Contextual Information

- **Office Sales KPI:**
  - Bullet Graph
  - Query Filter Observer

- **Sales KPIs:**
  - Map
  - Status Indicator
  - Status Indicator Legend
  - Snapshot™ Report
  - Summary Row

- **Customer Details:**
  - Record List and Detail
  - Contextual Information

- **Sales Hierarchy:**
  - Hierarchy Drilldown

- **Customer Sales:**
  - Summary and Drilldown
Figure 4-1 and Figure 4-2 on page 82 are screen captures of our completed example application.
4.2 Best practices for displaying data

Before you display data, take these best practices into consideration. We discuss best practices in detail in 2.6, “Moving forward with best practices” on page 49. These are general best practices to consider before and during development:

- Naming Standards: Define naming standards to ensure consistency across your application.
- Use high-level builders that incorporate the functionality of many builders, such as:
  - Summary and Drilldown
– Map
– Hierarchy Drilldown
– Record List and Detail
– Status Page

Use builders to format your model, such as:
– Comment builder: Organize your builders in the outline pallet by using comment builders.
– Rich Data Definition builder: Add formatting to your data, such as currency and data formatting.
– Data Column Modifier builder: Add formatting to your table by making column headers user friendly, enabling sorting of columns, enabling paging, and setting messages for empty data behavior.

4.3 Why WebSphere Dashboard Framework enhances WebSphere Portlet Factory

WebSphere Dashboard Framework enhances WebSphere Portlet Factory, because it offers you a handful of additional specialized builders designed to facilitate the creation of dashboard portlets. These builders automate the most common dashboard design patterns and use cases. A builder is a reusable component that generates Java and Java Server Pages (JSP) code based on the context in which it is called. A builder might add a status indicator to a page, link up all of the fields in a form to an XML variable, or help you to create or consume a Web service. Think of a builder as a "virtual programmer" that can dynamically write code. A builder can write different code in different situations or based on different profiles.

4.3.1 Alert Customizer

The Alert Customizer builder exposes a user's alert notification threshold and subscription settings, so that they can be easily customized and saved. We discuss this in detail in Chapter 6, “Alerts” on page 147.

4.3.2 Alert Data

The Alert Data builder enables you to identify data in a Service Provider model and make it available for use in alerts. This is discussed in detail in Chapter 6, “Alerts” on page 147.

4.3.3 Annotations

The Annotations builder allows users to add comments, responses, and file attachments to charts and tables. This is discussed in detail in Chapter 7, “Annotations” on page 179.

4.3.4 Bullet Graph

The Bullet Graph builder is used to add a bullet graph to a page to visually display target and actual values for a goal. This builder also features animation of the graph.
In the Graph Style section, you can turn on animation of the bullet graph. Select the Use Animation input to display the graph in slow motion from left to right for greater visual interest. The default is to not use animation, which shows the graph as a complete image only. Figure 4-3 is an example of this input.

![Figure 4-3  Bullet Graph builder: Graph Style](image)

In the Graph Information section, you can specify information, such as the Title and Description of the bullet graph. Figure 4-4 is an example of these inputs.

![Figure 4-4  Bullet Graph builder: Graph Information](image)

In the Graph Data section, you define the list of status names, their colors, and range values. Also, the Current Value and the Target Value are defined here. All of the inputs in this section are required. Figure 4-5 is an example of the inputs for the Graph Data section.

![Figure 4-5  Bullet Graph builder: Graph Data](image)
In the Graph Customization section, you can customize the scale level and tick attributes. The two available scale types are Percentage and Number. If the Number option is selected, an additional input Append Unit is displayed. Figure 4-6 is an example of these inputs.

![Graph Customization](image)

Figure 4-6  Bullet Graph builder: Graph Customization

In the Advanced section of the Bullet Graph, you can control the generation of the main action. By default, this option is not selected. Figure 4-7 is an example of the Bullet Graph used in run time.

![Bullet Graph builder: Run time](image)

Figure 4-7  Bullet Graph builder: Run time

### 4.3.5 Contextual Information

The Contextual Information builder is used to display additional information, such as Sales Rep contact information, for a specific element on a page. A pop-up window displays the information when you click or hover over the element.

In the Page Location section, you define where on the page the contextual information functionality exists. Figure 4-8 is an example of the Page Location inputs.

![Contextual Information builder: Page location](image)

Figure 4-8  Contextual Information builder: Page location

In the Information Definition section, you define the contextual information. The Initialization Action input is used to call an action that can be used to execute a data service. This is useful for those instances when the contextual information is retrieved from a separate data service call. The input Content Type gives you two options for displaying the contextual information.
They are:

- Page in model
- Text Content

The Page in Model option allows you to choose a page in the model that displays the contextual information. The Text Content option can be used to display information as HTML, a JSP, or an Indirect Value. Figure 4-9 is an example of the input using the Page in Model option. Also, displayed in Figure 4-10 is an example of the input using the Text Content option.

In the Trigger section, you define how the contextual information is triggered. The options available are:

- Named Tag
- Link
- Image

The Enable Trigger input must be selected to give access to the Trigger inputs. The Named Tag option displays the contextual information when the user hovers over the named tag on the page. The Link option displays the contextual information when the user hovers over a link. The Image option displays an image relative to the named tag that the user can hover over to display the contextual information. Figure 4-11 on page 87 is an example of these inputs. Also displayed in Figure 4-12 on page 87 is an example of the Contextual Information builder used in run time.
4.3.6 Hierarchy Drilldown

The Hierarchy Drilldown builder creates a series of pages and allows users to drill down to more detailed levels of data. The navigation through different levels is done by clicking links to pertinent data. This builder provides you with the functionality of *breadcrumbs* navigation or you can use custom navigation.

In the Page Flow section, you specify the pages of the hierarchy and the corresponding action to load the data. By default, the WebSphere Dashboard Framework provides HTML for the page to import input. It is located in /solutions/dashboard/hierarchydrilldown.html. A typical input for the Action to Load input is a call to a data service. For example, a data service call can be specified in this input, such as `DataServices/SalesHierarchy/getSalesOfficesForRegion` and the required parameters, `${Variables/SalesHierarchyDrilldown_SalesRegions_SelectedRowData/RegionRow/REGION_ID}`, can be defined in the Override Inputs section of the Service Consumer. Figure 4-13 on page 88 is an example of the Override Inputs section of the Service Consumer builder. Figure 4-14 on page 88 is an example of the Page Flow inputs of the Hierarchy Drilldown builder.
Here, you can also specify the drilldown link and the name to appear in the breadcrumbs navigation as specified in Figure 4-14.

Using this builder, you can also control the generation of a main action. You can also enable the Export to Excel, Print Page, and Annotations functionality. Figure 4-15 on page 89 is an example of these inputs.
Figure 4-16 on page 90 is an example of the Hierarchy Drilldown builder used in run time. This example uses the hierarchy of Regions, Offices for the selected region, and Opportunities for the selected office.
4.3.7 ID Gatherer

The ID Gatherer builder helps gather IDs, such as Office ID, Region ID, and so on, for a logged in user. This builder helps provide a mechanism to gather IDs by hooking into the OnRequest event and not allowing other processing to continue until the IDs have been gathered and stored. These IDs are stored in the session common to all of the portlets for the user. If an ID has been stored, but someone else changes it, the new value is used. Multiple key names can be specified, and this builder does not allow processing to proceed until all IDs specified are available. Figure 4-17 on page 91 is an example of the inputs for this builder.
4.3.8 Map

The Map builder is used to display a map that displays color-coded regions based on a dynamically calculated status. This builder is similar to the Summary and Drilldown builder, which we discuss in 4.3.18, “Summary and Drilldown” on page 111, because these builders both provide you with a Summary Table, Chart, and Drilldown functionality.

The Map builder provides you with many input options. The first required input that you define is Name. The Name input defines the name of the Map builder. In the Initialization Action input, you specify which action to call to initialize the data. For example, a possible value for this input is `DataServices/SalesKPIs/getSaleKPIs`. If you use a Data Service call for this input, the required input Data automatically is populated with the results of the action. An example of this is `DataServices/SalesKPIs/getSaleKPIs/results`. If any other Initialization Action is specified, such as a method call, you must populate the Data input with a result set. You can also enter a page title for the summary page in the Page Title input. Figure 4-18 is an example of the Map builder properties.

Next, you have a section to define the map chart properties. There is a Provide Custom Style input, which after it has been selected, allows you to define the Chart Style. An XML file is specified for the Chart Style input. If the Provide Custom Style input is not selected, the default chart style is used. This default style file can be found in `/solutions/dashboard/chartstyles/Map.xml`. The Map File input specifies the map data for the chart. Sample maps are prepackaged with WebSphere Dashboard Framework. These sample maps can be found in `/solutions/dashboard/chartstyles/maps`. In the Column for Region input, you can specify the column whose data becomes the regions in the map. The drop-down list is populated with a list of available options as a result from the value that you specified in the Data input. There is also an option to specify a value that is not listed in the
drop-down list by using the ellipses (....) button. The Column for Value input allows you to specify the data to use as the value for each region in the map. Figure 4-19 is an example of the Chart Properties inputs.

<table>
<thead>
<tr>
<th>Chart Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide Custom Style</td>
</tr>
<tr>
<td>Chart Style</td>
</tr>
<tr>
<td>Map File</td>
</tr>
<tr>
<td>Column For Region ^</td>
</tr>
<tr>
<td>Column For Value</td>
</tr>
</tbody>
</table>

*Figure 4-19  Map builder: Chart Properties*

The next input that you can set is Match Data Regions With Map. This input gives you the option of either automatically matching the Map Values to the Data Values or manually defining this relationship. When Manually is selected, the list of available Map Values are prepopulated, and you simply need to specify to which Data Values the Map Values correlate. Figure 4-20 is an example of the Match Data Regions With Map inputs.

<table>
<thead>
<tr>
<th>Match Data Regions With Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Radio Button] Automatically</td>
</tr>
<tr>
<td>![Radio Button] Manually</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Map Value</th>
<th>Data Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA/States</td>
<td></td>
</tr>
<tr>
<td>USA/Graphics</td>
<td></td>
</tr>
<tr>
<td>USA/States/IA</td>
<td>IA</td>
</tr>
<tr>
<td>USA/States/IL</td>
<td>IL</td>
</tr>
<tr>
<td>USA/States/IN</td>
<td>IN</td>
</tr>
<tr>
<td>USA/States/KS</td>
<td>KS</td>
</tr>
<tr>
<td>USA/States/MI</td>
<td>MI</td>
</tr>
<tr>
<td>USA/States/MN</td>
<td>MN</td>
</tr>
<tr>
<td>USA/States/MD</td>
<td>MD</td>
</tr>
</tbody>
</table>

*Figure 4-20  Map builder: Match Data Regions With Map*

Finally, within the Chart Properties section of the Map builder, you can specify values for the following inputs: Chart Format, Chart Width, Chart Height, and Show Chart Legend. The Chart Format lists the available options of PNG, JPG, SWF, SVG, and WBMP. The chart’s width and height can be modified from the default values of 400 and 300. You can also choose whether to display the chart’s legend by choosing one of the options: Has Legend or No Legend. Figure 4-21 on page 93 is an example of the chart properties inputs.
In the Summary Pages to Create section, you have the option of specifying whether to display both the Chart and Table pages or only the Chart page. Here, you can also control if a main action is created. This is useful for those instances when you might want to create a custom main action to perform additional processing. Figure 4-22 is an example of these inputs.

The Map builder also has a Page Layout and Navigation Section. In this section, you can customize the page’s presentation and navigation. There is an input of Page Layout Mode where you can choose to display the chart and table on separate pages or on the same page. You can also customize the page navigation type by selecting either the Action Menu, Button option, or the Tabbed Pages option. The Action Menu or Button option gives the user a selectable menu or button option to select to which page to navigate. The Page Order and Labels input allows you to rearrange the order in which the pages display and define the labels of the pages that appear in the Action Menu or Tabbed Pages. A default initial page can also be specified by selecting the option of either Chart or Page. The Tab Base HTML Page, Tab Background Class, Selected Tab Background Class, Tab Text Class, and Selected Tab Text Class are where you specify the HTML and classes to define the tab’s presentation. Default values are provided for all of these inputs. Figure 4-23 on page 94 is an example of the inputs for the Page Layout and Navigation section.
In the Status Indicator section, you define the thresholds for the defined status options. Here, you can define status options by providing the status, the short description, and the color that corresponds to the status. The Indicator Logic input gives you the options of Match a value to status, Compare values to thresholds, and Use Custom Logic. If the list of status options defined in the Status Options input matches the list of statuses returned from the data service, you can select this option. If you select the option of Compare values to thresholds, a Thresholds table appears for you to define the threshold value to compare to the selected value. This selected value is defined using the Value for Comparison input, which gives you the options of Selected Column for Value or Other Reference. If a selected value does not meet any of the conditions defined, you can specify a default status to display. Figure 4-24 on page 95 is an example of inputs for the Status Indicator section.
In the Summary Table Properties section, you specify the column to use as a link to drill down into the details. This input is only important if the drilldown functionality is used. A default HTML template file is provided, but you can also customize it. Figure 4-25 is an example of the inputs for the Summary Table Properties inputs.

In the Drilldown Properties section, you define the inputs required for the drilldown functionality. By default, No Drill Down is specified. But, if you want drilldown functionality, Call an Action and Display Data and Call an Action options are available. When the Call an Action and Display Data option is selected, additional inputs are available to you. The most important inputs are Action to Call and Drilldown Data. In the Action to Call input, you specify the data service method to execute. Two possible examples of this are:

SalesKPIsGetSaleOppsWithArgs(${Variables/Map_SelectedRowData/Row/OFFICE_ID})

and

DataServices/SalesKPIs/getSaleOpps/results

For our purposes, we use the method call that allows us to pass the method an argument. The Map builder creates the variable SelectedRowData that contains the value of the selected row. Another input that is available to you is the Navigation Type. Here, you can specify Breadcrumbs, Back Button, or No Default Navigation. This is the navigation that the user uses to navigate back to the summary page. Figure 4-26 on page 96 is an example of the inputs for the Drilldown Properties section.
The Action Menu Options section allows you to configure certain features to make them available for the user. The ability to add Export to Excel and Print Page contents functionality is incorporated into this model. These features are available by merely checking the check box next to the appropriate input. There are also inputs available to you to define the Tooltip Text and Exported File Name. Annotations can also be set up for the Map builder. We discuss Annotations in detail in Chapter 7, “Annotations” on page 179. Figure 4-27 is an example of the inputs for the Action Menu Options section.

Figure 4-27  Map builder: Action Menu Options

Figure 4-28 on page 97 is an example of the Map builder in run time.
4.3.9 ODS Export

The ODS Export builder exports the content of a variable or method call to an Open Document Spreadsheet (.ODS) file. The ODS Export builder sends portlet data to an ODS file, which can then be displayed in a browser window. The original content must be in XML format. After you save the builder call, the content is read and converted to an ODS file. When users click the Export Image button, button, or link that you have added to a page, they see the spreadsheet document in the browser window. Figure 4-29 on page 98 is an example of the builder's inputs.
4.3.10 Query Filter

The Query Filter builder is discussed in detail in Chapter 5, “Filtering” on page 133.

4.3.11 Query Filter Form

The Query Filter Form builder is discussed in detail in Chapter 5, “Filtering” on page 133.

4.3.12 Query Filter Observer

The Query Filter Observer builder is discussed in detail in Chapter 5, “Filtering” on page 133.

4.3.13 Record List and Detail

The Record List and Detail builder is used to display a list and detail page. This builder is especially useful when creating a portlet that responds to an event and displays record details. You can achieve this result by using only the detail functionality of this builder.
In the Pages to Create section, you can select to create a record list page, record detail page, or both. Here, you can also control the generation of the main action. Figure 4-30 is an example of these inputs for the Pages to Create section.

![Figure 4-30 Record List and Detail builder: Pages to Create](image)

In the Record List Properties section, you define the record list data, enable paging, and specify the detail link column. An example entry for the input Action to Load Data is `DataServices/Customer/getAllCustomers`. This executes the data service to return the data to display, which is specified in the input Record List Data. Figure 4-31 is an example of the inputs for the Record List Properties section.

![Figure 4-31 Record List Details builder: Record List Properties](image)

The Record Details Properties section is available to you when the Pages To Create - Record Detail Page option is selected. Here, you define the record detail data and the navigation used to allow the user to navigate back to the list page. You can select the source of the record details using the options Selected Row in Record List or Other Reference. When Other Reference is selected, you are able to define the action to call to load the detail data. This is specified in the Action to Load Data input. An example of this input is: `CustomerGetCustomerDetailsWithArgs(${Variables/CustomerListAndDetail_SelectedRowData/Row/CUSTOMER_ID})`. This example calls the data service method that accepts arguments. The Record List and Detail builder creates a `SelectedRowData` variable, which is used as the argument to the data service method. Figure 4-32 on page 100 is an example of the inputs for the Record Details Properties section.
In the Action Menu Options section, you can enable the Auto Show/Hide feature of the menu, export to Excel, enable Print Page, and enable Annotations. We discuss Annotations in more detail in Chapter 7, “Annotations” on page 179. An example of the inputs for the Action Menu Options is shown in Figure 4-33.
4.3.14 Snapshot Report

The Snapshot Report builder is used to create a printable report of the top level pages in a portlet. This is an extremely useful functionality when a portlet contains important data or tabbed pages. The Snapshot Report creates a report that displays the contents of more than one page.

The Contents for Snapshot Report section gives you the opportunity to specify which pages to include in the report. Here, you can also specify the Snapshot Load Action if a portion of the report requires that an action execute before rendering the contents. Figure 4-34 is an example of the Contents for Snapshot Report inputs.

![Contents for Snapshot Report](image)

**Figure 4-34  Snapshot Report builder: Contents for Snapshot Report**

The View Snapshot Control section allows you to define the control type for displaying the report and where it needs to be located on the page. The available control types are:

- Link
- Button
- Image Button
- None (use separate control builder)

In the Page Location, you specify on which page this control resides and where the control is located on the page. If the Link or Button option is chosen as the control type, the Button/Link text input is available to you to define the text to display. Figure 4-35 is an example of the View Snapshot Control section inputs.

![View Snapshot Control](image)

**Figure 4-35  Snapshot Report builder: View Snapshot Control**
And the last section, the Report Appearance section, is where you can modify the report’s look and feel. One very important input is the Show Filter Options input. This input is useful when the model uses a Query Filter Observer, which is discussed in detail in Chapter 5, “Filtering” on page 133. When you select this input, the filters that were selected in the Query Filter portlet that were used as parameters for the portlet’s data service display. Figure 4-36 is an example of the Report Appearance section inputs.

Figure 4-36  Snapshot Report builder: Report Appearance

Figure 4-37 is an example of the Snapshot Report builder used in run time. The first figure, Figure 4-37, displays the portlet with the snapshot report link and the page tabs. The second figure, Figure 4-38 on page 103, displays the contents of the snapshot report and both the chart and the table pages.

Figure 4-37  Snapshot Report builder: Report link and page tabs
4.3.15 Status Indicator

The Status Indicator builder adds a status indicator, such as a colored text cell or image. You can add this status indicator by either applying the colored text to an existing column in the table or by creating a new column to display an image or text indicator.

The Status Indicator properties section supplies you with the inputs of Name, Field, and Type of Indicator. Use the Name input for specifying the name of the builder. Use the Field input to identify the field that is evaluated against the status thresholds. The Type of Indicator input gives you three options from which to choose:

- **Style formatting**
  The style formatting indicator type highlights the field’s text or background that was specified in the Field input.

- **Image indicator**
  The image indicator uses an image to indicate the status. This image is either added to an existing table column or a new column is added.

- **Text indicator**
  The text indicator adds textual information to indicate the status. This textual information also is either added to an existing or newly added column.
Figure 4-39 is an example of the Status Indicator inputs.

<table>
<thead>
<tr>
<th>Name</th>
<th>RevenueStatusIndicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>[MapTable.Map_table.RowSet.Row]/PERCENT</td>
</tr>
<tr>
<td>Type of Indicator</td>
<td>○Style formatting ○Image indicator ○Text indicator</td>
</tr>
</tbody>
</table>

**Figure 4-39  Status Indicator properties**

The Indicator Options section allows you to define the status options and the status option location in the table. When the Image indicator option is selected for the input Type of Indicator, the following inputs are available to you. The first option, Image Location, gives you the options of Add image in existing column or table cell and Add new column with image. For our purposes, we selected the option **Add new column with image**. Based upon the option selected, the status indicator image is placed in either an existing or a newly created column in the table. You also have the ability to specify the New Column Name and Column Placement by entering values in these inputs. The Status Options input is required, and this area is where you define the list of available statuses. Here, you also specify an image to display. WebSphere Dashboard Framework comes with images that you can use. These files are located in /solutions/images. Figure 4-40 is an example of the Indicator Options inputs.

**Figure 4-40  Status Indicator builder: Indicator Options**

In the Indicator Logic section, you define the logic to use when evaluating the field. In the Indicator Logic input, you select the option to use to evaluate the status. The available options are:

- Match a value to status
- Compare value to thresholds
- Use custom logic
- Alert determines status
- Same as other indicator on page

For our example, we use the option Compare value to thresholds. This option gives you a new input, Thresholds. The Thresholds input defines the threshold value to use to compare to the field value to get the status. The field value is specified using the Value for Comparison input. The Value for Comparison input gives you two options: Selected Field or Other Reference. We use **Selected Field** for our example. The selected field is compared to each of the thresholds defined in the Thresholds input. This comparison results in the status to display in the table. There is also a Default Status input where you can specify a default status if none of the threshold conditions are met. Figure 4-41 is an example of the Indicator Logic inputs.
4.3.16 Status Indicator Legend

The Status Indicator Legend builder is used in conjunction with the Status Indicator builder. This builder is used to provide an explanation of the visual indicators on the page.

In the Status Indicator Legend input Name, you define the name of the builder. In the Page Location input, you define on which page the legend displays and where the legend’s location resides. Figure 4-43 on page 106 is an example of the Page Location inputs.
In the Base Status Indicator input, you specify which status indicator lookup to use. If the model uses a Status Indicator or Map builder, the list is prepopulated with these lookup options to use. The Status Option Setting in Legend provides you with two options from which to choose: Use Default Status Option Settings and Modify Status Option Settings. For our example, we use the option **Modify Status Option Settings**. This option allows you to modify the explanatory text for each of the options. Use this option when you want a more descriptive status definition. Figure 4-44 is an example of the Status Option Settings inputs.

**Figure 4-44** Status Indicator Legend builder: Status Option Settings

Figure 4-45 on page 107 is an example of the Status Indicator Legend used in run time.
4.3.17 Status Page

The Status Page builder creates a page with a dial chart and table that graphically indicate performance against a goal. The status options are defined in the Status Properties section. Here, you can also specify the desired trend and current value. Figure 4-46 is an example of the inputs for the Status Properties section.

You can also define the pages to create. The available options are Chart and Table. You can select both of these options or only one option. Here, you can also control the generation of a main action. Figure 4-47 on page 108 is an example of these inputs.
If the Table Page option is selected, additional inputs are available to you. In the section Page Layout and Navigation, you define the page layout and specify whether the chart and table are on the same page or separate pages. You can also specify the order and the labels of the pages. Figure 4-48 is an example of these inputs.

In the Chart Properties section, you can provide a custom chart or style. The chart title, legend, width, and height can also be modified. Figure 4-49 is an example of the Chart Properties inputs.

In the Table Properties section, you define the value styles for each of the statuses. The Show Actual Value, Show Goal Value, and Show Variance Value options can be selected independently of each other to display those columns in the table. The Goal Determination input is used to specify a method to determine the goal. The option Use upper threshold for a status can be selected to use one of the threshold values defined in the Status Options properties. Figure 4-50 on page 109 is an example of the inputs for the Table Properties section.
In the Action Menu Options section, you can add the functionality of Export to Excel, Print Page, and Annotations. Figure 4-51 is an example of these inputs.

Figure 4-50  Status Page builder: Table Properties

Figure 4-51  Status Page builder: Action Menu Options

Figure 4-52 on page 110 and Figure 4-53 on page 110 are examples of the Status Page builder in run time.
Figure 4-52  Status Page builder: Runtime example

Figure 4-53  Status Page builder: Runtime example two
4.3.18 Summary and Drilldown

The Summary and Drilldown builder creates a summary page that displays data summarized by a field, such as department, customer, or time period. The chart and table created by this builder enable a portlet user to drill down on a summary field to view more detailed information about that field. You must specify an Initialization Action in order to initialize the summary data. An example of this action is a call to execute a data service. An example of these inputs is shown in Figure 4-54.

![Figure 4-54 Summary and Drilldown builder properties](image)

In the Summary Pages to Create section, you specify which pages to create. The available options are:
- Both Chart and Table
- Only Chart
- Only Table

Here, you can also control the generation of a main action. Figure 4-55 is an example of these inputs.

![Figure 4-55 Summary and Drilldown builder: Summary Pages to Create](image)

In the Page Layout and Navigation, you define the page layout, page navigation, page order and labels, and the default initial page. This section is available when you select the option Both Chart and Table. Figure 4-56 is an example of these inputs.

![Figure 4-56 Summary and Drilldown builder: Page Layout and Navigation](image)
In the Summary Chart Properties section, you can specify which type of chart to use, the X-axis and Y-axis, and the chart style. WebSphere Dashboard Framework comes with sample chart styles that are located at /WEB-INF/factory/samples/WebCharts/. The available chart styles are:

- Bar
- Line
- Pyramid
- Area
- Pie
- Radar
- Step

In the Column Selection input, you can specify which columns to use for the Y-axis. The available options are Use All Columns, Include Specific Columns, and Exclude Specific Columns. Figure 4-57 is an example of these inputs.

![Summary Chart Properties](image)

**Figure 4-57  Summary and Drilldown builder: Summary Chart Properties**

In the Summary Table Properties and Drilldown Properties sections, you specify the link to use to navigate to the drilldown data. There are three drilldown types available to you:

- No Drilldown
- Call an Action and Display Data
- Call an Action

When the Call an Action and Display Data option is selected, the Action to Call and Drilldown data inputs are made available to you. An example of an action to call input is a data service method that accepts inputs. You can pass the `SelectedRowData` variable as the input. An example of this is:

```
SAOrdersGetOrdersByCustomerIdWithArgs(${Variables/CustomerOrdersSD_SelectedRowData/Row/CUSTOMER_ID})
```

You can also specify the navigation type. The available options for this are Breadcrumbs, Back Button, and No Default Navigation. No Default Navigation is selected when you want to provide custom navigation using other builders. Figure 4-58 on page 113 is an example of the Summary Table Properties and Drilldown Properties inputs.
The Summary and Drilldown builder also has an Action Menu Options section where you can enable the Export to Excel, Print Page, and Annotations functionality. Figure 4-59 is an example of these inputs.
4.3.19 Summary Row

The Summary Row builder provides an additional row to the table that summarizes all of the values of a specified column. This builder provides you with a quick and easy solution to perform mathematical calculations.

The Container Field input requires you to select the field whose child element is to be modified. After this input is populated, you see a list of available columns to perform calculations. The Summary Types available to the user are:

- No Summary
- Sum of Values
- Average of Values
- Count Number of Values
- Text or Custom Value
- Add Two Columns
- Subtract Two Columns
- Multiply Two Columns
- Divide Two Columns
- Divide and Present as Percent

For our example, we applied the Summary Type of **Sum of Values** on the Actual KPI column, and we applied the Summary Type of **Average of Values** on the Percent column. Figure 4-60 on page 115 is an example of the Summary Row builder inputs.
Figure 4-60  Summary Row builder inputs

Figure 4-61 is an example of the Summary Row builder in run time.
4.3.20 Table Customizer

We discuss the Table Customizer builder in detail in 4.4.4, “Table Customizer” on page 126.

4.4 Customizing the look and feel

The following sections describe ways to customize your portlet. The majority of a model's inputs can be profiled. After these inputs are profiled, you can expose the configuration of these profiled inputs in the portlet's edit or configure options. Next, we discuss a few customization options, such as creating a custom edit or configure page for your portlet, customizing a portlet's thresholds, customizing a portlet's chart, and customizing a portlet's table.

4.4.1 Creating a custom edit or configure page

To create a custom edit page, you use the Portlet Customizer builder. This builder creates custom edit or configure pages for a portlet. Here, you specify which portlet to customize. After the portlet is selected, a list of the available profile sets appears. You can modify the number of pages to display in the customizer UI by selecting a value from the drop-down list next to the Number of Pages input. If the Used Tabbed Pages input is selected, tabbed page navigation is used in place of page navigation buttons. The Input Values section is where you can specify which inputs are on which pages. Select the Treat Text as XML input when using the Table Customizer builder. This enables the Data Page builder to recognize the Data Column Modifier data as a Table (Columns and Rows). In the Display Options section, you can select the Multiple Pages to import option, which allows you to import different pages and map them to specific pages in the customizer. Figure 4-62 on page 117 is an example of the inputs for the Portlet Customizer builder.
**Portlet Customizer**

Create custom edit or configure pages for any portlet. Specify the portlet and determine which profiles to use for customization in the portal.

### Properties

- **Name**: CustomerOrdersCustomizer
- **Portlet**: portlets/CustomerOrders
- **Profile Sets**:

<table>
<thead>
<tr>
<th>Profile Set</th>
<th>Input Prefix</th>
<th>Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomerOrders</td>
<td>CustomerOrders</td>
<td>Values</td>
</tr>
</tbody>
</table>

### Update Profile Sets for Portlet

- **Number of Pages**: 1
- **Used Tabbed Pages**: ☑
- **Input Values**:

<table>
<thead>
<tr>
<th>Input</th>
<th>Page</th>
<th>Prompt</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom</td>
<td>1</td>
<td>DataFor...</td>
<td>&lt;Columns&gt;&lt;Column&gt;&lt;Name&gt;OFFICE_ID&lt;/Name&gt;&lt;□□&lt;Status&gt;Def</td>
</tr>
</tbody>
</table>

- **Treat Text as XML**: ☑

### Display Options

- **Page Import**:
  - Single Page
  - Multiple Pages
- **Default Page URL**: /factory/pages/default_portletcustomizerpage.html
- **Make URLs Absolute**: ☑
- **StyleSheet URL**: /factory/pages/default_portalcustomizer.css
- **HTML Template File**: 

### Label Translation Settings

Settings for managing the translation of labels in a way that can be easily internationalized.

- **Localized Resource**: 

### Advanced

- **Post-Save Method**:
- **Post-Save Method Behavior**: Always call the Post-Save method
- **Generate main method**: ☑
- **Add Clear Data Button**: ☑
- **Remain in Customizer on error**: ☑

*Figure 4-62  Portlet Customizer builder*
4.4.2 Customizing a threshold

To customize a threshold, we use the Map builder in our example. First, we profiled the Thresholds input of the Map builder. You only need to click the profile icon next to the Thresholds input and accept the default profile. Figure 4-63 is an example of profiling the thresholds.

![Figure 4-63 Threshold profile values](image)

Next, we need to create a new model to contain a Portlet Customizer model. This creates the customizer UI page. An example of this builder is shown in Figure 4-64 on page 119.

**Important:** Be sure to select the input Treat Text as XML.
Finally, we only need to specify the Custom Configure Model in the Portlet Adapter of the model containing the Map builder. Figure 4-65 on page 120 is an example of the Portlet Adapter builder.
The threshold customization is ready to test. Figure 4-66 on page 121 is an example of the Map portlet before the threshold customization, the threshold customization UI, and the Map portlet after the threshold customization.
4.4.3 Adding a Chart Customizer

To add Chart Customization to a portlet, you must first profile the Chart Type input of the Summary and Drilldown builder. You only need to click the profile icon next to the Chart Type and accept the default profile. An example is shown in Figure 4-67 on page 122.
Next, you need to create a new model to contain the Portlet Customizer builder. This creates the page for the customizer UI. In this model, you can also format the page using Select or Radio Button builders, for example. Figure 4-68 on page 123 is an example of the Portlet Customizer model used for the chart customization.
Finally, you need to specify the Custom Configure Model in the portlet adapter of the model that contains the Summary and Drilldown builder where the chart type input was profiled. Figure 4-69 on page 124 is an example of these inputs.
Now, we can test to see the chart customization functionality by clicking the configure option of the portlet. Figure 4-70 on page 125 shows the original chart portlet, which displays a Pie chart, the chart type customization UI, and the results of changing the chart type to Bar.
Figure 4-70  Chart Customizer: Run time
4.4.4 Table Customizer

Use the Table Customizer builder to:

- Create a customizer portlet for editing Data Column Modifier data
- Enable or disable the paging function
- Set page size
- Enable or disable the category view
- Set a drilldown link if a drilldown builder is used in the portlets

This builder is especially helpful to the user, because it allows the user freedom to customize the portlet's look and feel.

The steps are:

1. The first step that you must do is to profile the contents of the Data Column Modifier builder by selecting the profile input button next to the column contents. Figure 4-71 on page 127 is an example of the profiled column contents.
2. When the profile input button is clicked, you are prompted with a Profile Input window to create the profile entry name and profile values. By default, a profile entry name is created. You can choose to use this default name or create a new entry name. Figure 4-72 on page 128 is an example of the Profile Input window.
3. A default Profile is created for you named Default. Here, you can modify the profile values of the profile inputs by selecting the pencil icon to the right of the profile value. Figure 4-73 is an example of the profile value inputs.

4. After the profiling of the Data Column Modifier builder is complete, you must create a new model to contain the Table Customizer builder. This new model must first contain a Portlet Customizer builder. The Portlet Customizer builder is used to create a custom edit or configure pages for a portlet. Here, you specify the portlet and which profiles to use for customization. We explain the inputs required for this builder in 4.4.1, “Creating a custom edit or configure page” on page 116.
5. Next, you need to add a Table Customizer builder. In this builder, you specify the container field to use, can modify column properties, can make columns required, and can enable paging, drilldown, and category view customization. Figure 4-74 on page 130 is an example of the inputs for the Table Customizer builder.
### Table Customizer

Create a customizer panel for editing Data Column Modifier data, enabling or disabling the paging function and setting page size, enabling or disabling the category view, and setting a drilldown link if a drilldown builder is used in the panel.

#### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>CustomerTableCustomizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Field</td>
<td>/Page/..\CustomerOrders\CustomerOrder\DataPage/Table/..\ColumnInfo\Column\Column</td>
</tr>
</tbody>
</table>

#### Settings for Column Management

Settings for generating and managing the column properties

Tip: You can reorder the fields as they appear on the page by dragging and dropping the rows in the table below. Type directly in cells to set properties.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Status</th>
<th>Column Heading</th>
<th>Column Width</th>
<th>Alignment</th>
<th>Column Sort Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column.Counter</td>
<td>Do Not Change</td>
<td>#</td>
<td>Default</td>
<td>NotSortable</td>
<td></td>
</tr>
<tr>
<td>Order</td>
<td>Do Not Change</td>
<td>Order</td>
<td>Default</td>
<td>NotSortable</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Do Not Change</td>
<td>Name</td>
<td>Default</td>
<td>NotSortable</td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>Do Not Change</td>
<td>Status</td>
<td>Default</td>
<td>NotSortable</td>
<td></td>
</tr>
<tr>
<td>Heading</td>
<td>Do Not Change</td>
<td>Heading</td>
<td>Default</td>
<td>NotSortable</td>
<td></td>
</tr>
<tr>
<td>Width</td>
<td>Do Not Change</td>
<td>Width</td>
<td>Default</td>
<td>NotSortable</td>
<td></td>
</tr>
<tr>
<td>Sortable</td>
<td>Do Not Change</td>
<td>Sortable</td>
<td>Default</td>
<td>NotSortable</td>
<td></td>
</tr>
<tr>
<td>Alignment</td>
<td>Do Not Change</td>
<td>Alignment</td>
<td>Default</td>
<td>NotSortable</td>
<td></td>
</tr>
</tbody>
</table>

#### Settings for Row Management

Settings for managing the row position and required properties

Tip: You can set required or position-fixed properties for different rows. Type directly in cells to set properties.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Required Field</th>
<th>Fixed Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFFICE_ID</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>CUSTOMER_ID</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>CUSTOMER_NAME</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>REP_ID</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>ACCOUNT_STATUS</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>FIELD</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>PHONE</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>POSTCODE</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>FAX</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

#### Settings for Paging Control of the Table Data

Settings to control paging of the table data

Enable Paging Customization: [ ]

#### Settings for Drilldown Field Management

Settings for drilldown behavior

Enable Drilldown Customization: [ ]

#### Settings for Category View Control

Settings for category view behavior. Paging is not recommended if this feature is enabled.

Enable Category View Customization: [ ]

> Localization

---

*Figure 4-74  Table Customizer builder*
6. The table customization is available for configuration by the user selecting the configure option of the portlet. Figure 4-75 is an example of the table customization feature used in run time.

![Customer Orders]

Figure 4-75  Table customization: Run time
Filtering

Filters provide a user with a way to show the exact data that the user wants, depending on the value in the filter. An example is a sales manager, who looks after a series of sales offices. When the sales manager logs in to that sales manager's dashboard, the sales manager sees data for all of the offices for which that sales manager is responsible. Using a filter allows that sales manager to focus on data that is specific to one sales office. For example, the sales manager can focus on the New York office by using a filter, and therefore, the applications that make up the sales manager's dashboard display data specific to that office.

Filtering is a powerful feature of WebSphere Dashboard Framework that gives your users power over what they see. In this chapter, we explain what a query filter is and how to build one. Also, we explain how to enable inter-portlet communication using the Query Filter Observer builder. We describe the creation of a role-driven filter in the final section and provide examples for you, the developer.

In this chapter, we discuss the following topics:

- Creating a Query Filter:
  - Query Filter Form builder
  - Query Filter builder
- Enabling portlets to respond to filters:
  - Query Filter Observer
- Profiling by creating a role-driven filter
5.1 Create a Query Filter

To create a functional Query Filter, you must use three important builders. They are:

- Query Filter Form
- Query Filter
- Query Filter Observer

The Query Filter Form builder creates the form to contain the filters created by using the Query Filter builder. To enable other portlets to realize the filter values, the Query Filter Observer builder is used. Inter-portlet communication is achieved simply using these three builders. The following sections describe how to use these builders in your application.

5.1.1 Query Filter Form

The Query Filter Form is the first builder that the user needs in order to create Filtering functionality. This builder creates the form that sets the filter values in the session to be used by other portlets. One to many Query Filters can be added to this form.

The Query Filter Form properties section requires you to define the Name and a Session Key Name that specifies the name of the key to be used to place the query filter XML in the session. Figure 5-1 is an example of the inputs for these properties.

![Figure 5-1 Query Filter Form builder properties](image)

In the Form Submit Behavior section, you select the Submit Behavior of the form. The options from which you can choose are:

- Automatic: Whenever an input is changed
- Manually: Provide an ‘Apply’ button

In the Form Name required input, you enter the form name that was specified in the base HTML page. The default value for this input is queryFilterForm. You can also edit the label to display for the Apply button. Figure 5-2 is an example of the Form Submit Behavior section inputs.

![Figure 5-2 Query Filter Form builder: Form Submit Behavior](image)

In the Advanced section, there is one input, Generate Main Action. If checked, this option generates the main action for the model. You can also remove the check from this option and
create a custom main action to allow for additional processing to occur. Figure 5-3 is an example of the Advanced section inputs.

![Figure 5-3 Query Filter Form builder: Advanced](image)

### 5.1.2 Query Filter

To get started using the Query Filter builder, you must specify the Filter Form to use. The Filter Form input is prepopulated with a list of available filter forms. You must select the filter form for where the Query Filter resides.

The Query Filter builder gives you four types of filter options: Lookup, Data Range, Hierarchy, and Custom UI.

**Lookup**

The Lookup filter option allows you two UI control types. These are:

- Dropdown List (Select)
- Radio Button Group

**Dropdown List (Select)**

The Select option displays a drop-down list of elements defined by the specified Lookup Table builder. This builder is specified by selecting the lookup table from the list of available options in the Lookup Table Used input. You can also define a default value to be selected by entering a value in the Default Value option. Other options available to you are Allow Multi Select and Size (List Height). You can check the option Allow Multi Select to allow the user to select more than one option in the drop-down list. The Size option gives you the ability to define how many options display when the drop-down list is expanded without scrolling.

**Radio Button Group**

The Radio Button Group option displays a group of radio buttons that correspond to the result set of the lookup table. You can specify the orientation of the radio buttons by selecting either Horizontal or Vertical in the Radio Button Orientation input. Figure 5-4 is an example of the Query Filter builder with a UI Control Type of Radio Button Group.

![Figure 5-4 Query Filter builder: Radio Button Group](image)
If the Radio Button Group UI Control Type is selected, you are presented with additional inputs. The section Radio Button Image Support allows you to use images in conjunction with the radio buttons. This functionality is enabled by selecting the check box for the input Use Image. After this input is selected, additional inputs are available to you. In Image Options, you define the Image IDs to use for comparison to the values of the lookup table. The Image Options also allow you to define a short description and specify the image's file, width, and height. The Image Logic input presents you with two options:

- Match option's value to image ID
- Custom Logic

The Match option's value to image ID option automatically matches the Image ID defined in the Image Options section to a value returned from the lookup table. The Custom Logic option allows you to call an Action to evaluate the image for each row. Input Mappings for the arguments of the action can also be defined. Here, you can pass the variables created from the Query Filter builder to the action. An example of this variable is \${Variables/Offices_RBGLoopVar/VALUE}. \textit{VALUE} corresponds to the Value Tag defined in the Lookup Table builder. A Default Image can be selected to use when no conditions are met. Image Placement is where you can specify the relative position of the image. There is also a Hide Radio Button option, which can be selected to hide the radio button and only display the image.

Figure 5-5 on page 137 is an example of the Radio Button Image Support section's input using the Custom Logic option. An example of the Lookup Table being used for the Query Filter is provided in Figure 5-6 on page 137. Figure 5-7 on page 138 is an example of the Method that is called in the Action input of the Query Filter. This method prints out the results of \${Variables/Offices_RBGLoopVar/VALUE} and \${Variables/Offices_RBGLoopVar/LABEL} to the SystemOut.log file. This method also returns the optionValue to use for evaluating the image to display.
Figure 5-5  Query Filter builder: Radio Button Image Support

Figure 5-6  Query Filter builder: Lookup Table
A runtime example of a query filter using the radio button group with the image support option is displayed in Figure 5-8 on page 139.
Figure 5-8  Query Filter: Radio Button with Image Support

**Date Range**

The Date Range filter option results in two calendar controls, a start date and an end date, that are placed on the query filter form. This option also places a drop-down list containing a list of values, such as This Week and This Month. When a selection is made, the list of values automatically populates the two calendar controls. There are three types of selectors for the date range control. They are:

- Date Range list with Start and End date inputs
- Only Date Range list
- Only Start and End Date inputs

When an option is selected for the date range control that includes the date range list, a prepopulated drop-down list is created. This list contains the options of:

- This Year
- Current YTD
- Last year
- Last half year
- Last week
- This week
- Last month
- This month
- Current and last 4 quarters
- This quarter
- Last 12 months
- This half year
- Last quarter
- Custom

When one of these options is selected in the drop-down list, the dates are automatically populated in the Start and End date controls. Figure 5-9 on page 140 is an example of the inputs for the Date Range filter.
Figure 5-9  Query Filter builder: Date Range

A runtime example of a query filter using the Date Range option is displayed in Figure 5-10.

Figure 5-10  Query Filter: Date Range Run time

**Hierarchy**

The Data Hierarchy filter option allows you to present a list of entries, one per level in the data hierarchy. The All entry is also an available option. An example of the input for the Data Hierarchy filter is shown in Figure 5-11 on page 141.
An example of the Hierarchy filter used in run time is shown in Figure 5-12. A drop-down list of Regions is presented to the user. The user selects a region, and a second drop-down list containing offices for the selected region appears.

Custom UI
The Custom UI filter option allows you to define the user interface if the previously described filter options do not provide the desired user interface. For example, you might want to define a user interface that displays check boxes.
5.2 Enabling portlets to respond to filters

After you have created a Query Filter, you can now enable inter-portlet communication. You achieve inter-portlet communication by adding the Query Filter Observer builder to those portlets that you want to have access to the query filter data.

5.2.1 Query Filter Observer

The Query Filter Observer builder requires you to specify the Query Filter Model that sets the session inputs in the builder. After the model is selected, the list of available filters populates the Actions for each Filter section. In this section, you can call a unique action for each filter whenever the corresponding filter is changed. You can also call a universal action when any of the filters are changed by listing the action or actions in the section Actions to Call on Change. Figure 5-13 is an example of these inputs.

![Query Filter Observer](image)

**Figure 5-13  Query Filter Observer**

5.3 Profiling: Creating a role-driven filter

After your Query Filter is developed and tested, you might want to add additional functionality, such as applying profiling to your filter. **Profiling** is a set of parameters that varies the way that an application behaves. A profile feeds values into builders based on the user’s identity or other contextual information, such as language. Using profiles, you can automatically generate variations of an application or portlet (from the same model) for different users, customers, or situations. In this example, we profile the office filter according to the user’s
portal group membership. We have created three portal groups: AMERICAS, ASIA, and EUROPE. Each group contains a user, for example, america is a member of the AMERICAS group, asia is a member of the ASIA group, and europe is a member of the EUROPE group. Figure 5-14 and Figure 5-15 show the users and groups.

For each user group, we display only the offices that are assigned to the group. For example, members of the ASIA group only have the office filter choices of China and Japan available to them. First, we need to profile our Lookup Table builder to use the data service appropriate
for the user’s group. We selected the profile icon next to the **Data Service** input in the Lookup Table builder. Next, we created three new profile values: America, Asia, and Europe. For each of these new profile values, we called the appropriate data service. For example, for Europe, we set the profile value to `DataServices/QueryFilter/getOfficesForEurope`. Figure 5-16 is an example of the profile value for the data service input of the Lookup Table builder.

![Figure 5-16 Lookup Table’s Profiled Data Service and Profile Input values](image)

Next, we need to profile the Default Value input of the Query Filter. By default, the value for this input is 1001. This default value does not work for the profile groups of Asia and Europe. The default values for these profile groups are 2001 and 3001. Figure 5-17 shows an example.

![Figure 5-17 Query Filter’s Profiled Default value and Profile Input values](image)

Last, we need to modify a few settings in the profile set. All of the profile sets are located in `/WebContent/WEB-INF/profiles/`. After the profile set is open and ready for modifications,
navigate to the Select Handler of the profile set. Select **WPS Group Segment Handler** from the Profile Selection Handler drop-down list. An example is shown in Figure 5-18.

![Figure 5-18 Profile Selection Handler](image)

Next, navigate to the Manage Profiles of the profile set. Here, we edit the profiles and map the portal group to it. Select the first profile to be edited and click **Edit Profile**. The Edit Profile box appears. Expand the **Advanced** section. Here, we click **Add External** to enter the portal group exactly as it appears in the portal. After all of the profiles have been edited, save the profile set. Figure 5-19 is an example of the Edit Profile dialog.

![Figure 5-19 Edit Profile](image)

The profiled query filter is now ready to test. We log in to the portal as America and expect to see the Query Filter portlet with only New York and Mexico as the available filter options. The New York filter option is selected by default. Figure 5-20 on page 146 is a window from our example application with America as the logged in user.
Figure 5-20  Example application
Alerts

In this chapter, we discuss the WebSphere Dashboard Framework Alerts module. We also build on the example application that we have created up to now, and we add alerting features to that application. We outline the steps that are required to configure alerts in a production environment and discuss various ways of setting up alert instances.

This chapter contains the following sections:

- An overview of the capabilities
- Configuration options
- Configuring data for alerts
- How to set up an alert instance
6.1 Overview of alert capabilities

One of the most significant features of WebSphere Dashboard Framework is the robust alerting module that ships with the product. This can be used to alert business users to events that are happening within the business in real time. An example might be that sales are falling below their forecast. When the threshold of this event occurs, an e-mail or a Short Message Service (SMS) message can be sent to a designated business person.

There are a number of types of alerts that you can configure, including event-based, asynchronous, and basic alerts. These alerts also support the escalation of alerts to which no one has responded. These alerts can also be configured by a business user without any knowledge of a programming language.

The primary components of the alerting module are:

- **The Alerts engine**
  
  The Alerts engine obtains the configuration information and based upon that information decides when to trigger an alert event. There is also a cache control for the Alerts engine that allows for greater performance in large scale deployments.

- **A set of application specific written alert evaluators**
  
  An *evaluator* is a business rule that is responsible for deciding when to trigger an alert. There are three types of evaluators that are either written as a simple script, a WebSphere Portlet Factory model, or a Java class. This gives you tremendous flexibility in defining the alerts. Each evaluator is used by an alert definition, which is configured in the Manage Alerts portlet. Alert expiration and notification are also specified in the Alert definition.

- **A persistence manager**
  
  A persistence manager is responsible for storing all of the configuration and user preferences for the Alerts engine.

Figure 6-1 shows how these elements interrelate.

![Figure 6-1 High-level architecture of the Alerting module](image-url)
6.2 Configuration

In this section, we describe the steps to set up your environment to take advantage of the Alerts engine. The high-level steps required are:

1. Selecting a persistence manager
2. Adding alert features to your projects
3. Configuring the Alerts engine to use the persistence manager
4. Configuring the persistence manager to the data source
5. Preparing the database

6.2.1 Step one: Selecting a persistence manager

The choice of persistence managers is abstracted from the Alerts engine. This gives you the freedom to choose the back-end data source that best suits the needs of your environment. There are functional differences between the data stores. Consider these differences up front to avoid discovering limitations as your applications develop.

There are two types of persistence:

- Relational database persistence
- XML persistence

**Relational database persistence**

The relational database stores all alerting data in a central relational database. The currently supported databases for persisting alert data are:

- Oracle®
- Microsoft SQL
- IBM DB2

**Note:** Cloudscape™ is not supported if you plan to use Annotations or alerts.

The primary advantages of database persistence is that it guarantees that changes to alert data are performed in the context of a database transaction and that changes are immediately propagated to all portlets whose Alerts engines use the database persistence manager.

The length and size of fields all have default values. By default, entries such as alert definition ID have a limit of 128 characters. You can view all of these defaults in the creation SQL script, which is located in:

WebContent/WEB-INF/solutions/alerting/db_persistence/\<database\>_alerting_tables.sql.

**Tip:** Database persistence is the recommended choice in a production environment.

**XML persistence**

XML persistence stores the persistence as XML files on the file system. The disadvantage of this method is that the manager does not guarantee that changes to alerting data are
performed in the context of a transaction. Also, any changes by one manager are not shared with others unless there is a shared file system.

The advantage of XML persistence is that this is the default persistence manager and is preconfigured without any further configuration steps. This makes it an excellent choice for development purposes.

**Note:** XML persistence is not recommended for production.

### Migrating alerts from development to production

It is normal practice to have XML persistence in a development environment but to have database persistence in a production environment. WebSphere Dashboard Framework provides you with a tool to help you migrate the alerts from one environment to another.

The Import button moves your configuration from your local XML store into the database. The Export button takes the configuration stored in the database and moves it to your local XML repository.

After you select the alert feature set, the Import Definitions portlet is automatically added and provides an easy to use interface as shown in Figure 6-2.

![Image of Import Definition portlet](Figure 6-2 Import Definition portlet)

By default, the alert XML persistence files are located in:

```
<project_name>/WebContent/WEB-INF/solutions/alerting/xml_persistence/alert_defs.
```

If necessary, you can change this by adding a file called override.properties in the

```
<project_name>/WEB-INF/config folder and adding the following:
```

```properties
bowstreet.solutions.alerting.xmlFilePersistenceManager.alertDefFilePath=${Designer.rootDir}
<alert_defs_path>
```

where `<alert_defs_path>` is the path to which you want the XML to persist.
6.2.2 Step two: Adding alert features to your project

Next, we need to add alerting to our main project. This adds the My Alerts portlet and Manage Alerts portlet to the project, as well as the builders that you need to make your data available for use by the Alerts engine.

To add the features, right-click on the name of your project and select properties. Next, expand the WebSphere Portlet Factory and select the Feature Info link.

You need to check:
- Dojo Ajax:
  - Dojo Extensions
- WebSphere Dashboard Framework:
  - Alerts Module
  - Alerts Module Examples (Optional)

Click OK and the features are added to the project and deployed if you have automatic deployment enabled.

6.2.3 Step three: Setting up the database

We wanted to show you the use of annotations, which we show in Chapter 7, “Annotations” on page 179. For that reason, we used DB2 as our data persistence layer. Setting up persistence for Oracle and Microsoft SQL is extremely similar. Refer to the product Help for specific details.

Create a database
There are several features in WebSphere Dashboard Framework that rely on a data store. These features include the sample applications, team calendar, annotations, and alerts. There is a lot of flexibility in defining the setup for these data stores. We recommend that you talk to your database administrator and decide the best combination of databases for your organization. We guide you through one way to set up the database.

First, create the database in DB2 by opening the DB2 Control Center.

Navigate to Instances → DB2 → Databases. Then, right-click Create Database → Standard. Refer to Figure 6-3 on page 152.
Next, give your database a valid name. We called our database WDF_PROD. Click Finish.

**Connect to the database**

In the Control Center, navigate to the database that you just created in the Object View pane, right-click and click connect. Enter your administrative user name and password and click OK.

**Creating the tables**

After you have a connection to the database, you can set up the tables that are required by the alert module. Open the project that you created in 6.2.6, “Step six: Creating an Alerts Scheduler project” on page 154 and navigate to: `<ProjectName>/WebContent/WEB-INF/solutions/alerting/db_persistence/`. In this directory, locate a text file called `db2_alerting_tables.sql`. Open the file and copy the contents onto the clipboard.

Go back to the DB2 Control Center, and in the same window where you connected to the database, you now see the Query link in the bottom right view. Click that link and it takes you to the query window. See Figure 6-4 on page 153.
Next, paste the contents of the clipboard, which are the contents of db2_alerting_tables.sql, into the top pane and click Execute. See Figure 6-5.
Ensure that you see messages, such as “The SQL Command Completed Successfully,” in the bottom pane.

6.2.4 Step four: Setting up a data source

The next step in the process is to set up a connection so that our WebSphere Portal server can connect to the database. For the steps to create a data source, see Figure 3-4 on page 56.

6.2.5 Step five: Configuring your projects to use database persistence

The next step is to set up an override.properties file. This file supersedes any default properties that have been set up by the product.

To set up an override.properties file, create a text file in the `<project_Name>/WEB-INF/config` called `override.properties`. Within this text file, add the lines in Example 6-1 and adjust the last line if you have specified a different JDBC name.

Example 6-1 Entries for override.properties

```properties
bowstreet.solutions.alerting.persistenceManager.implementation=com.bowstreet.solutions.alerting.impl.database.DatabasePersistenceManager
bowstreet.solutions.alerting.databasePersistenceManager.idFactoryClass=com.bowstreet.solutions.alerting.impl.database.DB2InstanceIDFactory
bowstreet.solutions.alerting.databasePersistenceManager.dataSourceName=jdbc/v5/dashboad
```

Tip: If you want to enable caching at this stage, enter these lines:

```
bowstreet.solutions.alerting.databasePersistenceManager.enableCaching=true
bowstreet.solutions.alerting.databasePersistenceManager.cacheTimeoutInterval=60
```

6.2.6 Step six: Creating an Alerts Scheduler project

The next step is to create a WebSphere Dashboard Factory project that supports the Alerts Scheduler feature, which is the application that controls when alerts are scheduled and generated.

Create a project that includes the Alerts Scheduler feature in the WebSphere Dashboard Framework tooling. This project also needs to include the Alerts Module and Dashboard Framework features at a minimum. When you get to the Deployment Info dialog, you must select WebSphere Native (deprecated) as the application programming interface (API) to use. Then, deploy the project to your applications and WebSphere Portal server. If you have automatic deployment enabled, this happens automatically for you.

Refer to 2.3, “Creating a WebSphere Dashboard Framework project” on page 22 if you need to refresh your memory about how to create a project.

Important: You must select WebSphere Native (deprecated) as your Portlet API for the scheduler project.
Next, you need to add the override.properties file from 6.2.5, “Step five: Configuring your projects to use database persistence” on page 154 to the Alert Scheduler project. Just copy and paste the same file into this project. This ensures that both projects use the same persistence and therefore know about each other.

**Important:** From this point forward, copy all alerting code from the original project to the scheduler project, including the models providing alert data, evaluator models, evaluator Java classes, and Linked Java Object (LJO) classes. The two projects must contain identical alert information.

### 6.2.7 Step seven: Configuring e-mail and SMS

To configure e-mail, edit this XML file:

```xml
<project_name>\WebContent\WEB-INF\solutions\alerting\xml_persistence\notifier_defs\email.xml
```

You need to locate and change the following values:

- The XML that starts with `<Enabled>`, change to true.
- The XML that starts with `<Name>EMail.mail.host</Name>`, change the value to the fully qualified domain name of your mail host.
- The XML that starts with `<Name>EMail.from.address</Name>`, change the value to the user from whom you want the e-mail to come.

If your e-mail system requires a user name and password to send mail, you need to change the following values:

- The XML that starts with `<Name>EMail.mail.user</Name>`, enter your mail systems user name.
- The XML that starts with `<Name>EMail.mail.password</Name>`, enter the password for your mail system.

You then need to save and redeploy the Web archive (WAR) file.

Then if you use DB2, use the Import Definitions portlet on the **Notifier Definitions** tab to use the import functionality to move the e-mail configuration from your development environment to your production environment.

After this is complete, the user must enter the user e-mail address in the personalization area of the My Alerts portlet.

After everything is configured, you see e-mails, such as the e-mail in Figure 6-6 on page 156.
You can change the e-mail content if required. Refer to the product Help for instructions.

Configuring for Short Message Service (SMS) notification is similar to configuring e-mail notifications. However, the SMS implementation uses a third-party Web service provided by:

http://vsmsgateway.com

You need a license from this site to successfully use the SMS notifier. The license key is used to provide the value of the SMS.license.key property in the sms.xml file in the WEB-INF\solutions\alerting\xml_persistence\notifier_defs folder in your Web application project.

6.3 Configuring data for alerts

After the Alerts module is configured, the next step is to expose particular attributes of your data to the alerting engine. This requires a decision about what sort of data you want to make available. Consider the business rules that you anticipate to be part of your application and the data and services that are required to implement those rules. It is beneficial to consider these implications up front when designing an application.
6.3.1 Alert data

To begin using the Alerts module, we need to make our data available for use when creating alert definitions. To achieve this, the Alert Data builder is added to a Service Provider model. For our example, we use a service provider model that fetches data from DB2. This is the service provider model to which we add the Alert Data builder. In our example, the Alert Data Method input is simply a call to an Action List builder that executes our data service. The Alert Data Row input is where we specify the schema row of our data returned by the Alert Data Method. Below these inputs is a list where we define our parameters. The Name section is where we defined the name of our parameter. For our purposes, we named our parameters similarly to the attribute name in the database. The schema path is the fully qualified path to the schema element value associated with the alert parameter. Figure 6-7 displays the inputs of the Alert Data builder that we created.

![Figure 6-7 Alert Data]

*Alert Data*

Enables you to identify data in a Service Provider model and make it available for use in alerts. Must be used in conjunction with the Service Provider builder and the alerts module.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>SalesKPIAlertData</td>
<td></td>
</tr>
</tbody>
</table>

*Discrete Alert Parameters Selection*

This section is used to identify discrete values that will be passed to the alert evaluator.

Discrete values are ones that are not included in the main table of data. If you are defining data for an existing alert, then select the alert parameters and assign them default values. The alert parameters are named AlertName:ParameterName.

If you are planning to define alerts later, type in the parameter name and assign it a default value. Be sure to use descriptive names since they will be used later to create alerts.

*Table-based Alert Parameters Selection*

This section is used to identify the schema path for all of the table-based data that will be passed to the alert evaluator. The Alert Data method is typically one that is also published with a Service Provider builder.

If you are defining data for an existing alert, select the alert parameter and assigning it a default value. The alert parameters are named AlertName:ParameterName.

If you are planning to define alerts later, type in the parameter name and assign it a default value. Be sure to use descriptive names since they will be used later to create alerts.

<table>
<thead>
<tr>
<th>Alert Data Method</th>
<th>Alert Data Row</th>
</tr>
</thead>
</table>
| getSaleKPIsSchema | getSaleKPIsSchema/Row/

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales_KPI_Percent</td>
<td>getSaleKPIsSchema/Row/PERCENT</td>
</tr>
<tr>
<td>Sales_KPI_Quarter</td>
<td>getSaleKPIsSchema/Row/QUARTER</td>
</tr>
<tr>
<td>Sales_KPI_State</td>
<td>getSaleKPIsSchema/Row/OFFICE_STATE_SHORT</td>
</tr>
</tbody>
</table>
After you finish defining the Alert Data builder in your service provider model, you must copy your model to the Alert Scheduler project. This makes the alertable data available to the Alerts module. Figure 6-8 displays the Alert Data parameters, which we created by using the Alert Data builder, which is available for use when creating an alert definition.

![Alert data parameters](image)

6.3.2 Alert Customizer

When you create your project and add the Alert feature, the Manage Alerts and My Alerts portlets are made available in your project. The My Alerts portlet uses a customizer model, My Alerts Customizer, to allow the user to configure many inputs of the portlet. For our example, we use the My Alerts and My Alert Customizer models to discuss the Alert Customizer builders and the additional builders used to enable customization. To get started, you must profile the inputs that you want the user to be able to configure. Figure 6-9 on page 159 shows all of the inputs of the My Alerts model that were profiled to allow for customization.
In the Portlet Customizer builder, you specify which model to customize. Here, you also specify which profile set to use. After a profile set is specified, the list of available inputs is provided to you. You can specify which inputs are on which page. Figure 6-10 on page 160 displays the inputs used for this builder.
Figure 6-10  My Alerts Portlet Customizer
The Alert Customizer builder exposes the user’s threshold, subscription, and notification settings in the edit or configure option of a portlet. Using this builder, you can enable any or all of the settings to expose. For example, you might not enable the customization of thresholds so that the user cannot configure these settings. The Customizer Page Input is where you specify on which page to place the Threshold, Subscription, or Notification settings. The Imported Page input is populated by default with the value of /solutions/pages/alerting/alertCustomizerDefault.html, but you can also change this value. Figure 6-11 displays an example of the inputs for the Alert Customizer builder.

![Alert Customizer](image)

In the Portlet Adapter builder, you specify how the profile set in the model is accessed to customize when it is in WebSphere Portal. The Custom Edit Type input must have a selected option of Custom Model. This presents a Custom Edit Model entry where you can specify the Alert Customizer model. Figure 6-12 on page 162 displays the inputs for the Portlet Adapter builder.
After you complete these steps, customization of the My Alerts portlet is available to the user in WebSphere Portal.

### 6.4 Creating an alert definition with the Manage Alerts portlet

There are several ways to create an alert instance and the method that you choose depends on the complexity of the business rule that you need to implement. WebSphere Dashboard Framework provides you with the flexibility to meet these rules by providing:

- **Script-based alerts**
- **Java-based alerts**
- **Model-based alerts**

You can also use *external* alerts. These are alerts that rely on an external system to push the alerts to the Alerts engine. A common use case is a Web service that produced an alert. This is an extremely powerful integration option but is beyond the scope of this IBM Redpaper. See the product Help for more details.
6.4.1 Script-based alerts

The quickest way to create an alert is to use the Manage Alerts portlet and select the alert definition option Script-based. Here, you define the Alert script logic using a wizard interface. Also, there are data parameters available that were made alertable, which we described in 6.3, “Configuring data for alerts” on page 156. Select the New Alert button on the Manage Alerts portlet to get started.

Figure 6-13 shows the initial step required to create a script-based alert. Here, we define the Alert name and create a new category.

![Figure 6-13 Alert Identity](image)

In the Type section of the Manage Alerts portlet, you specify the type of alert definition to create. For this example, which is shown in Figure 6-14, we select the option **Script-based**.

![Figure 6-14 Alert Type](image)

You can specify the period used to set the alert creation and expiration dates in the Expiration section. The available options for this input are:

- Minute
- Hour
- Day
- Week
- Month
- Quarter
For our example, we select the option of **Minute**. This allows us to see the results of our alert definition immediately. Figure 6-15 is an example of the Expiration input window.

In the Logic section, you create the script logic. Here, you define the Alert active expression and the Alert priority expression. The Alert active expression defines when an alert is made active and displayed to the user. The Alert priority expression defines when to adjust the priority of an alert. The Logic user interface makes available to you a list of available parameters, operators, and functions to assist in building the alert script. Figure 6-16 displays an example of the Logic section with sample alert expressions.
In the Alert Display section, you define the displayable text of the alert. Here, you define the Summary Text and Detail Text of the alert. The summary text displays in the list view of the My Alerts portlet. The detailed text displays when the user clicks to drill down into more detail of the alert. You are given a list of available parameters, operators, and functions to assist in creating the displayable text. Figure 6-17 shows an example of inputs for the Display section.

![Alert Display Diagram](image)

**Figure 6-17 Alert Display**

In the Alert Feature section, you select the features to be used with the alerts generated from this definition. The available features are:

- Alert escalation: Certain business performance indicators or critical failure conditions are so important to an organization that increasing levels of management need to be notified if these conditions persist. For example, if a customer service request by a key customer or business partner exists for more than four hours without resolution, the Customer Service Manager probably needs to be notified. Escalation of this type of alert can help a business ensure that an important opportunity does not pass by without devoting the proper amount of attention to it.

- Automated notification (e-mail, SMS, and so forth): Alert definitions can be configured to support automatic notification to alert subscribers. The schedule for notification is specified in the Features section of the alert definition. Notifications can be scheduled to happen every minute, hour, day, week, or month. After they are scheduled, notifications are dispatched automatically for all alerts that are active when the scheduled time expires.

- User authorization: To restrict access to alerts, you need to select people for the User authorization feature in an alert definition. The people selected here represent the collection of users, who are authorized to subscribe to this definition’s alerts. For these users to receive these alerts in the My Alerts portlet, they must enable the Subscribe option in the Subscriptions tab in the personal settings for the My Alerts portlet. Users,
who are not listed in the User Authorization window, cannot subscribe to this definition’s alerts. If this feature is not enabled, all users are authorized to subscribe to this definition’s alerts.

- Related bookmark: The bookmark adds a link to an Alert Detail page, which can go to a WebSphere Portal page, a Web site, or a file. Add a related bookmark to an alert definition to help users resolve a problem or find information related to the alert. The link that you define in the alert definition is added to the Alert Detail page that a user sees in the My Alerts portlet.

Figure 6-18 displays an example of the Alert Features section.

If the Alert Escalation feature is selected, the Escalation section is made available to you. In the Escalation section, you can define the escalation approach and acknowledgement handling. Escalation is used by the system to assign owners to an alert. Figure 6-19 on page 167 displays an example of the Alert Escalation inputs.
If the User Authorization feature is selected, the user authorization section is made available to you so that you can select the users and groups that are authorized to view the alerts generated by the definition. Figure 6-20 displays an example of this section.

If the Automated Notification feature is selected, you have access to a section where you can define the notification schedule of the alert.
The available options are:

- At the start of every minute
- At the start of every hour
- Daily at 12:00AM
- Weekly at 12:00AM every Sunday
- Monthly at 12:00AM on the last day of the month
- Quarterly at 12:00AM on the last day of the quarter
- Yearly on the first day

For our testing purposes, we selected the option **At the start of every minute**, which makes testing the notification of the alert easier and quicker. Figure 6-21 displays an example of the Automated Notification section.

![Automated Notification]

**Figure 6-21  Alert Automated Notification**

The final step to create a script-based alert is the Review section. Here, you are presented with all of the details of the alert that you have just created. If changes need to be made, you can simply click the Back button or go directly to the section that you want to edit by selecting the section on the left. Figure 6-22 on page 169 displays an example of the Review section of the script-based alert that we have created.
After you select **Finish**, the newly created Alert displays in the Manage Alerts portlet, and the My Alerts portlet displays Sales key performance indicator (KPI) alerts. This alert immediately displays, because we selected the alert expiration period of **At the start of every minute**. Ideally, in production, you need to select a more meaningful alert expiration period. Figure 6-23 on page 170 displays the Manage Alerts and My Alerts portlets with the alert definition that we just created.
6.4.2 Java-based alerts

A Java-based alert is an alert definition that uses a class-based evaluator. This class-based evaluator has four specific requirements:

- The evaluator class must be declared public and must be derived from the interface AlertEvaluator in the package com.bowstreet.solutions.alerting.impl. This interface defines the method named “evaluate” that is invoked by the Alerts engine when it evaluates the state of alerts.
- The class must define a public no-argument constructor. This constructor is used by the Alerts engine to create instances of the evaluator class.
- The evaluate method must return a list of zero or more object instances derived from GenericAlert in the package com.bowstreet.solutions.alerting.impl.
- When deployed, the class must be on the Factory's classpath so that it can be loaded by the Alerts engine at run time. The Factory's classpath includes /WEB-INF/work/classes, /WEB-INF/work/lib, /WEB-INF/lib, /WEB-INF/classes, and the application server's classpath. If the evaluator class is not located in one of these places, then evaluation of the alerts associated with the definition fails at run time. The Alerts engine creates an instance of the class each time that it needs to evaluate alerts. If the class cannot be loaded or it does not implement the AlertEvaluator interface, alerts for the associated definition are not evaluated and the Alerts engine logs an error.

Figure 6-24 on page 171 displays sample code for a stub evaluator class.
For our example, we use the sample class `Example_JBE` that generates a simple alert that appears in the My Alerts portlet. This class is a great starting point when developing your own custom Java-based evaluator class. Figure 6-25 on page 172 displays the code from `Example_JBE`.

```java
package com.mycompany.alerting.evaluators;

import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.Map;

import com.bowstreet.solutions.alerting.AlertDialogException;
import com.bowstreet.solutions.alerting.impl.AlertEvaluator;

public class ThresholdEvaluator implements AlertEvaluator {
    public ThresholdEvaluator() {
        super();
    }

    public List evaluate(String alertID, String categoryID, Locale locale, Map properties) throws AlertDialogException {
        return new ArrayList();
    }
}
```

Figure 6-24  Stub evaluator class

For our example, we use the sample class `Example_JBE` that generates a simple alert that appears in the My Alerts portlet. This class is a great starting point when developing your own custom Java-based evaluator class. Figure 6-25 on page 172 displays the code from `Example_JBE`.

```java
package com.mycompany.alerting.evaluators;

import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.Map;

import com.bowstreet.solutions.alerting.AlertDialogException;
import com.bowstreet.solutions.alerting.impl.AlertEvaluator;

public class ThresholdEvaluator implements AlertEvaluator {
    public ThresholdEvaluator() {
        super();
    }

    public List evaluate(String alertID, String categoryID, Locale locale, Map properties) throws AlertDialogException {
        return new ArrayList();
    }
}
```

Figure 6-24  Stub evaluator class
package com.boulevard.solutions.alerting.samples;

import java.util.ArrayList;

/**
 * Sample class-based alert evaluator.
 */

public class Example_JBE implements AlertEvaluator {

    private static final String FILE_NAME = "com.boulevard.solutions.alerting.samples.messages";
    private static final String SAMPLE_DISPLAY_NAME = "Sample Display Name";
    private static final String SAMPLE_TEXT_MESSAGE = "Sample Text Message";

    public Example_JBE() {
    }

    protected GenericAlert createAlert(String alertID, String categoryID, Locale locale, Map properties) {
        GenericAlert alert = new GenericAlert();
        Locale localeForBundle = (locale == null) ? Locale.getDefault() : locale;
        ResourceBundle bundle = ResourceBundle.getBundle(FILE_NAME, localeForBundle);
        this.getMessage().getLocalizedMessage();

        // The creation date of an alert class instance is the date on which the logical alert
        // is represented is considered to be created in the system. For example, a Sales alert
        // that is based on last month's sales data would have a creation date set to the last
        // day of the current month. That is, instances of this alert that use sales data from
        // April 2006 would have a creation date of 01 May 2006 rather than the actual system
        // time the alert class instance was created. The code below uses current system time
        // to demonstrate several features of the AlertEngine: it is not intended to represent
        // best practice in setting an alert's creation date.

        // The expiration date of an alert is used by the Alerts Engine to determine whether the
        // alert instance is still active and needs to be re-evaluated. In the code below we are telling
        // the Alerts Engine that this instance is only valid 1000 milliseconds from the date above
        // it was created. If configured to cache alert instances, then Alerts Engine will cache
        // this instance for no more than 10 seconds.

        Date currentDate = new Date();
        alert.setCreationDate(currentDate);

        // We handle the version ID specially since it's used by the Alerts Engine to track an
        // alert instance. The standard for creating a version ID is to create a string that
        // identifies a particular alert version across time and space. The creation date takes
        // care of identifying the first part (it takes into account current time and the period
        // over which business data remains constant). The final piece is made up of the
        // property values used to generate the alert. We are guaranteed that the properties
        // passed into this method are not null we can simply add them to the end of the
        // version ID string.

        final String evaluatedVersionID = alert.getCreationDate().toString();

        // Create a mapping of priority to color that can be used by the portal to create a
        // richer visual display of this alert. Note that the portal may receive this map as
        // a string if this evaluator is called for as a service-based alert. As an alternative
        // you could pass XML back and reconstruct it in the portal.

        Map colorMap = new HashMap();
        colorMap.put("1", "red");
        colorMap.put("2", "yellow");

        // Set the alert name, category, and text to be a format appropriate to the user based
        // upon their preferred locale. Note that we’re using the locale that was passed in to
        // the evaluate method (this is the user's preferred locale). Note also that we use the
        // getBundle method form that takes a classloader. This is necessary since this class
        // may have been loaded by the Bundle framework class loader.

        alert.setDisplayName(bundle.getString(SAMPLE_DISPLAY_NAME));
        alert.setDisplayCategory(bundle.getString(SAMPLECATEGORYID));
        alert.setDisplayText(bundle.getString(SAMPLE_TEXT_MESSAGE));
        alert.setDetailsDisplayText(bundle.getString(SAMPLEDETAILS_TEXT_MESSAGE));

        return alert;
    }

    // ... more code...
}

// Figure 6.25 Example JBE

172 Building Composite, Role-Based Dashboards with WebSphere Dashboard Framework
When creating a Java-based alert definition, begin by clicking **New Alert** on the Manage Alerts portlet. The window appears to define the alert name and either select the category or create a new category. For the alert definition type, select the **Custom** option. This option presents you with the window to input the name of the Java class. Figure 6-26 displays an example of this input.

![Figure 6-26 Java Custom Evaluator](image1)

After you have reached the Review step, click **Finish** and view the alert generated. Figure 6-27 displays the review step of the Java-based alert creation.

![Figure 6-27 Alert Review](image2)

An alert now displays in the My Alerts portlet. This is shown in Figure 6-28 on page 174.
6.4.3 Model-based alerts

A model-based alert is an alert definition that uses a WebSphere Portlet Factory model-based evaluator. This class-based evaluator has two specific requirements:

- The model must define a public method named “evaluate” that returns an instance of java.util.List. Furthermore, the list must contain zero or more object instances derived from com.bowstreet.solutions.alerting.impl.GenericAlert.

- The evaluate method must take four parameters: an Alert ID, a Category ID, a locale, and a set of properties. Both IDs are java.util.Strings, the locale is of type java.util.Locale, and the properties are an instance of java.util.Map. The Alerts engine creates an instance of the model and invokes the method when it needs the model to evaluate alerts associated with the definition. If a method of the specified signature cannot be found in the model, then alerts for the definition cannot be evaluated and the Alerts engine logs an error.

To create a model-based alert, you use a model that is created in Portlet Factory containing an Action List and Linked Java Object builder. The LJO builder uses the Java class from the previous section, 6.4.2, “Java-based alerts” on page 170. Of course, additional builders can be used in this model for any additional customization.

For our example, we created an empty model named ThresholdEvaluator. Next, we add a Linked Java Object builder. This is the builder that exposes the methods of the Java class that we created in the last section to our model. Figure 6-29 on page 175 displays the inputs of the Linked Java Object builder used in our model.
Next, we add an Action List builder to our model. This action list is simply for calling our evaluate method of our Java class. The parameters defined are required to be the same as the parameters of the evaluate method of the Java class. This action list's return type must be java.util.List. Figure 6-30 on page 176 displays the action builder that we created.
We are ready to create a new alert using the Manage Alerts portlet. We have named our alert Model-Based Alert and created a new category of Model-Based Alert. We selected an alert evaluation type of custom to give us access to the Custom Evaluator window. Here, we specify the model that we use. Figure 6-31 on page 177 displays the inputs that we have used for this window.
Additional inputs can be specified for the next steps of creating the alert. For this example, we specified automatic notification to occur every minute. Figure 6-32 displays the review window of our alert definition.

After we select **Finish**, our model-based alert displays in the My Alerts portlet. Figure 6-33 on page 178 shows the model-based alert displayed in the My Alerts portlet.
### My Alerts

<table>
<thead>
<tr>
<th>Priority</th>
<th>Text</th>
<th>Creation Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Annotation Alert (1 alert)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Java Based Alert (1 alert)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Model Based Alert (1 alert)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sales KPI (2 alerts)</td>
<td></td>
</tr>
</tbody>
</table>

**Model Based Alert**

*Figure 6-33*  Model-Based Alert
Annotations

Today, collaborative and interactive Web applications are pervasive. Online book sellers allow customers to rate and write reviews for books; potential book buyers can use this information when contemplating a purchase. News Web sites allow users to add their opinions and comments to news articles. People post comments to blogs, discussion forums, and digital images. The examples are endless.

In this chapter, we demonstrate how you use Annotations to add similar collaborative capabilities to the charts and tables in your dashboard application.

We discuss the following topics:

- Overview of annotations
- Configuring annotation persistence
- Annotating charts and tables
- Enabling annotations
- Configuring alerts for annotations
- Annotations administration
7.1 Overview of annotations

As you have seen in previous chapters, provider models allow you to retrieve data from a variety of repositories; then, you can use this data in tables and charts in dashboard consumer models.

Building upon this functionality, you can use WebSphere Dashboard Framework’s Annotations feature to add collaborative functionality to your dashboard application. Specifically, users can:

- Create comments to charts, tables, and table rows
- View comments
- Respond to comments

Additionally, users can attach files to the comment, send alerts for the comment to other users, and control who can view the comment.

For example, Figure 7-1 shows a window that illustrates adding a comment to a chart.

![Figure 7-1 Adding a comment to a chart](image)

Users can also view the comments added to charts and tables. Figure 7-2 on page 181 shows comments added to a chart.
7.2 Configuring Annotations persistence

Annotation data can be stored on the file system in XML format or in a relational database. The persistence method that you choose depends on whether you are in a development or production environment.

7.2.1 XML persistence

By default, Annotations data is stored in XML format. The Annotations XML data is stored in the WEB-INF/solutions/annotations/xml_persistence/annotations directory.

This directory can be changed by using the:
ibm.wdf.annotation.xmlFilePersistenceManager.annotationFilePath property.

This method is well-suited for prototyping during development. For production environments, we recommend configuring relational database persistence, described in section 7.2.2, “Relational database persistence” on page 181.

Note: XML persistence is best-suited for development and demonstrations. For production environments, we recommend that you use relational database persistence.

7.2.2 Relational database persistence

Relational database persistence is recommended for production environments. Annotations data can be stored in DB2, Oracle, or Microsoft SQL Server™.

Note: Cloudscape is not supported for Annotations persistence.
To model a production environment, we configured relational database persistence with IBM DB2 V9.1. This required setting several configuration properties, configuring a database, and creating a JDBC resource and Data Source in WebSphere Application Server.

**Configuring Annotations properties**

Relational database persistence requires adding or modifying several properties, which are described in the WebSphere Dashboard Framework Help. Example 7-1 shows the DB2-specific properties that we added to WEB-INF/config/override.properties.

---

**Note:** WebSphere Dashboard Framework or Portlet Factory upgrades can overwrite changes in bowstreet.properties. As a best practice, we recommend adding these properties in WEB-INF/config/override.properties to prevent your losing Annotation properties. You might have to create this file if it does not exist.

**Example 7-1 Database persistence properties added to override.properties**

```properties
ibm.wdf.annotation.persistenceManager.implementation=com.ibm.wdf.annotation.impl.database.DatabasePersistenceManager
ibm.wdf.annotation.databasePersistenceManager.idFactoryClass=com.ibm.wdf.annotation.impl.database.DB2InstanceIDFactory
ibm.wdf.annotation.databasePersistenceManager.dataSourceName=jdbc/v5/dashboard
bowstreet.upload.enabled=true
ibm.wdf.annotation.enableSecurity=true
```

---

**Important:** Be certain that there are no spaces at the end of entries in the properties file, especially for the `ibm.wdf.annotation.databasePersistenceManager.idFactoryClass` and `ibm.wdf.annotation.persistenceManager.implementation` properties. The spaces are easy to miss and can cause the application to fail.

The first two properties define Java classes that WebSphere Dashboard Framework uses for DB2 persistence. The next property specifies the JDBC data source name that we create in the subsequent section “Configuring JDBC resource and data source” on page 183.

In addition to comments, the Annotations function allows you to attach files to the responses. This functionality relies on Portlet Factory's File Upload builder. For security, File Upload is disabled by default. Therefore, we set `bowstreet.upload.enabled = true` in our override.properties file. If this is not set to true, you receive an error message at run time stating “File upload is not enabled.”

**Note:** The `bowstreet.upload.enabled` property must be set to true. If not, an error occurs at run time when commenting on a table or chart.

Finally, we enabled access control for Annotations by specifying `ibm.wdf.annotation.enableSecurity=true`. This setting applies to the Privacy Settings section of the Annotations user interface, described in 7.3.1, “Annotations user interface” on page 183, where users can control who is allowed to view the comment.
This property must be set to true; otherwise, the Annotations engine does not provide access control to comments, and all users can view comments, even if only certain users have been specified in the Privacy Settings.

**Note:** The `ibm.wdf.annotation.enableSecurity` property must be set to true to enable access control for Annotations.

After we configured our application to store Annotations data in DB2, we configured a database to store Annotations data.

**Configuring the database**

We used the DB2 Control Center to create a database to store Annotations data. WebSphere Dashboard Framework includes a SQL script that creates the necessary annotation-specific tables for each supported relational database. Because we use DB2, we ran `db2_annotations.sql`.

**Configuring JDBC resource and data source**

After we configured the Annotations properties and the database, we used the WebSphere Application Server Administration console to configure a JDBC resource and data source so that our application can interact with the database. We configured the data source with the same name values as the `ibm.wdf.annotation.databasePersistenceManager.dataSourceName` property in Example 7-1 on page 182.

### 7.3 Annotating charts and tables

Section 7.1, “Overview of annotations” on page 180 provided a brief overview of adding a comment to a chart. This section describes each field in the Annotations user interface and provides additional detail about annotating charts and tables.

#### 7.3.1 Annotations user interface

The Annotations user interface, shown in Figure 7-3 on page 184, is common regardless of whether you annotate a chart, a table, or a table row.
Headline and comment fields
Similar to the subject line of an e-mail or forum post, the Headline field is used to enter a brief one line headline of the comment.

Similar to the body of an e-mail or forum post, the Comments field allows the user to enter comments to the chart or table. See Figure 7-4.

Attachments
As shown in Figure 7-5, the Attachments section allows the user to add attachments to the annotation.
The attachment functionality relies upon Portlet Factory’s File Upload builder. For security, file uploading is disabled by default.

**Note:** Enabling file uploading and using the attachments functionality require that `bowstreet.upload.enabled=true` is added to the project’s override.properties file.

### Alerts

The Alerts section, shown in Figure 7-6, allows the user to:

- Send an alert for the comment to a group of users.
- Request a response to the comment from a group of users.

![Alerts](image)

*Figure 7-6   The Alert section allows the user to send alerts or request responses for comments added with the Annotations builder*

The Find People button presents the Directory Search application, shown in Figure 7-7 on page 186, which allows the user to search for and select users and groups to whom the annotation alert is sent.
Figure 7-7  Clicking Find People presents the Directory Search application to search for and select users to whom to send the alert

Users, who were selected to receive the alert, can view the alert in the Annotation Alerts section of the My Alerts portlet, as shown in Figure 7-8.

Figure 7-8  An Annotations Alert in the My Alerts portlet

**Note:** Alerting for Annotations requires configuration. See 7.5, “Configuring alerting for annotations” on page 198.

**Privacy settings**
This section, shown in Figure 7-9 on page 187, allows the user to control who can see the comment. The default is that anyone can see the comment.
If **Only certain individuals should be able to see this comment** is selected, **Find People** displays. If you click this button, the Directory Search window appears, allowing you to search for and select the users who can view the comment. The Directory Search window is shown in Figure 7-7 on page 186.

### 7.3.2 Commenting charts

Charts can be added to your application with the Web Charts - Enhanced builder or with high-level dashboard builders, such as the Map builder or the Summary and Drilldown builder. The Annotations function allows users to comment on these charts.

For example, the Office Sales KPIs portlet in our application contains a chart created by the Status Page builder. Users can comment on the chart by selecting **Make a comment** from the **Actions** menu, as shown in Figure 7-10.

When you select **Make a comment**, the Annotations user interface displays to the right of the chart, as shown in Figure 7-11 on page 188.
7.3.3 Viewing and responding to chart comments

Selecting the View all comments to the graph hyperlink above the chart or selecting View comments from the Actions menu, shown in Figure 7-12 on page 189, allows the user to view the chart comments. Note that if no comments exist for the chart, the links and actions to view the chart comments do not display.
Figure 7-12  Users can view the comments to a chart by selecting View comments from the Actions menu or selecting the View all comments to the graph link above the chart.

The Chart Comments display to the right of the chart. Comments have a blue background and responses have a gray background. Figure 7-13 shows a comment and a response comment to the chart in the Office Sales KPIs portlet.

Figure 7-13  A comment and response comment for the dial chart.
The user can respond to a comment by single-clicking the comment headline and selecting **Respond** from the menu, as shown in Figure 7-14.

![Figure 7-14](image)

**Figure 7-14** Users can respond to comments by clicking the comment headline and selecting **Respond** from the menu.

### 7.3.4 Commenting tables and table rows

In addition to charts, Annotations can be enabled for tables and table rows. When you configure Annotations for a table or table rows, the **Actions** menu contains a **Make a comment** action. For example, to annotate a table or table row in the Sales KPIs portlet of our application, **Make a comment** can be selected from the **Actions** menu as shown in Figure 7-15.

![Figure 7-15](image)

**Figure 7-15** To annotate the table or a table row, select **Make a comment** from the **Actions** menu.

The Annotations user interface displays to the right of the table, as shown in Figure 7-16 on page 191. By default, Annotations assumes that you annotate the entire table; this is indicated by the yellow line surrounding the table.
If row annotation is enabled, you can annotate a specific row in the table by single-clicking that specific row. As shown in Figure 7-17, the row is highlighted and the user can enter comments for the specific row.

When a row is highlighted, clicking the row again disassociates the comment with the row, allowing you to comment on the entire table instead.

7.3.5 Viewing and responding to table and table row comments

Selecting the View all comments to the table or table data hyperlink above the table or selecting View comments from the Actions menu allows the user to view table and table row comments in our application, as shown in Figure 7-18 on page 192.
When comments exist for a table or table row, the user can view comments by selecting the link above the table or selecting View comments from the Actions menu. Note that if no table or table row comments exist, the View comments action does not display.

If there are comments to the overall table, a yellow line surrounds the table and the table comments are shown to the right. Figure 7-19 shows that the table contains one comment.

By default, if comments to particular rows exist, a yellow sheet of paper icon appears at the beginning of the rows that contain comments. For example, the first table row in Figure 7-20 has a comment.

Clicking the yellow piece of paper icon displays the comments for the particular row as shown in Figure 7-21 on page 193.
Figure 7-21 Viewing the comments for a particular table row

The user can respond to a comment by clicking the comment headline and selecting **Respond** from the menu. See Figure 7-14 on page 190.

### 7.4 Enabling annotations

You can directly enable the Annotations functionality in certain dashboard builders or use the Annotations builder in your model. This section describes both methods.

#### 7.4.1 Enabling Annotations in high-level dashboard builders

The following high-level dashboard builders allow you to easily enable the Annotations functionality directly in the builder:

- Hierarchy Drilldown
- Map
- Record List and Detail
- Status Page
- Summary and Drilldown

For example, in our `SalesKPIs.model`, we used the Map builder to display Sales KPI in a chart and table on the page. In the Actions Menu Options section of the Map builder, we enabled Annotations for the chart, table, and table rows as shown in Figure 7-22 on page 193.

![Figure 7-22 Enabling chart and table annotations in the Map builder](image)
The required Table Annotation ID and Chart Annotation ID fields identify comments made to this particular map; we entered unique values for both fields.

The Table Annotations Text and Chart Annotations Text customize the text of the create comment action. When no values are entered, the default value is **Make a comment**.

Because we enabled chart annotations, an **Actions** button appears next to the map. Clicking **Actions** allows you to choose **Make a comment**, as shown in Figure 7-23.

![Figure 7-23 When Enable Annotation to Chart is enabled in the Map builder UI, the Actions menu contains an option to make a comment](image)

Because we enabled Annotations for table and table rows, an Actions menu also appears at the top right of the table, as shown in Figure 7-24 on page 194.

![Figure 7-24 When Enable Annotation to Table is selected in the Map builder UI, the Actions menu contains an option to make a comment](image)
7.4.2 Using the Annotations builder in a model

The previous section demonstrated how to enable Annotations functionality directly for charts and tables that are produced by selected high-level dashboard builders. You can also add Annotations functionality to tables and charts in your application by using the Annotations builder in your model. In addition to selecting a chart or table to annotate, the Annotations builder allows greater flexibility in customizing the Annotations look and feel, such as the create comment action text and control type for adding comments.

Although it is not used in our dashboard application, to illustrate this use case, we included a sample AnnotationBuilderExample.model, which is located in <projectName>/models/ITS0_Example.

The model retrieves and displays a list of orders for customer ID 1002 and renders them in a table on a page. Annotations allows the user to add comments to the table and the table rows.

The builder call list is shown in Figure 7-25.

![Figure 7-25 The builder call list for AnnotationBuilderExample.model](image)

To retrieve a table of data to annotate, we added a Service Consumer builder. As shown in Figure 7-26 on page 195, we selected the CustomerOrdersProvider model for the Provider Model input.

![Figure 7-26 The Service Consumer provides access to the services defined in the CustomerOrdersProvider provider model](image)

Next, we added a Page builder to create an HTML page that displays the orders.

Then, we added a Data Page builder. Data Page is a powerful builder that builds a structure based on XML data stored in a variable and renders the structure on a given page. Because the getOrdersByCustomerId service operation in the provider stores the results in a variable, we selected this results variable for the Variable input. For the Page in Model and Location for New Tags input, we selected the HTML page that we created before and selected the tag location where the table needs to be placed. See Figure 7-27.
Now that the model contained table data that we can annotate, we added an Annotations builder to the model, as shown in Figure 7-28 on page 196.
Because we were interested in annotating table data, we selected **Table** for the Annotation Type input. When annotating a table, you must select a value for the **Table Reference** input; this value specifies the table data that you plan to annotate.

The Icon Image input allows you to specify a custom image icon to use to denote the existence of a comment for a row. By default, the value is /solutions/images/wdf/annotation/Annotation.gif. As shown in Figure 7-20 on page 192 in Section 7.3.5, “Viewing and responding to table and table row comments” on page 191, this image is a yellow piece of paper icon located at the beginning of the rows that contain comments.

The **Control Type** input allows you to specify either a button, link, icon, or Action Menu control for users to select in order to add a comment. If Button, Link, or Icon is chosen, the builder allows you the flexibility to choose a tag location on a page to place the control. We chose **Action Menu**, which creates an action menu at the top right of our table.

The **Button/Link Text** input allows you to customize the button or link text for adding annotations. We entered the text, **Add a comment**.

And last, with the Service Consumer, Data Page, Page, and Annotation builders in place, we created an Action List to set the `getOrdersByCustomerId` input to customer ID `1002`, invoke the data service, and render our orders page. Because this Action List needs to be the entry point to this model, we named the Action List **main**. See Figure 7-29 on page 197.

![Figure 7-29 The Action List assigns the customer ID input, invokes the data service, and displays the orders page](image)

With the Action List in place, we can now run our model. As shown in Figure 7-30, the **Actions** Menu allows us to select **Add a comment**.

![Figure 7-30 Selecting Add a comment to comment on the table or a table row](image)

As with any table annotations, we can comment on the entire table or a row in the table. Figure 7-31 demonstrates adding a comment to the first row.
Selecting View comments from the Actions Menu allows us to view the table and the table row comments, as shown in Figure 7-32 on page 198 and Figure 7-33 on page 198.

7.5 Configuring alerting for annotations

As shown in Figure 7-6 on page 185, Annotations allows you to send alerts to users for comments. This capability relies on the WebSphere Dashboard Framework Alerts functionality. Before reading this section, we recommend reading Chapter 6, “Alerts” on page 147.
The Alerts functionality of Annotations requires several configuration tasks:

- Your project must include the Alerts Module and Dojo Extension feature sets.
- Alerts must be configured. We explain this task in Chapter 6, “Alerts” on page 147.
- If the Alerts function is configured for database persistence, the Alert Definition for Annotations must be migrated to the database.
- An Alert Definition must be configured for Annotations.

If you are unsure if your project includes the Alerts Module and Dojo Extension feature sets, right-click the project and select Properties. In the Properties window, select WebSphere Portlet Factory, then Feature Info. If these feature sets are not enabled, enable them as shown in Figure 7-34 on page 199.

**Figure 7-34** To use Alerts with Annotations, the project must include the Alerts Module and Dojo Extension feature sets

### Migrating the alert definition for annotations to the database

If the Alerts functionality is configured for database persistence, the alert definition for Annotations must be migrated to the database. Because our application is modeled as a production environment, we used the `ImportDefinitions.model` to migrate the default alert definition to the database. The `ImportDefinitions.model` is included with the Alerts Module feature set and is located in WEB-INF/models/solutions/alerting/admin.

After deploying this model to a page, we clicked **Import** for the alert definition entitled **Annotations Alert**. After clicking **Import**, a green check mark in the To Database column confirms that the alert definition was successfully migrated to the database. This is shown in Figure 7-35 on page 200.
Building Composite, Role-Based Dashboards with WebSphere Dashboard Framework

Figure 7-35   Use the Import Definitions model to migrate the Alert Definition for Annotations to the database

Configuring an Alert Definition for Annotations
As explained in Chapter 6, “Alerts” on page 147, an Alert Definition provides instructions to the Alerts engine about which data to monitor, how often to evaluate the data, which users can subscribe to the alert, and other instructions. Alert definitions are managed through the Manage Alerts portlet, which is included with the Alerts Module feature set.

WebSphere Dashboard Framework’s Annotations functionality requires a special Alert Definition. WebSphere Dashboard Framework includes a predefined Java-based Alert Definition for Annotations called Annotation Alert.

Although this predefined alert definition can be modified, for example, to change the notification schedule, we recommend duplicating it, making the desired changes to the duplicate alert definition, and disabling the predefined alert definition. Most of the inputs in the duplicated alert definition can be left to their default values.

Note: For safety, make a duplicate of the predefined Annotation Alert alert definition and modify the duplicate version. Then, disable the original predefined alert definition to prevent users receiving duplicate alerts.

As shown in Figure 7-36 on page 201, clicking Annotation Alert alert definition presents a menu containing action to duplicate the alert.
Because the Alert name must be unique, we entered ITSO Annotation Alert. Because this is an alert definition for Annotations, we kept the default value, Annotation Alert, for the Category name. See Figure 7-37.

Clicking Next allows you to configure the Type of alert. We accepted the default values. The alert definition for Annotations must be set to Custom as shown in Figure 7-38 on page 202.
On the next window, we accepted the default values, as shown in Figure 7-39. The class designated in the Evaluator Class input is specific for annotation alerts and must not be changed.

The Data Selection window allows you to select any data you want to add to the alert definition. It is not necessary to select data for Annotation Alerts; therefore, we accepted the defaults and clicked Next.

The Parameters window allows you to add any parameters that you want to allow the user to configure in the My Alerts portlet. We did not include any customizable parameters in our application.

The Features window allows you to select alerts features. As shown in Figure 7-40 on page 203, we selected only Automated notification (E-mail, SMS, etc).
The Automated Notification window allows you to select when the alerting system checks for new alerts to send. As shown in Figure 7-41, we selected **At the start of every hour**.

After reviewing our configuration on the Review window, we clicked **Finish** to save our duplicated alert definition for Annotations. We then verified that our duplicated alert can be seen in the Manage Alerts portlet, as shown in Figure 7-42.
7.6 Archiving annotations

The Annotations feature set includes an administrative portlet that you can use to remove annotations from the database and archive them to a file.

As shown in Figure 7-43, the Annotation Administration portlet allows you to specify a range of annotations to archive.

![Annotation Administration](image)

*Figure 7-43  The Annotation Administration portlet*

The archived files are stored in the `/WEB-INF/solutions/annotations/db_persistence/archives` directory, which is relative to the deployed Web archive (WAR) directory on WebSphere Portal.
Additional material

This IBM Redpaper refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this paper is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

ftp://www.redbooks.ibm.com/redbooks/REDP4313

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select Additional materials and open the directory that corresponds with the IBM Redpaper form number, REDP4313.

Installing the example application

This section provides an explanation of the steps necessary to install the example application. We assume that you already have WebSphere Dashboard Framework installed; if you do not, refer to 2.1, “Installing WebSphere Dashboard Framework” on page 16.

The example application requires WebSphere Dashboard Factory 6.0 (or later). We expect these dashboards to function correctly in a variety of server and database configurations compatible with WebSphere Dashboard Factory 6.0 (or later). We assume that the following products are installed:

- WebSphere Application Server (WAS) V6
- WebSphere Portal Server (Portal) V6
- DB2 V8
- Lotus Domino 8
The overall installation steps are:
1. Download the distribution zip file.
2. Create database and define data source.
3. Deploy Lotus Notes database to Domino.
4. Create projects in the WebSphere Dashboard Factory designer (automatic deployment must be enabled).
5. Import dashboard to import the WebSphere Portlet Factory archive.
6. Configure project for Domino.
7. Redeploy project to redeploy your WebSphere Portlet Factory project.
8. Populate the database.

Downloading

Download the IBM Redpaper ZIP file:
1. Create a temporary directory, for example, c:\temp\example_dashboards.
2. Download the IBM Redpaper dashboard zip file to the temporary directory.
3. Unzip the downloaded file.

Note: After unzipping, the temporary directory contains a WebSphere Portlet Factory archive ZIP and pkg file. Do not unzip these files; they are used in another installation step. After they are unzipped, you have the following files:
- WDF.zip: This is the main WebSphere Dashboard Framework project.
- AlertsScheduler.zip: This is the alert scheduler WebSphere Dashboard Framework project.
- WDFDB2.zip: This is the DB2 database associated with the example application.
- WDFDomino.zip: This is the domino database used with the example application.
- setupPages.xml: This is the XML access script for setting up the users, pages, and portlets on the portal server.

Create database and define data source

Use your favorite technique to create the database called WDF_PROD. For example:
1. Start → IBM DB2 → Command line tools → Command window
2. db2 create database WDF_PROD.

Next, configure a data source for this project. Refer to 3.1.2, “Configuring the JDBC provider and data source” on page 54 for instructions about how to do this with the following notes:
1. Ensure that you set the JNDI name to jdbc/v5/dashboard.
2. Ensure that you use WDF_PROD for the database name.
3. Ensure that you specify your DB2 server address in the Server Name field.

Deploy the Lotus Notes database

Copy the SA_Orders.nsf to the root level of the data directory on your Lotus Domino server and ensure that the database can be opened using a Notes Client.

Note: The Domino Data Access builder used in `<yourProject>/models/data/CustomerOrdersProvider.model` is configured to look for the database at the root level of the Domino data directory. If you locate the database elsewhere in the data directory, you need to update the Database Name input in the Domino Data Access builder.

You must create a full-text index on the database. If not, the example application is unable to search the database.

Important: You must create a full-text index on the database. Otherwise, the Customer Orders portlet in the Example Application will not display data from SA_Orders.nsf.

Use the following steps to create a full-text index:

1. In Lotus Notes 8, open the SA_Orders.nsf database.
2. Select File → Application → Properties.
3. In the Database properties window, select the Full-Text tab, denoted by a magnifying glass icon. See Figure A-1 on page 208.
4. On the Full-Text tab, click Create Index. See Figure A-1 on page 208.
5. In the Create Full-Text Index window, accept the default values and click OK. See Figure A-2 on page 208.

Important: WebSphere Dashboard Framework product ships with a sample application and sample data. The example application in this IBM Redpaper uses the same sample data that ships with WebSphere Dashboard Framework and also uses the same JNDI name, jdbc/v5/dashboard.

If you have already installed the sample application that ships with WebSphere Dashboard Framework, and it is configured to use DB2 with this JNDI name, you can simply reuse this data source. However, if the product sample application uses Cloudscape and the JNDI name is jdbc/v5/dashboard, temporarily remove the Cloudscape data source with this JNDI name. Otherwise, you cannot create the DB2 data source with this JNDI name for this IBM Redpaper example application.
Creating the projects in WebSphere Dashboard Framework

You need two projects in WebSphere Dashboard Framework:

- The shared project contains all the models used throughout this paper. Name this project WDF when you create it.
- The scheduler project is used for alerting schedules. Name this project AlertsScheduler when you create it.

For instructions about creating a project, see 2.2.2, "WebSphere Portlet Factory concepts" on page 21.

Ensure that you add the features as shown in Figure A-3 on page 209.
Figure A-3  Adding features to the project

For instructions to create the scheduler project, see 6.2.6, “Step six: Creating an Alerts Scheduler project” on page 154. Ensure that you select the WebSphere Native application program interface (API) when deploying this project.

**Important:** The alert scheduler must use the WebSphere Native API and not the Java Specification Request (JSR) 168 API.

**Import WebSphere Portlet Factory archive**

Repeat this step for both the AlertsScheduler project and the WDF project.

Highlight the project name, right-click and select **Import → WebSphere Portlet Factory Archive** as shown in Figure A-4 on page 210.
Next, browse to where you unzipped the downloaded material and import WDF.zip for the WDF project and the AlertsScheduler.zip for the AlertsScheduler project. As it imports, select Yes to all overwrite dialog boxes.

**Configure project for Domino**

You need to tell WebSphere Dashboard Framework where your Domino server is located. In WebSphere Dashboard Framework, navigate to WebContent → WEB-INF → config → domino_config and edit itso_domino_server.properties to the values that are appropriate for your Domino server.

If you have not already done so, create a full-text index on the SA_Orders.nsf database, as described in “Deploy the Lotus Notes database” on page 207.

**Redeploy project**

For the Alert Scheduler project, right-click on the project and select Build Portlet Factory WAR → Build Portlet WAR.

Repeat the same step for the WDF project as shown in Figure A-5 on page 211.
Populate the database

To populate the database, follow these steps:

1. In the downloaded material, there is a file called WDFDB2.zip. Unzip this file to any location, for example, C:\Import\.
2. Next, open a DB2 Command window. In Microsoft Windows, click Start → IBM DB2 → DB2COPY1 → Command Line Tools → Command Window and change to the directory where you just unzipped the zip file.
3. Run the following command to import the database structure: `db2 -tvf wdf_prod.sql`.
4. Then, run `db2move WDF_PROD load`.

Configure e-mail alerts

If you want to configure the e-mail notifications to work in your environment, you need to change the e-mail information that is located in:

`WDF\WebContent\WEB-INF\solutions\alerting\xml_persistence\notifier_defses\email.xml`
Change the values for the mail server and e-mail from address. Ensure that the e-mail server that you specify is able to accept e-mail from external sources.

After you deploy the project and configure the pages, you need to import your e-mail definition into the DB2 database. To do this, go to the admin page under the ITSO Dashboard tab in the portal and locate the Import Definitions portlet. On this portlet, click the second tab for Notifications and select the Import button next to e-mail.

Set up pages and portlet

Use the XML access script files that are in the download package to create dashboard pages in the portal and to place dashboard portlets onto those pages:

1. Copy the setupPages.xml to the bin directory of your portal server, C:\IBM\WebSphere\PortalServer\bin
2. Open a command line dialog and navigate to the bin server of the application server, C:\IBM\WebSphere\AppServer\bin, and run setupcmdline.bat.

In the same command window, change directories to C:\IBM\WebSphere\PortalServer\bin and run:

xmlaccess.bat -user wpsadmin -password wpsadmin -url <yourserver>:10038/wps/config
-in setupPages.xml -out result.xml

**Important:** The XML access script assumes that you are using the default user repository. All of the users use the organizational unit of the default organization. So the admin user is cn=wpsadmin,o=default organization. If you have security enabled in your environment, you need to edit the XML access script to reflect your LDAP setting. To do this, open the script in a text editor and search for default organization and make the appropriate changes.

After the script has completed, check the results.xml file, which is in the <installDir>/portalserver/bin directory to ensure that there are no errors in it.

After you check the file, log in to the portal server to see an ITSO Dashboard tab in the home section of the portal with four pages underneath that tab.

System requirements for downloading the Web material

We recommend the following system configuration:

- **WebSphere Dashboard Framework:** 6.0 or above
- **Portal server:** WebSphere Portal Server 6.01 or higher
- **Database:** IBM DB2 V9.1 or higher
- **Notes and Domino:** 6.0 or above

How to use the example application

The example application is made up of four pages within WebSphere Portal.

At the top level, there is an ITSO Dashboard tab. Under that tab, there are four pages: Dashboard, KPIs, Actuals, and Admin.
There are also four users to explore. All of the users have their passwords set to password:

- **SalesManager**: This is the overall manager that has the authority to see all of the sales data.
- **America**: This user is responsible for sales in the American region.
- **Europe**: This user is responsible for sales in the European region.
- **Asia**: This user is responsible for sales in the Asian region.

Log on as each user, explore the application, and notice the differences for each user. Explore the personalization options for the various portlets and how these options change the behavior of those portlets.
Building Composite, Role-Based Dashboards with WebSphere Dashboard Framework

Introduction to WebSphere Dashboard Framework

WebSphere Dashboard Framework gives application developers the power to create rich graphical interfaces to business data, which allow the user to make key decisions based upon that data. WebSphere Dashboard Framework also provides powerful collaboration features as well as real-time alerts to provide an organization with the tools needed to make timely decisions.

In-depth look at WebSphere Dashboard Framework builders

This IBM Redpaper provides you with the knowledge to get started with WebSphere Dashboard Framework, as well as take an in-depth look into its key features. Whether you are a line of business user wondering what it is all about or an advanced application developer, this IBM Redpaper will give you a greater understanding of how WebSphere Dashboard Framework can be utilized within your organization.

Leverage key features, such as Annotations and Alerts

Through the use of an application, this IBM Redpaper will solve the dashboarding requirements of a fictitious organization and help you use the same concepts in your own organization.

For more information: ibm.com/redbooks