Unified Reporter Server
ULTIMATE GUIDE
Release 1.9

http://www.nimsoft.com/
# Table of Contents

## Chapter 1 Introduction ......................................................... 9
  1.1 Welcome to JasperReports Server ........................................ 9
  1.2 Community and Commercial Editions .................................... 9
  1.3 About this Guide ........................................................... 10
      1.3.1 Technical Business Analyst ....................................... 10
      1.3.2 Report Developer .................................................. 10
      1.3.3 System Developer .................................................. 10
      1.3.4 System Administrator and Database Administrator .............. 10
  1.4 Other Resources ............................................................ 11
      1.4.1 Standard Documentation .......................................... 11
      1.4.2 Premium Documentation .......................................... 11
      1.4.3 JasperForge ........................................................ 11
  1.5 Getting Started ............................................................ 11

## Chapter 2 Dashboards ............................................................ 13
  2.1 User Interface Components ............................................... 14
  2.2 Context Menus .............................................................. 15
  2.3 Tips and Tricks ............................................................. 16

## Chapter 3 Ad Hoc Reports and Data Exploration ............................ 19
  3.1 About the Ad Hoc Editor .................................................. 19
      3.1.1 User Interface Components ....................................... 20
      3.1.2 Context Menus .................................................... 25
  3.2 Working with Ad Hoc Reports ............................................ 25
      3.2.1 Getting Started with the Editor ................................ 26
      3.2.2 Report-level Formatting Options ................................ 27
      3.2.3 Ad Hoc Tables ..................................................... 28
      3.2.4 Ad Hoc Charts ..................................................... 30
      3.2.5 Ad Hoc Crosstabs ................................................ 32
      3.2.6 Summaries ......................................................... 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.7</td>
<td>Setting the Data Format</td>
<td>35</td>
</tr>
<tr>
<td>3.2.8</td>
<td>Custom Fields</td>
<td>35</td>
</tr>
<tr>
<td>3.2.9</td>
<td>Running and Saving Reports</td>
<td>37</td>
</tr>
<tr>
<td>3.2.10</td>
<td>Tips and Tricks</td>
<td>37</td>
</tr>
<tr>
<td>3.3</td>
<td>Administering Ad Hoc Reports</td>
<td>38</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Administering Topics</td>
<td>38</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Administering Domains</td>
<td>38</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Administering Styles</td>
<td>38</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Ad Hoc Editor Configuration</td>
<td>41</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Scalability</td>
<td>42</td>
</tr>
<tr>
<td><strong>Chapter 4</strong></td>
<td><strong>Custom Data Sources</strong></td>
<td><strong>43</strong></td>
</tr>
<tr>
<td>4.1</td>
<td>Data Sources in JasperReports Library</td>
<td>43</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Query Executers</td>
<td>44</td>
</tr>
<tr>
<td>4.2</td>
<td>Overview of the Custom Data Source Examples</td>
<td>44</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Custom Bean Data Source</td>
<td>44</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Webscraper Custom Data Source</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>Prerequisites and Installation of the Examples</td>
<td>45</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Java Development Kit</td>
<td>45</td>
</tr>
<tr>
<td>4.3.2</td>
<td>About Apache Ant</td>
<td>45</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Installation</td>
<td>45</td>
</tr>
<tr>
<td>4.4</td>
<td>Creating a Custom Data Source</td>
<td>46</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Implementing the ReportDataSourceService Interface</td>
<td>46</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Defining Custom Data Source Properties</td>
<td>46</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Implementing Optional Interfaces</td>
<td>47</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Creating the Message Catalog</td>
<td>47</td>
</tr>
<tr>
<td>4.4.5</td>
<td>Defining the Custom Data Source in Spring</td>
<td>48</td>
</tr>
<tr>
<td>4.4.6</td>
<td>Configuring the Message Catalog</td>
<td>49</td>
</tr>
<tr>
<td>4.5</td>
<td>Installing a Custom Data Source</td>
<td>49</td>
</tr>
<tr>
<td><strong>Chapter 5</strong></td>
<td><strong>Query-Based Input Controls</strong></td>
<td><strong>51</strong></td>
</tr>
<tr>
<td>5.1</td>
<td>Creating a Query-based Input Control</td>
<td>52</td>
</tr>
<tr>
<td>5.2</td>
<td>Built-in Parameters for Query-based Input Controls</td>
<td>55</td>
</tr>
<tr>
<td>5.3</td>
<td>Domain-based Queries</td>
<td>56</td>
</tr>
<tr>
<td>5.4</td>
<td>Cascading Input Controls</td>
<td>57</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Parameters in Input Control Queries</td>
<td>58</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Step-by-Step Example</td>
<td>58</td>
</tr>
<tr>
<td><strong>Chapter 6</strong></td>
<td><strong>Access Control</strong></td>
<td><strong>61</strong></td>
</tr>
<tr>
<td>6.1</td>
<td>Authentication</td>
<td>61</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Spring Security Filter Chain</td>
<td>61</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Authentication Processing Filter</td>
<td>62</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Further Documentation</td>
<td>63</td>
</tr>
<tr>
<td>6.2</td>
<td>Authorization</td>
<td>63</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Repository Design for Access Control</td>
<td>64</td>
</tr>
</tbody>
</table>
Chapter 7  Securing Data in a Domain .................................................. 77
  7.1  Business Case ................................................................. 78
  7.2  Implementation .............................................................. 78
       7.2.1  Overview of CZS’s Process .............................................. 78
       7.2.2  Sales Numbers Domain ................................................. 79
       7.2.3  Roles ................................................................. 80
       7.2.4  Users ................................................................. 80
       7.2.5  Security Files and Profile Attributes ............................. 81
       7.2.6  Preparation .......................................................... 86
       7.2.7  Creating a Security File .............................................. 87
       7.2.8  Testing and Results .................................................. 90
  7.3  Domain and Security Recommendations .................................. 91
  7.4  Reference Material ......................................................... 92
       7.4.1  Domain Design in XML Format ..................................... 92
       7.4.2  Domain Security File ................................................ 96
       7.4.3  Alternative Filter Expressions using the testProfileAttribute Function .................................................. 98

Chapter 8  Application Security ....................................................... 101
  8.1  Using SSL in the Web Server .............................................. 101
       8.1.1  Setting Up an SSL Certificate ...................................... 102
       8.1.2  Enabling SSL in the Web Server .................................... 102
       8.1.3  Configuring JasperReports Server to Use Only SSL .......... 103
  8.2  Disabling Unused HTTP Verbs .......................................... 103
  8.3  Setting the Secure Flag on Cookies ..................................... 104
  8.4  Setting httpOnly for Cookies .......................................... 104
  8.5  Using a Protection Domain Infrastructure ............................. 104
       8.5.1  Enabling the JVM Security Manager ............................. 105
       8.5.2  Restoring Disallowed Permissions ................................ 105
       8.5.3  Additional Customizations for Previous Versions of Tomcat .... 106
  8.6  Final Note ................................................................. 107

Chapter 9  JasperReports Server Plug-in for iReport .......................... 109
  9.1  Getting Started ............................................................ 109
       9.1.1  Installation and Startup ............................................ 109
       9.1.2  Server Connection Configuration .................................. 110
       9.1.3  Common Connection Errors ....................................... 111
## Chapter 10 JasperReports Server APIs

10.1 Java APIs
   - The Public JasperReports Server API
   - Repository API
   - Report Data Source Service API
   - Report Scheduling API
   - Users and Roles API
   - Object Permissions API
   - OLAP Connection API
   - Flushing the Analysis Cache Using the API

10.2 Ad Hoc Launcher Java API
   - Communicating with the Ad Hoc Editor using AdhocData
   - Integration with JasperReports Server
   - A Sample Ad Hoc Launcher

10.3 Repository HTTP API
   - Executing ReportUnits
   - Linking to Content
   - Viewing Resources in the Repository

10.4 Ad Hoc Editor HTTP API
   - Examples URIs
   - Action Codes

10.5 Dashboard Designer HTTP API
   - Examples URIs
   - Action Codes for the Dashboard Designer

## Chapter 11 Customizing the User Interface

11.1 Changing the UI With Themes
   - Working with Themes
   - Changing the Logo
   - Hiding UI Elements
   - Changing Colors and Fonts
   - Changing the Spacing in the UI
   - Replacing the Default Theme
11.2 Working With Web App Files ......................................................... 159
  11.2.1 File Locations ................................................................. 160
  11.2.2 JavaScript and JSP Folders ............................................... 160
  11.2.3 Unzipping WAR Files ..................................................... 161
  11.2.4 Upgrading With UI Customizations ................................. 161
11.3 Customizing the Branding with SiteMesh .................................. 161
  11.3.1 web.xml ................................................................. 162
  11.3.2 sitemesh.xml and decorators.xml .................................. 162
  11.3.3 main.jsp and decorator.jsp ......................................... 163
  11.3.4 Editing decorators.jsp for Rebranding ............................ 164
11.4 Customizing JSP and JavaScript for the Login Page ..................... 165
  11.4.1 Reloading the JasperReports Server Web App ....................... 167
11.5 Customizing the Report Rendering Page ................................... 168
  11.5.1 Customizing the Input Controls Form .............................. 168
  11.5.2 Customizing the Report Viewer ...................................... 170
11.6 Customizing Menus .............................................................. 171
  11.6.1 Removing a Menu Item .................................................. 172
  11.6.2 Restricting a Menu Item by Role .................................... 173
  11.6.3 Adding an Item to the Main Menu ................................... 175
  11.6.4 Adding a New Main Menu ............................................. 178
  11.6.5 Changing Other Menus ................................................ 179
  11.6.6 Action Model Reference ............................................... 179
11.7 Working With Source Code Files ............................................. 184
11.8 Adding a Custom JSP Page in a Spring Web Flow ......................... 185
11.9 Adding Custom Export Channels ............................................ 188
  11.9.1 About Export Parameters ............................................. 189
  11.9.2 Adding the Exporter to the Report Viewer ........................ 189
  11.9.3 Adding the Export Format to the Scheduler ....................... 191
  11.9.4 Adding the Export Channel to Web Services ...................... 192

Chapter 12 Designing a Cluster ..................................................... 195
  12.1 Sample Cluster Architecture .............................................. 195
  12.1.1 JasperReports Server Clients ......................................... 197
  12.1.2 Load Balancer .......................................................... 197
  12.1.3 JasperReports Server Instances ..................................... 197
  12.1.4 Shared Repository Database ......................................... 198
  12.1.5 Job Schedulers .......................................................... 199
  12.1.6 Other Shared Resources .............................................. 199
  12.2 Jaspersoft OLAP in a Cluster .............................................. 200
  12.3 Session Management and Failover ........................................ 200
    12.3.1 Impact on Browser Users .......................................... 201
    12.3.2 Impact on Web Services Clients ................................. 201
  12.4 Cluster Design Process ..................................................... 201
  12.5 Performance Requirements ............................................... 202
  12.6 Availability Requirements ............................................... 203
CHAPTER 1  INTRODUCTION

JasperReports Server, the Business Intelligence (BI) platform from Jaspersoft, helps your users make well-informed decisions by providing advanced reporting and analysis tools.

This chapter contains the following sections:

- Welcome to JasperReports Server
- Community and Commercial Editions
- About this Guide
- Other Resources
- Getting Started

1.1  Welcome to JasperReports Server

JasperReports Server builds on JasperReports, the world’s most popular open source Java reporting library. It provides a comprehensive family of Business Intelligence (BI) products, including robust static and interactive reporting, dashboards, report server, data analysis, and data integration capabilities. You can use JasperReports Server as a stand-alone product or as part of an integrated end-to-end BI suite that utilizes common metadata and provides shared services, such as a repository, security, and scheduling. JasperReports Server exposes comprehensive public APIs for seamless embedding into other applications, as well as the capability to add custom functionality easily.

You may want to download JasperReports Server 4.2 Enterprise Edition for evaluation or contact Jaspersoft for upgrading to the commercial edition.

1.2  Community and Commercial Editions

JasperReports Server is a component of both a community project and commercial offerings. Each integrates the standard features such as security, scheduling, a web services interface, and much more for running and sharing reports. Commercial editions provide additional features, including Ad Hoc charts, flash charts, dashboards, Domains, auditing, and a multi-organization architecture for hosting large BI deployments.

Both community and commercial editions use the same Spring framework for easy integration into your applications, as well as an interface based on CSS for easy customization.

This guide discusses all editions. Sections of the guide that apply only to the commercial editions are indicated with a special note.
1.3  About this Guide

Because this Ultimate Guide is a comprehensive resource for users with many different needs, it includes information that may not be relevant to you. The following user descriptions and document maps can help you find the information that pertains to you.

1.3.1  Technical Business Analyst

Technical business analysts know their business, data, and processes. They are power users who generate business intelligence for others.

If you are a technical business analyst, refer to the following sections of this document:
- Chapter 2, “Dashboards,” on page 13
- Chapter 6, “Access Control,” on page 61
- Chapter 7, “Securing Data in a Domain,” on page 77
- Section 10.2, “Ad Hoc Launcher Java API,” on page 134
- Section 11.1, “Changing the UI With Themes,” on page 154
- Section 11.6, “Customizing Menus,” on page 171

1.3.2  Report Developer

Report developers understand their business and its data and create reports for other users.

If you are a report developer, refer to the following sections:
- Chapter 3, “Ad Hoc Reports and Data Exploration,” on page 19
- Chapter 4, “Custom Data Sources,” on page 43
- Chapter 5, “Query-Based Input Controls,” on page 51

1.3.3  System Developer

System developers leverage JasperReports Server functionality in their own product. They extend and change the source code, system configurations, and other low-level options.

If you are a system developer, refer to the following sections:
- Chapter 8, “Application Security,” on page 101
- Chapter 10, “JasperReports Server APIs,” on page 123
- Chapter 11, “Customizing the User Interface,” on page 153

1.3.4  System Administrator and Database Administrator

System administrators install, deploy, maintain, and troubleshoot JasperReports Server, along with other systems in their environment. They also administrate security, including the creation and maintenance of users, roles, organizations, the assignment of permissions, and configure authorization and authorization in general. Database administrators (DBAs) administer database management systems (DBMS), and are familiar with both relational and Online Analytical Processing databases. They plan, configure, tune, and maintain the schemas that store business data.

If you are a system or database administrator, refer to the following sections:
- Chapter 3, “Ad Hoc Reports and Data Exploration,” on page 19
- Chapter 4, “Custom Data Sources,” on page 43
- Chapter 5, “Query-Based Input Controls,” on page 51
- Chapter 6, “Access Control,” on page 61
- Chapter 7, “Securing Data in a Domain,” on page 77
- Chapter 8, “Application Security,” on page 101
1.4 Other Resources

The following sections list other sources of documentation and information to help you work with JasperReports Server.

1.4.1 Standard Documentation

JasperReports Server includes standard documentation that is often referenced in this guide:

- JasperReports Server Installation Guide
- JasperReports Server User Guide
- Jaspersoft OLAP User Guide
- JasperReports Server Administrator Guide (commercial editions only)
- JasperReports Server Web Services Guide
- JasperReports Server Source Build Guide

Jaspersoft maintains two sets of documentation for JasperReports Server: one for the community project and one for the commercial editions. The respective versions are found in the `<js-install>/docs` directory when you install JasperReports Server, except for the JasperReports Server Source Build Guide, which is found in the source code distribution package. The community project documentation is also available at JasperForge.org.

1.4.2 Premium Documentation

This guide also references other Jaspersoft premium guides:

- Jaspersoft OLAP Ultimate Guide
- JasperReports Server External Authentication Cookbook
- JasperReports Ultimate Guide
- iReport Ultimate Guide

Users of the JasperReports Server commercial editions can access these documents from the Support Portal. For JasperReports Server community project users, these documents are available for purchase, separately or in documentation packs, at http://www.jaspersoft.com/ultimate-guides.

1.4.3 JasperForge

JasperForge.org is the open source development portal for the Jaspersoft Business Intelligence community. Here you’ll find developers and experts contributing to projects and answering questions. Registration is free and open to anyone.

1.5 Getting Started

JasperReports Server must be installed and configured before you can use it. For information, refer to the installation guide for your product edition.

The directory where JasperReports Server is installed is referred to as `<js-install>` in this guide. The default installation directory is:

- Windows: C:\Program Files\jasperreports-server-4.0
- Linux: <USER_HOME>/jasperreports-server-4.0

To connect to JasperReports Server, make sure your database and application server are running, then enter the corresponding URL in a supported browser:


Where:

- `<hostname>` is the name of the computer hosting the application server where JasperReports Server is installed.
<port> is the number of the port specified when the application server was installed. For example, if you installed the Jaspersoft BI Suite evaluation software, the default URL is:

http://localhost:8080/jasperserver-pro/login.html

If JasperReports Server is secured using SSL (Secure Socket Layer) encryption, both the protocol and the port differ. For example, a typical SSL-secured URL for JasperReports Server Professional follows this format:

https://localhost:443/jasperserver-pro/login.html

On the JasperReports Server Login page, enter a user ID and password and click **Login**. The following table lists the credentials for the evaluation server:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Password</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>superuser</td>
<td>superuser</td>
<td>System-wide administrator (commercial editions only)</td>
</tr>
<tr>
<td>jasperadmin</td>
<td>jasperadmin</td>
<td>Administrator for the default organization (commercial editions)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>System-wide administrator (community project)</td>
</tr>
<tr>
<td>joeuser</td>
<td>joeuser</td>
<td>Sample end-user</td>
</tr>
<tr>
<td>demo</td>
<td>demo</td>
<td>Sample end-user for the SuperMart Dashboard demonstration</td>
</tr>
</tbody>
</table>

Depending on the configuration of your system, the Login page may also enable you to change your password. If there is a **Change password** link, click the link to enter a new password. If there is no link, only your system administrator can change the password.
CHAPTER 2  DASHBOARDS

A dashboard displays several reports in a single, integrated view. A dashboard can also include input controls that determine the data displayed in one or more reports, other dashboards, and any other web content. By combining different types of related content, you can create appealing, data-rich dashboards that quickly convey business trends.

This chapter provides details about the dashboard designer and includes the following sections:

- User Interface Components
- Context Menus
- Tips and Tricks
2.1 User Interface Components

The figure below illustrates the main components of the dashboard designer:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available Content</td>
<td>The list of content you can add to the dashboard, including Special Content and repository content. Standard content includes buttons and input controls for the reports you add. Repository content is limited to reports and dashboards; you cannot include analysis views or report resources such as images. You can double-click items in the list to place it automatically in the dashboard, or drag and drop items to place them yourself. To hide the column of available content, click the icon in the top left corner of the column; this is helpful when arranging content in a large dashboard. Click the same icon on the minimized column to expand it again.</td>
</tr>
<tr>
<td>Canvas</td>
<td>Occupying the right side of the designer, the canvas area is a visual editor for your dashboard content. It displays an example of how the dashboard will look, but the contents are not interactive. To interact with the dashboard as it will appear to users, click Preview. Before frames and other content have been added, the dashboard area only displays the title area and the grid, which is helpful in aligning content. Optionally, you can set guide lines to show you the edge of common screen sizes.</td>
</tr>
<tr>
<td>Tool bar</td>
<td>These buttons let you interact with the dashboard as a whole or change your view of the designer. The Save button gives you the choice of saving with the current name (overwriting) or saving a copy as a different name. The Options button lets you control the size of the optional guide lines on the canvas.</td>
</tr>
<tr>
<td>Title bar</td>
<td>The area of the dashboard where the title is generally displayed. If you do not use this title field, it displays as blank in a dashboard and you can place content over it.</td>
</tr>
</tbody>
</table>
Context Menus

Context menus appear when you right-click elements in the dashboard designer. Each menu offers additional options for the selected element.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection frame</td>
<td>Shows the active element in the dashboard for moving or resizing. Drag anywhere inside the selection frame to move the element. Drag the blue square in the lower left corner to resize it. Click or right-click in any other element to change the selection.</td>
</tr>
<tr>
<td>Input control and label</td>
<td>If your dashboard contains reports that include input controls, they appear in the Special Content folder of the Available Content list. Once you add an input control to the canvas, you can edit its associated label and change the default value. The value of these input controls determine the content of any frames that refer to them.</td>
</tr>
<tr>
<td>Free text</td>
<td>Similar to a label, Free Text fields allow you to add text directly to the dashboard. Such text might include instructions about input controls or a description of the dashboard’s purpose. Unlike labels, the font size of Free Text changes when you change the size of its frame.</td>
</tr>
</tbody>
</table>
| Standard controls                | JasperReports Server provides these standard dashboard controls:  
  - **Submit.** When this button is included in a dashboard, changes to input control values only update the frame when you click **Submit.** For more information, refer to the [JasperReports Server User Guide](#).  
  - **Reset.** Reverts the value of all input controls to their default value saved in the dashboard.  
  - **Print.** Opens the browser’s Print Preview window and strips off the server’s page headers and footers. In the Print Preview, the Print button is hidden. |

For labels and free text, you can also click the text to edit it when the blue selection frame is not active. For reports and custom URLs you can set options such as scroll bars and the refresh interval, as shown in Figure 2-2.
### 2.3 Tips and Tricks

Here are some tips on designing dashboards:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-select options</td>
<td>When you select multiple frames, you can move or resize them all at once. To select several frames at once, use click-and-drag to draw a box around them or Ctrl-click while you select the frames.</td>
</tr>
<tr>
<td>Passing hidden parameters</td>
<td>If a report on the dashboard has a parameter that isn’t mapped to an input control in the dashboard, you can set a value for that parameter by adding it to the URL. To do so, append &amp;hidden_&lt;parameter&gt;=&lt;value&gt;, where &lt;parameter&gt; is the name of a parameter defined in the report and &lt;value&gt; is a valid value for the parameter. For example, to set a dashboard report’s Country parameter to USA, the URL might be similar to: http://&lt;hostname&gt;:8080/jasperserver-pro/flow.html?_flowId=dashboardRuntimeFlow&amp;dashboardResource=%2Fdashboards%2FMyDash&amp;hidden_Country=USA. The hidden parameter applies to all reports that reference this input control. This can be useful when emailing a link to a dashboard. You can select the default value the recipient will see by editing the URL in the email.</td>
</tr>
<tr>
<td>Suppressing server UI decoration</td>
<td>By default, dashboards are displayed on a standard JasperReports Server page called the Dashboard Viewer. This page includes other elements of the user interface such as the search field and main menus. To suppress all of the server UI decoration around the dashboard, add the viewAsDashboardFrame parameter to the dashboard URL as follows: http://&lt;hostname&gt;:8080/jasperserver-pro/flow.html?_flowId=dashboardRuntimeFlow&amp;dashboardResource=%2Fdashboards%2FMyDash&amp;viewAsDashboardFrame=true. This can be useful when emailing a link to a dashboard. You can strip out the header and footer so the recipient’s eye is drawn to the most important information. This can also be useful when embedding the dashboard in another application.</td>
</tr>
<tr>
<td>Embedding a dashboard in another page</td>
<td>You can embed a dashboard in another HTML page outside of JasperReports Server by creating an iFrame and specifying the dashboard’s URL as the iFrame’s src attribute.</td>
</tr>
</tbody>
</table>
| Using keyboard shortcuts         | You can move or delete dashboard content using your keyboard:  
  - Arrow keys: move up, down, left, or right.  
  - Delete key: delete the selected content.  
  - Escape: close the dialog or cancel edit, depending on your selection.  
  - Ctrl: select multiple frames or buttons. When you are dragging or resizing content, Ctrl disables the snap-to-grid behavior. |
| HTML in text fields              | Free Text fields have limited support for HTML code. For example, to create a label in a different color, such as blue, you could define a span within the Free Text field. In this case, you could enter the following: <span style="color:blue;">Blue Text</span>. Note that this advanced feature may return unexpected results, depending on the nature of the HTML you enter and the browser that displays the dashboard. |
| Adding labels on input controls  | When you add an input control to the dashboard, its label is automatically added as well; you can also add labels manually by dragging the Text Label standard control from the Available Content list. Text labels are like free text elements, but they cannot be resized. |
Adding local content, such as images

You can add images and other types of web content to a dashboard. To create a Custom URL frame that displays an custom content, either:
  - Publish the content to a URL-addressable location and create a Custom URL frame that points to it; or
  - Add the file under the jasperserver-pro directory in your installation, by default <js-install>/apache-tomcat/webapps/jasperserver-pro/. For example, you could create the jasperserver-pro/content directory to hold a file named cow_logo.jpg. The URL to this content would be http://<hostname>:8080/jasperserver-pro/content/cow_logo.jpg.

Custom URLs in dashboards support any content that can be displayed in an iFrame. See the example in Figure 2-3 on page 17.

Sizing frames

To automatically fit the frame around a report or dashboard, right-click the item and select **Size to Content**. You can also hide and show scroll bars in frames with other options on the context menu.

---

**Figure 2-3  Dashboard Tips and Tricks**

- Free text field using HTML
- Image URL from the file system
- Context menu for hiding scroll bars
CHAPTER 3  AD HOC REPORTS AND DATA EXPLORATION

This section describes functionality that can be restricted by the software license for JasperReports Server. If you don’t see some of the options described in this section, your license may prohibit you from using them. To find out what you’re licensed to use, or to upgrade your license, contact Jaspersoft.

JasperReports Server’s Ad Hoc Editor is a browser-based, interactive tool for designing reports and exploring your data:

- As a report designer, the Ad Hoc Editor lets you easily create and edit reports on a day-to-day basis. To create a report, select a Topic, Domain, or OLAP client connection, each of which defines a query and data source, and a report type. Then select the fields in your data source that should appear in your report. The Ad Hoc Editor’s interactive display of your results lets you evaluate your design without having to run the entire report first. Finally, you can run the report, print it, save it, and export it in several file formats.

- As a data explorer, the Ad hoc Editor provides analysis options (such as slice and pivot) to help you recognize trends and outliers in your data. You can drill into specific details or analyze your data at a very high level. For example, you might create a crosstab that shows the kinds of products a customer purchases together. Though your intention in creating the crosstab was to rank the popularity of certain items, the crosstab might also reveal correlation between customers’ purchases. These correlations, which you weren’t aware of before, may give you insight into how you can improve your business. For example, you might run a promotion to encourage the correlation, or change your store layout to expose customers to more options. Understanding your data can help you make better decisions.

The interesting trends and anomalies revealed by data exploration can lead you to create a report highlighting your findings. Conversely, while creating a report for distribution, you may identify a trend that warrants further investigation. You can move seamlessly between the two activities—report creation and data exploration.

This chapter contains the following sections:

- About the Ad Hoc Editor
- Working with Ad Hoc Reports
- Administering Ad Hoc Reports

3.1  About the Ad Hoc Editor

The Ad Hoc Editor is an interactive tool that displays the data fields retrieved from your data sources. The effects of your changes are evident immediately, and you can adjust the display to highlight the most relevant and compelling aspects of your data. For example, add a new field by simply double-clicking it or dragging it from the list of available fields to the report area. When you add a field to a chart, it appears as a measure; when you add the same field to a table, it is displayed as a column.
Remove a field by dragging it out of the report area. In a table, reorder columns in the same way: simply drag a column to its new location. You can drag to reorder many elements in the report area, such as groups in tables, measures in charts, and column and row groups in crosstabs. Such changes can drastically affect your report’s readability and appeal.


### 3.1.1 User Interface Components

Figure 3-1 below illustrates the main components of the Ad Hoc Editor:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields panel</td>
<td>The list of available fields in the chosen Topic or Domain. If you are using a Domain, fields may appear in named sets. Use the arrow beside the set name to expand or collapse a set of fields. To hide the panel of available fields, click the icon in the top left corner; this is helpful when arranging content in a large report. Click the same icon on the minimized panel to expand it.</td>
</tr>
<tr>
<td>Change source/Select fields</td>
<td>This menu lets you select a different Topic or Domain for your report. All data and formatting are lost when you select a different Topic or Domain. When creating a report from a Domain, you can also select different fields to change the list that appears in this panel.</td>
</tr>
<tr>
<td>Canvas</td>
<td>The canvas area displays the fields you add to the report, along with controls for modifying the appearance and contents of the report.</td>
</tr>
<tr>
<td>Tool bar</td>
<td>The tool bar at the top of the canvas provides access to many of the Ad Hoc Editor’s functions, such as saving and running the report, undoing and redoing changes, and changing the report’s style. For more information, refer to the JasperReports Server User Guide.</td>
</tr>
<tr>
<td>Data selector</td>
<td>Click Sample Data or Full Data to switch between the amount of data displayed. Use sample data to design a report quicker, and use full data to see the same data used in the full report. When you preview or run the report, full data is displayed regardless of the data selector shown in the editor.</td>
</tr>
</tbody>
</table>
The following figure illustrates the canvas when working on table reports, especially the display of groups:

![Figure 3-2 Ad Hoc Editor’s Table Layout](image)

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report type selector</td>
<td>Click Chart, Table, or Crosstab to view your data in that type of report. Changes made in one type of report apply to the data displayed on another.</td>
</tr>
<tr>
<td>Title bar</td>
<td>The top portion of the report area; click to add or edit the title of the report. To remove the title, point to the Properties menu and select Toggle the Title Bar.</td>
</tr>
<tr>
<td>Group control</td>
<td>Drag fields onto this icon to make them groups in the table. This control does not appear for crosstabs, because crosstab grouping is defined in the column and row headers.</td>
</tr>
<tr>
<td>Report area</td>
<td>Occupying the middle of the editor, the report area displays an example of how the report will look. For a complete view of your report, click for design mode or to run the report.</td>
</tr>
<tr>
<td>Filters panel</td>
<td>This panel displays any filters defined for the report. You can set the filter values and see the resulting change in the report area. To hide the filters panel, click the icon in the top left corner of the panel. Click the same icon on the minimized panel to expand it again.</td>
</tr>
<tr>
<td>Filter controls</td>
<td>These menus let you change the display of filters, either collectively or individually. The menu on the filter panel lets you collapse all filter or remove all filters. The menu on each filter lets you view the filter operation, for example “is one of,” or delete the filter. To minimize the view of an individual filter, click the arrow beside its name.</td>
</tr>
</tbody>
</table>
The following figure illustrates the report area when creating a chart in the Ad Hoc Editor:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column labels</td>
<td>Displays the label for each column, above the table in a header row. By default, the column label is the one defined in the selected Topic or Domain. Highlight the column and right-click to see the context menu with options to change or remove the label. When you remove a column label, its database name appears in the editor, but the label does not appear in the final report.</td>
</tr>
<tr>
<td>Group headers</td>
<td>Displays the label of the group and the value of the current group. For example, if you grouped by Country, Argentina and Bolivia might be group values. The first occurrence of a group header can be selected to operate on that group, as shown in Figure 3-2. You can drag the group header out of the table to remove that group, or right-click to access group options such as changing the label or creating a filter on the same field. Groups and sub-groups can be nested, and their first occurrence headers are all found at the top of the report. You can reorder these first headers to change the group nesting order.</td>
</tr>
<tr>
<td>Group summary</td>
<td>Repeats the label and value of the current group or sub-group to indicate the end of the group. If the column is summarized, the summary shows a value. To select a different summary function, modify the column summary.</td>
</tr>
<tr>
<td>Column Summary</td>
<td>Displayed at the bottom of a summarized report, the column summary gives a value for all the rows in the column. To select a different summary function, highlight the cell of the total summary, right-click and select Change Function from the context menu.</td>
</tr>
<tr>
<td>Column of Values</td>
<td>A vertical region in the report area representing data from a single field.</td>
</tr>
</tbody>
</table>

![Figure 3-3 Ad Hoc Editor's Chart Layout](image)
The following figure illustrates the report area when displaying the Crosstab tab of the Ad Hoc Editor:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chart area</td>
<td>A chart’s appearance is determined by the type of chart, the fields selected as measures, and the field that groups the data. Drag a field into the chart area to add a series, called a measure in the Ad Hoc editor. Right click the chart area to change the chart type and display options. Most chart types (such as pie and bar charts) display summarized data based on a single group, while others (such as scatter and time series) display each individual data point.</td>
</tr>
<tr>
<td>Group control</td>
<td>For charts that show summarized data, you can drag a field to the group control to define the grouping along the horizontal axis. Only one level of grouping can be displayed; dragging a second field to the group control will replace the first.</td>
</tr>
<tr>
<td>Scale</td>
<td>All measures are plotted against the same scale, which is sometimes confusing. You can use a calculated field to multiply or divide one measure so that its values are closer to those in the other measures in your chart. You can set the format of the scale by setting the data format of the first measure.</td>
</tr>
<tr>
<td>Legend</td>
<td>The legend is created automatically when fields are added to the chart as measures. You can drag field names in the legend to rearrange or remove measures. Right click a field name to change its label, summary function, or data format. Right click the chart for the option to remove the legend.</td>
</tr>
<tr>
<td>Group label</td>
<td>Name of the field used for grouping to create the series in the chart. The title can be hidden.</td>
</tr>
<tr>
<td>Group values</td>
<td>The group values on the horizontal scale indicates the values by which the chart is grouped. If there are too many values in the chart, use filters to reduce your data.</td>
</tr>
<tr>
<td>Resize handle</td>
<td>Appears when you hover inside the chart area. Drag the resize handle to enlarge or reduce the size of your chart.</td>
</tr>
</tbody>
</table>

You can add the same field to a chart in more than one capacity, for example, as a measure and a group. Because the nature of a chart is to display summarized data, there is no data selector. Charts always present the full data set, not a sample.

The following figure illustrates the report area when displaying the **Crosstab** tab of the Ad Hoc Editor:
The following figure illustrates the report area when displaying the **Crosstab** tab of the Ad Hoc Editor:

![Ad Hoc Editor's OLAP-based Crosstab Layout](image)

**Figure 3-5  Ad Hoc Editor’s OLAP-based Crosstab Layout**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row and column group labels</td>
<td>Displays the name of each field used for grouping. Use the group label to interact with the group: you can drag group labels to rearrange them, or right-click them to use the context menu. When no row groups are defined, a the words Row Group indicate this vertical region.</td>
</tr>
<tr>
<td>Row and column group values</td>
<td>Heading cells that show the group values. When there is more than one level of grouping, use the arrows on the outer groups to expand or collapse the inner groups. Right-click a group value to exclude it or keep only that value from among all group values of the same level.</td>
</tr>
<tr>
<td>Sorting controls</td>
<td>The icon beside each measure label shows the current sorting that applies to both rows and columns. Alphabetic sorting applies to the group values, and numeric sorting applies to the measure values. Click the icon to change the sorting. Because numeric sorting can only apply to one measure at a time, clicking one icon will reset the others to alphabetic sorting.</td>
</tr>
<tr>
<td>Measure labels</td>
<td>Displays the name of each measure in the crosstab. Use the measure label to interact with the measure: you can drag the measure labels to rearrange them, or right-click to change the summary function or data format of the measure.</td>
</tr>
<tr>
<td>Measures</td>
<td>Drag fields into this area to add them as measures. The measures show an aggregated value in each cell of the crosstab, as well as row and column totals for each level of grouping. Click on a measure value to open a table report in a new window showing the individual values that make up the aggregated value.</td>
</tr>
</tbody>
</table>

The following figure illustrates the report area when displaying the **Crosstab** tab of the Ad Hoc Editor:
### 3.1.2 Context Menus

Context menus appear when you right-click elements in the Ad Hoc Editor. Each menu offers options for the selected element.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available Measures</td>
<td>Displays all the measures defined in the current OLAP cube. Drag them to the <strong>Columns</strong> or <strong>Rows</strong> field to add them to the crosstab.</td>
</tr>
<tr>
<td>Measure Labels</td>
<td>Displays the label of each measure in the crosstab.</td>
</tr>
<tr>
<td>Dimensions</td>
<td>Displays the levels that have been added to the crosstab from each dimension. Note that dimensions can be added as both rows and columns.</td>
</tr>
<tr>
<td>Measures</td>
<td>Displays the measures that have been added to the crosstab. Note that measures can be added as either rows or columns.</td>
</tr>
<tr>
<td>Columns</td>
<td>The area of the crosstab that displays dimensions and measures as columns.</td>
</tr>
<tr>
<td>Rows</td>
<td>The area of the crosstab that displays dimensions and measures as rows.</td>
</tr>
</tbody>
</table>

**Figure 3-6** Sample Context Menus

*Figure 3-6* shows the following examples:

- Right-clicking a column in a table report opens a context menu with options for that column’s settings.
- Right-clicking the main area in a chart opens a context menu with options for chart-level settings.
- Right-clicking a group value in a standard crosstab’s row or column lets you slice the crosstab.

### 3.2 Working with Ad Hoc Reports

The following sections provide details on creating reports with the Ad Hoc Editor:

- Getting Started with the Editor
- Report-level Formatting Options
- Ad Hoc Tables
- Ad Hoc Charts
- Ad Hoc Crosstabs
- Summaries
- Custom Fields
- Running and Saving Reports
- Tips and Tricks
3.2.1 Getting Started with the Editor

This section describes the initial selections you make in the Ad Hoc Editor:

- Launching the Ad Hoc Editor
- Choosing a Topic, Domain, or OLAP Client Connection
- Choosing the Report Type

3.2.1.1 Launching the Ad Hoc Editor

To launch the Ad Hoc Editor:

- Click **Create > Ad Hoc Report**. The Source page appears.

![Figure 3-7 Source Dialog in the Ad Hoc Editor](image)

You can edit an existing Ad Hoc report if you have the correct permissions:

1. Enter a search term or click **View > Repository** or **View > Reports** to search or browse for your report.
2. Right-click the Ad Hoc report and select **Open in Designer**.

Only Ad Hoc reports for which you have write permission display this menu option.

3.2.1.2 Choosing a Topic, Domain, or OLAP Client Connection

Ad Hoc Editor displays data defined by a Topic, Domain or OLAP client connection:

- A Topic represents a group of data fields and a query from a related area. Examples might include Marketing Campaigns, Sales Figures, or Call Volume. A Topic is created either by uploading a JRXML file to the repository or by selecting a Domain when you launch the Ad Hoc Editor, and saving your selections as a Domain Topic.
- A Domain is a semantic layer defined by an administrator to expose certain tables and fields, as well as derived tables and calculated fields. Domains are created using the Domain Designer in JasperReports Server; they can also be created by uploading an XML file that follows a Jaspersoft-specific syntax. The fields in a Domain may not all be related; in this case, the Domain contains two or more data islands—groups of related data fields. When creating an Ad Hoc report based on a Domain, you choose the data fields (called items) that you want to make available to your users in the Ad Hoc Editor. In the editor, users can add or remove fields in the list of those available.
• An OLAP client connection is a definition for retrieving data from an OLAP data source. Such connections expose transactional data and define how the data can be seen as a multidimensional cube. Typically, business analysts interact with these cubes through OLAP views; the Ad Hoc Editor can also define reports based on cubes.

Typically, an administrator creates a set of Topics, Domains, and OLAP client connections that serve as good starting points for Ad Hoc reports. Examine the name and description of each to determine which best suits your needs. It is often helpful to create a sample report from a Topic, Domain, or OLAP client connection to understand its content.

• Click a Topic to select it, then click Table, Chart, or Crosstab. The Ad Hoc Editor appears with a list of the fields that you can add to the report area.

• Click the Domains tab, click a Domain, and click Choose Data. A wizard walks you through choosing the fields you want to include, filtering the data rows, renaming fields, and saving the new set of data as a Domain Topic.

• Click the OLAP Connections tab, click a connection, and click Crosstab. The Ad Hoc Editor appears with a list of the fields that you can add to the crosstab.

• If the server was installed with its sample data, the Source dialog includes examples that can help you familiarize yourself with the Ad Hoc Editor. For more information, refer to the JasperReports Server User Guide.

• For more information on Topics, see section 3.3, “Administering Ad Hoc Reports,” on page 38.

3.2.1.3 Choosing the Report Type

By default, the Ad Hoc Editor creates simple table reports that display data in rows and columns that can be grouped and sorted. In the Source dialog, choose Chart to display data in various kinds of graphical charts; choose Crosstab to combine several values into a concise set of rows and columns that display summarized data.

You can also change the type of report within the Ad Hoc Editor using the selector in the canvas area beneath the tool bar. You may change report types on the canvas at any time, even after you have placed fields and groups in your report. The Ad Hoc Editor displays the same information but adapts it to the report type selected. For example, if you display columns and groups in a table, those same fields are displayed as series and bars (respectively) in a chart. Be aware that the different report types are not perfectly equivalent:

• Charts can only display one row group and no column groups.
• Tables cannot display column groups.

If you switch to a report type that cannot display a group that you added to your report, it is left out. However, the Ad Hoc Editor remembers all the groups added to your report, and, as long as you do not modify the report, will display them when you switch back to another report type. Also, if you have defined column groups on a crosstab, you can pivot those groups so they are visible as row groups in a table or chart layout.

3.2.2 Report-level Formatting Options

These formatting selections apply to the entire report:

<table>
<thead>
<tr>
<th>Edit the report title</th>
<th>To add a title to the entire report, click Click To Add a Title and enter the title in the text field. To edit the title, click its current title and enter new text.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>To remove the title, point to Toggle the Title Bar.</td>
</tr>
</tbody>
</table>
3.2.3 Ad Hoc Tables

3.2.3.1 Columns

In tables, the Ad Hoc Editor enables you to add, remove, and format columns. You can drag-and-drop or choose options from the context menu that appears when you right-click a column.

For tables and crosstabs, the editor only displays a subset of your data to help you picture the final report. Click Full Data to view the full dataset; click Sample Data to return to the smaller subset.

Common tasks with columns include:

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a column</td>
<td>Drag it from the list of available fields and into the report area. While you drag, the field’s new position is highlighted with a vertical bar. You can also add a column by right-clicking in the list of available fields and selecting Add as Column from the menu that appears.</td>
</tr>
<tr>
<td>Remove a column</td>
<td>Drag it outside the report area. You can also remove columns using the context menu.</td>
</tr>
<tr>
<td>Change the order of columns</td>
<td>Drag columns to new locations. While you drag, a vertical line shows the column’s new location, and its border changes color.</td>
</tr>
<tr>
<td>Resize a column</td>
<td>Click a column’s right-hand border and drag it horizontally. Drag right to widen the column; drag left to narrow it. Your cursor changes as you hover it over a column border to indicate that you can drag the border. For more precision, click the column to highlight its borders before dragging them. The minimum width of a column is determined by its longest visible member (be it a row or the label itself).</td>
</tr>
<tr>
<td>Rename a column</td>
<td>Right-click the column and select Edit Label from the context menu. Enter the new name.</td>
</tr>
</tbody>
</table>
**Add a summary and select a function**

Right-click a column and select **Add Summary** from the context menu. JasperReports Server applies the default summary function for this column’s datatype. The summary function determines how the column’s summary is calculated, such as average or total. You can change the function once the summary is added; see the following table. For more information, see section 3.2.6, “Summaries,” on page 34.

**Add space**

To add white space between columns, drag the Spacer from the list of available fields and drop it in the report area. A new blank column is added. It acts as a resizable buffer between columns or between a column and the edge of the report. Drag the spacer’s edges to widen or narrow it. You can add any number of spacers to a report.

**Sorting**

Click \[ \text{\textsuperscript{ asc} asc} \] to open the sorting window. Add fields, change the direction of sort from ascending to descending. You can sort by multiple fields (including those not displayed in the report). If you use multiple fields, the server sorts the rows by the first field that appears in the sorting window. Within those rows, the data is sorted by the next field displayed in the sorting window.

**Filtering**

Right-click a field or a column and select **Create Filter** to filter your results by the values in that field. Depending on the datatype of the field, the server creates a suitable filter and selects an appropriate operator. For example, if you create a filter from a date field, the filter’s operator is set to \text{is between}; enter dates that define the time period that interests you. You can also select a different filter operation by pointing at the filter’s \[ \text{\textsuperscript{ filter}} filter \] icon and selecting Toggle Operation to display the operation field.

3.2.3.2 Grouping

Grouping the data in your reports can increase their impact and highlight anomalies that might otherwise go unnoticed. You can group data by multiple fields to create hierarchical levels. For example, when you group a report by both country and postal code, the report is organized as an alphabetical list of countries; within each country, the data is further broken down by postal code. Such grouping might reveal the locations where you are achieving the best results.

Common grouping tasks include:

<table>
<thead>
<tr>
<th>Create a group</th>
<th>Drag a field to the group control. You can also create a group by right-clicking a field in the list of available fields and selecting <strong>Add as Group</strong> from the context menu. When you add a group, the field automatically becomes the innermost group (that is, the last group).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change the order of groups</td>
<td>When creating groups, pay special attention to the order they appear in the Ad Hoc Editor. For example, it doesn’t make sense to group first by postal code then by country, because each postal code belongs to only one country. In this case, each country appears in the report multiple times: once for each postal code in that country. Drag the postal code group down and drop it under the country group in such a report. When working with groups, always remember to use the topmost group in the table. Change the order of groups by dragging them above or below other groups. While you drag it, a horizontal blue bar indicates the group’s new position, and its color changes when your cursor hovers over a valid location.</td>
</tr>
<tr>
<td>Groups and summaries</td>
<td>When you summarize a column, the server displays a summary value for each group, even when you have multiple groups. For example, sales data might be summarized for each postal code and also for each country. If you only want to see the summary rows, point to [ \text{\textsuperscript{ hide}} hide ] and select <strong>Hide Detail Rows</strong>. This option toggles between showing all the detail rows and just the group summaries.</td>
</tr>
</tbody>
</table>
When a table report is grouped by multiple fields, you may only see a few or even a single, partial group. Click **Full Data** to view the full dataset; click **Sample Data** to return to the smaller subset.

### 3.2.4 Ad Hoc Charts

You can create several types of charts. A chart displays one group, such as countries or stores, with any number of measures on the group. Some charts (such as bar and pie) display summarized data; others (such as time series and scatter) show each data point separately.

To filter a chart, right-click a field in the list of those available and select **Create Filter**.

You can't create a filter by clicking elements in the chart.

#### 3.2.4.1 Formatting Options for Charts

Charts can be of several types, such as bar or pie. You can edit the charts’ appearance as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rename a group</td>
<td>Right-click the group label and select <strong>Edit Label</strong> from the context menu.</td>
</tr>
<tr>
<td></td>
<td>When updating a group’s label, you must right-click the first instance of that group in the report area. If you right-click a subsequent instance of that group, the context menu doesn’t appear. For example, assume we are grouping a report based on the <strong>demo for adhoc</strong> Topic; if you group by city, you can only edit that group’s label by right-clicking the <strong>Aachen</strong> group; right-clicking the <strong>Albuquerque</strong> group does not open the context menu.</td>
</tr>
<tr>
<td>Remove a group</td>
<td>Drag the group above the report area. You can also right-click the group and select <strong>Remove from Table</strong> from the context menu.</td>
</tr>
<tr>
<td>Filtering</td>
<td>Right-click a group and select <strong>Create Filter</strong> to filter your results by the values in that field. Depending on the datatype of the field, the server creates a suitable filter, and selects an appropriate operator. For example, if you create a filter from a string field, the filter’s operator is set to <strong>is one of</strong>: select the items that interest you. You can also select a different filter operation by pointing at the filter’s <strong>•</strong> icon and selecting <strong>Toggle Operation to display the operation field.</strong></td>
</tr>
<tr>
<td>Filtering</td>
<td>You can’t create a filter by clicking elements in the chart.</td>
</tr>
<tr>
<td>Select the chart type</td>
<td>Right-click the chart, select <strong>Change Chart Type</strong> from the context menu, and click the desired type.</td>
</tr>
<tr>
<td></td>
<td>Pie charts can show only a single measure. Each member of the group is shown as a slice of the pie, and the size of each slice is the magnitude of the measure.</td>
</tr>
<tr>
<td></td>
<td>Scatter and time series charts don’t group data; each data point is displayed separately.</td>
</tr>
<tr>
<td>Select the display option for the chart type</td>
<td>Right-click the chart, select <strong>Display Options</strong> from the context menu, and click the desired option.</td>
</tr>
<tr>
<td></td>
<td>The options vary according to the chart type. For example, you can hide or show lines on a time series chart or align a bar chart horizontally or vertically.</td>
</tr>
<tr>
<td>Edit the chart legend</td>
<td>By default, the legend shows the field name and summary function of every measure in the chart.</td>
</tr>
<tr>
<td></td>
<td>To remove the legend, right-click the chart and select <strong>Legend and Labels &gt; Hide Legend</strong> from the context menu.</td>
</tr>
<tr>
<td></td>
<td>To add a legend to the chart, right-click the chart and select <strong>Legend and Labels &gt; Show Legend</strong> from the context menu.</td>
</tr>
<tr>
<td></td>
<td>To edit the legend, right-click a label, select <strong>Edit Legend Label</strong> from the context menu, and make your change in the text field.</td>
</tr>
<tr>
<td></td>
<td>Rearranging the labels in the legend rearranges them in the chart. Dragging a label out of the report area removes the corresponding measure.</td>
</tr>
<tr>
<td>Edit the group label</td>
<td>The group label is the name of the field used for grouping to create the series. It can’t be edited, but you can hide or show it. Right-click the chart and select <strong>Legend and Labels &gt; Hide Group Title</strong> or <strong>Legend and Labels &gt; Show Group Title</strong> from the context menu.</td>
</tr>
</tbody>
</table>
3.2.4.2 Groups

Summarized charts require a group to determine how the data are compared and summarized. You can’t select a group for an unsummarized chart but you can make the following choices:

<table>
<thead>
<tr>
<th>Select the group</th>
<th>Drag a field from the list of available fields to the Group control, or right-click the field in the list and select Add as Group or Replace Group from the context menu. Only one group can be selected at a time. The Ad Hoc Editor finds the unique values in the group, which is usually a non-numeric field such as city or country. However, any kind of field can be used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change the group</td>
<td>Drag a different group to the Group bar, or right-click the field and select Replace Group from the context menu.</td>
</tr>
<tr>
<td>Remove the group</td>
<td>Right-click the chart and select Remove Group from the context menu.</td>
</tr>
</tbody>
</table>

If you switch between report types, as described in section 3.2.1.3, “Choosing the Report Type,” on page 27, changing or removing the group in a chart will affect all row groups, even those that are not displayed in the chart. For example, if you have two nested row groups in a table, the outer group will be displayed when you switch to a chart. But if you change the group in the chart, both nested groups are replaced when you switch back to the table. Now the table only shows the one group added in the chart.

3.2.4.3 Measures

Options for displaying measures in a report include:

<table>
<thead>
<tr>
<th>Select a measure</th>
<th>Drag a field from the list of available fields to the report area, or right-click the field in the list and select Add as Measure from the context menu. Any number of measures can be added. The Ad Hoc Editor depicts the data for each group according to the type of chart. For example, in a scatter chart, the data are depicted as individual points arrayed across both axes of the chart; in a bar chart, data is summarized for each value grouping field, creating a series. Measures are typically numeric datatypes but any kind of measure can be added. For summarized charts, the Ad Hoc Editor selects the default summary type for each measure (based on the field type) but another summary type can be selected.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select a measure’s summary function</td>
<td>For summarized charts, right-click the chart, and select Measures &gt; [measure] &gt; Change Summary Function from the context menu and click the desired function. You can also right-click the measure’s name in the legend and select a function from the context menu. For more information on summary functions, see section 3.2.6, “Summaries,” on page 34.</td>
</tr>
<tr>
<td>Change the order of the measures</td>
<td>Drag-and-drop to rearrange the measure names in the legend. The measure are also rearranged in the chart itself.</td>
</tr>
</tbody>
</table>
3.2.5  Ad Hoc Crosstabs

JasperReports Server 4.2 includes two types of Ad Hoc crosstabs:
- Crosstabs based on Domains and Topics continue to be displayed in the style of previous releases.
- Crosstabs based on OLAP client connections offer a different user interface that more closely follows OLAP analysis conventions.

This section describes the former. For more information on the latter, refer to the JasperReports Server User Guide.

A crosstab summarizes data across row and column groups. When creating groups in a crosstab report, pay special attention to the order in which they appear in the Ad Hoc Editor. For example, it doesn’t make sense to group first by postal code then by country, because each postal code belongs to only one country. In this case, each country appears in the report multiple times: once for each postal code in that country. Change the order of groups by dragging them.

By default, a crosstab displays a subset of the data its query retrieves. Click Full Data to view the full dataset. This can be helpful in showing a more accurate view of the final report (by showing more data). Click Sample Data to return to the smaller subset and improve performance if you find the editor has slowed down.

3.2.5.1  Grouping

Row and column groups are symmetrical in crosstabs. All operations on row groups can also be applies to column groups. You can drag-and-drop or choose options from the context menu that appears when you right-click a column group. Common tasks with column groups include:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a group</td>
<td>Right-click the field in the list of available fields and select Add as Row Group or Add as Column Group. You can also add a group by dragging it to the Row Group or Column Group areas.</td>
</tr>
<tr>
<td>Remove a group</td>
<td>Select the row or column group by clicking its label, then drag it outside the report area. You can also remove groups using the context menu on the group label.</td>
</tr>
<tr>
<td>Set the granularity of date groups</td>
<td>When you select a date field as a group, you can specify the granularity of the group values. Right-click the group label, select Change Grouping and select Year, Quarter, Month, or Day.</td>
</tr>
<tr>
<td>Change the order of groups</td>
<td>Select the row or column group by clicking its label, then drag it to a new location. The other groups are highlighted as you drag over them to show where the selected group would go. You can also right-click the group label and select Move Right or Move Left for row groups or Move Up or Move Down for column groups.</td>
</tr>
<tr>
<td>Pivot one group</td>
<td>You can pivot any group from row to column or vice-versa to help adjust a crosstab for better readability. Select the row or column group by clicking its label, then drag it to the other area. You can also right-click the group label and select Move to Column Group for row groups or Move to Row Group for column groups.</td>
</tr>
<tr>
<td>Pivot entire crosstab</td>
<td>To pivot all row groups to column groups and vice-versa at the same time, click .</td>
</tr>
<tr>
<td>Sorting</td>
<td>By default, group values are always sorted alphabetically along both axes. Nested groups are also in alphabetical order under their parent values. Once you add measures, you can sort the crosstab by measure values for any given measure.</td>
</tr>
</tbody>
</table>
### Measures

Measures are the summarized values that populate the inner cells of your crosstab. Typically, measures are created from numeric fields in the Topic or Domain, but in some circumstances, it makes sense to use string fields when summarized as a count. For example, you might want to display the number of unique customers that made purchases in a given quarter.

You can drag-and-drop or choose options from the context menu that appears when you right-click a measure label.

To interact with measures, click (or right-click) the measure labels shown to the right of your row groups; clicking the measure value itself opens a new instance of the Ad Hoc Editor.

Common tasks with measures include the following:

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add a measure</strong></td>
<td>Right-click the field in the list of available fields and click <strong>Add as Measure</strong> on the context menu. You can also add a measure by dragging it to the Data Values area. Fields you add to the crosstab remain in the list of available fields so you can reuse them as necessary.</td>
</tr>
<tr>
<td><strong>Remove a measure</strong></td>
<td>Select the measure by clicking its label, and drag it outside the report area. You can also remove measures using the context menu on the measure label.</td>
</tr>
<tr>
<td><strong>Change the order of measures</strong></td>
<td>Select the measure by clicking its label, and drag it up or down to change the order of measures in each cell.</td>
</tr>
<tr>
<td><strong>Change Summary Functions</strong></td>
<td>Right-click a measure label and click <strong>Change Summary Function</strong> select a summary function, which determines how the value is calculated. For example, you may want to display an average rather than a total. For more information, see section 3.2.6, “Summaries,” on page 34.</td>
</tr>
</tbody>
</table>
### 3.2.6 Summaries

In table reports, each field can display a single summary calculation. In crosstab reports, each measure displays a summarized value. In charts, the type of chart determines whether measures are summarized. In general, you can change the summary function of any measure, with the exception of those that use the percent of total and percent of parent custom field functions.

Summaries show the result of a function applied to all data values. In table reports, they appear at the bottom of each group, as well as at the bottom of the report. In crosstab reports, they determine the values of the measures at the intersection of each row and column. In summarized charts, they determine the size or location of the graphical elements that represent your data.

In a table, summarize a column by right-clicking it and selecting **Add Summary** from the context menu. The summary function is automatically applied to all groups in the table. When a new group is added, it includes a summary for each column.

By default, JasperReports Server summarizes fields of each datatype as follows:

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Summary Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>Sum</td>
<td>Displays the sum of all values in the set.</td>
</tr>
<tr>
<td>Date</td>
<td>Count</td>
<td>Displays the total number of values in the set.</td>
</tr>
<tr>
<td>String</td>
<td>Count</td>
<td>Displays the number of values in the set.</td>
</tr>
<tr>
<td>Boolean</td>
<td>Count</td>
<td>Displays the number of values in the set.</td>
</tr>
</tbody>
</table>

You select from these options to set a measure’s summary function in any type of report:

<table>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
<th>Available for…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>Displays the average of all values in the set.</td>
<td>Numeric</td>
</tr>
<tr>
<td>Count All</td>
<td>Displays the number of rows in the set.</td>
<td>Boolean, Date, Numeric, and String</td>
</tr>
<tr>
<td>Distinct Count</td>
<td>Displays the number of unique values in the set.</td>
<td>Boolean, Date, Numeric, and String</td>
</tr>
<tr>
<td>Maximum</td>
<td>Displays the highest value in the set.</td>
<td>Numeric</td>
</tr>
<tr>
<td>Minimum</td>
<td>Displays the lowest value in the set.</td>
<td>Numeric</td>
</tr>
<tr>
<td>Sum</td>
<td>Displays the grand total for the set.</td>
<td>Numeric</td>
</tr>
</tbody>
</table>

On the **Table** and **Crosstab** tabs, you can remove the summaries:
• For tables, remove a column’s summary by right-clicking the column or the summary itself, and selecting Remove Summary from the context menu.
• For crosstabs, remove a column group’s summary by right-clicking the group label and selecting Delete Column Summary; remove a row group’s summary by clicking Delete Row Summary on the same context menu. These options are only available for the outer-most group on either axis.

3.2.7 Setting the Data Format

You can set the format of data throughout Ad Hoc. Click an object, such as a column or measure, and select Change Data Format from the context menu.

The options that appear in the menu are the formats available for objects of the given datatype. For example, for monetary datatypes, the menu might list $1,234.56, -$1,234.56 and ($1,234.56), while for date datatypes it might list December 31, 2008, and 12/31/2008. By default, non-integer fields use the -1,234.56 data format; integers use -1234.

In the report area, you can assign a data format to a field. In table reports, the format is applied to all rows as well as the group- and report-level summaries. In crosstab reports, the format is applied to the measures. In charts, it changes the legend or group label.

In some cases, the options in the Change Data Format menu are affected by the locale; for example:

3.2.8 Custom Fields

You can always create custom numeric fields in the editor, such as multiplying a field by an absolute number, subtracting one field from another, or displaying a field as a percent of a total for a column or group. You can add the fields to the report as you would any other numeric field.
As with any field, you can add a summary, change the data format, or change the display label of a custom field. You can also use custom fields in other custom fields to create more complex computed values.
3.2.9 Running and Saving Reports

From within the Ad Hoc Editor, you can run the report by clicking \( \text{ } \). This launches a new browser tab or window that shows the report in HTML format. You can export the report to a number of other formats, such as Adobe Acrobat (PDF) and Comma Separated Value (CSV). For more information, see the JasperReports Server User Guide.

To save a report, point at \( \text{ } \) and select Save Report. If this is a new report, you are prompted for the following details:

- **Name** – Enter a short name that uniquely identifies the report. For example, December 2008 Sales.
- **Description** (optional) – Enter text that explains the report’s content. If there are several similar reports in same folder, the description can help you and other users quickly find the right report. Good details to include are the Topic, Domain, or OLAP client connection upon which it is based, the fields you have included, and the question the report is meant to answer.
- **Location Folder** – Select a folder in the repository where the report will reside. Your choices are limited by the access you have in the repository.

If you are saving an existing report, the server doesn’t prompt you before saving. It simply overwrites the file in the repository. To save the report with a new name, point at \( \text{ } \) and select Save Report As. Then enter the same information as above. The next time you simply save the report, it will overwrite the new report name.

If you have unsaved work and you navigate away from the Ad Hoc Editor by selecting a main menu option (such as View > Repository), the editor will warn you that your current report will be lost. You can cancel to return to your report and save it if you wish.

If you navigate to different web site, there is no warning and the current report may be lost. Some browsers will return to the report with the back button.

3.2.10 Tips and Tricks

Here are some tips on working with the Ad Hoc Editor and formatting reports:

<table>
<thead>
<tr>
<th>Using spacers</th>
<th>To create space between the leftmost column and the group labels in a table report, drag a spacer to the leftmost position; the margin provides a buffer between the first column and the groups summaries.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding columns</td>
<td>To quickly add a column to a table, double-click the field name in the list of available fields.</td>
</tr>
<tr>
<td>Changing the report's source</td>
<td>When you start a report, you must select a Topic, Domain, or OLAP client connection. This choice determines the fields that are available for your report. If you want to start over with a different source of data, save your current report if desired, then hover your point to ( \text{ } ) at the top of the Fields list and select Change Source. If you start a new table, chart, or crosstab, the current report is lost. If you cancel the Source dialog, you return to the current report.</td>
</tr>
<tr>
<td>Changing fields</td>
<td>If your report is based on a Domain, click select above the list of available fields to change the fields that can be used in the report. You can add fields that weren’t previously used, as well as remove fields that aren’t referenced by custom fields and don’t currently appear in the report area.</td>
</tr>
<tr>
<td>Hiding context menus</td>
<td>If a context menu blocks your view, close it by clicking anywhere outside the menu or by pressing Escape.</td>
</tr>
</tbody>
</table>
3.3 Administering Ad Hoc Reports

While the previous section focused on end-user tasks, this section provides information for administrators who maintain Ad Hoc Topics and Domains and configure JasperReports Server, including:

- Administering Topics
- Administering Domains
- Administering Styles
- Ad Hoc Editor Configuration
- Scalability

For information about OLAP client connections, which can also form the basis of Ad Hoc reports, refer to the Jaspersoft OLAP User Guide and Jaspersoft OLAP Ultimate Guide.

3.3.1 Administering Topics

From a user perspective, Topics are sets of fields that can be added to a report. Topics provide a starting point for end-users using the Ad Hoc Editor. Under the covers, Topics are JRXML files that have been uploaded to a specific location in the repository. The reports in the /Ad Hoc Components/Topics folder populate the Topics tab that appears when users click Create > Ad Hoc Report.

The JRXML file that the Topic is based on must contain a query and a field list. For details about creating JRXML files, refer to the JasperReports and iReport Designer documentation, which is described in section 1.4.2, “Premium Documentation,” on page 11.

Perhaps the simplest way to create Topics is to create new JRXML files using iReport, then use the JasperReports Server plug-in for iReport to upload them to the repository. Note that iReport can edit Ad Hoc reports, but the grouping and sorting you defined in the Ad Hoc Editor are lost. For more information, refer to section 9.5.7, “iReport and the Ad Hoc Editor,” on page 122.

Any report layout in the Topic’s JRXML file is ignored by the Ad Hoc Editor. Jaspersoft recommends that a Topic’s JRXML file not include anything other that the query and field list.

When you create a JRXML file that will be used as a Topic, you can specify the name to display for each column that the Topic returns. To do so, define a field property named adhoc.display for each field declared in the JRXML. For more information, see the JasperReports Server User Guide.

3.3.2 Administering Domains

From a user perspective, Domains are sets of fields that can be configured and filtered before being added to a report. Domains provide a starting point for end-users using the Ad Hoc Editor. Under the covers, Domains are defined by a design that can be uploaded to the server and exported as XML. Unlike Topics, which must be stored in a specific folder in the repository, Domains are detected regardless of their location in the repository. The /Domains folder is included for your convenience, but the Domains tab in the Source dialog (which appears when users click Create > Ad Hoc Report) displays all the Domains to which you have access in the repository.

For details about creating Domains, refer to the JasperReports Server User Guide.

3.3.3 Administering Styles

Users select styles in the Ad Hoc Editor in order to determine the colors and general appearance of the table, chart, or crosstab. Each style is composed of two parts: the first determines the report’s appearance in the Ad Hoc Editor’s report area; the second determines the appearance of the final report when it is run or saved. Each style is defined by two files:

- When a report is created in the Ad Hoc Editor, its appearance is determined by cascading style sheets (CSS). The CSS files are located in the jasperserver-pro/adhoc/stylesheets directory and are named after the Ad Hoc styles. For example, the file olive.css defines the Olive style.
When a report is run or saved, its appearance is defined by a JRXML file that is located in the jasperserver-pro/WEB-INF/adhoc/themes directory. The JRXML files are also named after the Ad Hoc styles, for example olive.jrxml.

The location of these directories varies with your application server. If you are using the Apache Tomcat bundled with JasperReports Server, they are located in <js-install>/apache-tomcat/webapps/ on the host computer.

A style defines the colors and other visual aspects of each of its elements, such as font family, size, and weight. Colors are expressed as hexadecimal color codes typical of web applications. For more information about these colors, including a list of common codes, refer to Wikipedia’s web colors article.

### 3.3.3.1 Updating an Existing Style

You can modify the existing styles by specifying new fonts and colors. Updating a style includes these high-level steps:

1. Modifying the CSS that controls the style in the Ad Hoc Editor.
2. Modifying the corresponding JRXML that controls the style in the report output.
3. Modifying the name of the style.
4. Modifying the style’s thumbnail (that is, the graphic that represents the style in the Styles window).
5. Testing your changes.
6. Optionally, making the style a default for Ad Hoc reports.

The following instructions explain how to modify the Olive style.

#### To modify the CSS file:

1. Locate the olive.css file in your server installation (found in adhoc\stylesheets), and make a backup of the file.
2. Open the file in a text editor, such as Notepad, and update the definition of each element. Trial and error is often the simplest way to refine the style to suit your needs.
3. Save the file.

#### To modify the JRXML file:

1. Locate the olive.jrxml file in your server installation (found in WEB-INF\adhoc\themes), and make a backup of the file.
2. Open it in a text editor, and update the definition of each element to match the changes you made in the CSS file.
3. Save the file.

#### To rename a style:

1. Locate the adhoc_messages.properties file (found in WEB-INF\bundles), and make a backup of the file.
2. Open the file in a text editor, and locate the name of the Olive style:
   ```
   ADH_141_THEME_OLIVE=Olive
   ```
3. After this entry, add a new line and enter a new property and value. For example:
   ```
   ADH_1049_THEME_Pistachio=Pistachio Mist
   ```
4. Save the file.
5. If you support multiple languages, repeat step 1 through step 4 in the properties bundle file for each language.

#### To modify the thumbnail:

1. Locate the applicationContext-adhoc.xml file (found in WEB-INF), and make a backup of the file.
2. Open the file in a text editor, and locate the thumbnail definition for the Olive style:

```xml
<entry key="olive">
  <map>
    <entry key="display" value="ADH_141_THEME_OLIVE"/>
    <entry key="thumbnailTop" value="#003300"/>
    <entry key="thumbnailMiddle" value="#B7C18B"/>
    <entry key="thumbnailBottomLeft" value="#CECF9C"/>
    <entry key="thumbnailBottomRight" value="#CECF9C"/>
  </map>
</entry>
```

3. Edit the `display` value to point to the new property you added (ADH_1049_THEME_Pistachio) instead of the Olive label (ADH_141_THEME_OLIVE).

4. Edit the color definitions (such as #003300) to match the colors you defined in the JRXML and CSS files.

5. Save the file.


**To test the modified style:**

1. In the Ad Hoc Editor, create a test report.

2. In the tool bar, click Styles.

3. Verify that the thumbnail for the style you modified looks correct.

4. Verify that the new name is displayed for the style you modified. If you support multiple languages, verify each one.

5. Apply the modified style by clicking its name. The report area updates to reflect the new fonts and colors.

6. Review the new style to ensure that it meets your needs. If it doesn’t, edit the CSS file again and restart the server.

7. Click Run Report.

8. Review the new style in the report output to ensure that it meets your needs. If it doesn’t, edit the JRXML file again, and restart the server.

### 3.3.3.2 Creating a New Style

To add a new style to the ones provided by Jaspersoft, you must take steps similar to those described in section 3.3.3.1, “Updating an Existing Style,” on page 39:

1. Create the JRXML that controls the style in the report output. Save the file to WEB-INF/adhoc/themes in your server installation. Refer to the style JRXMLs included with JasperReports Server to see how to create your own.

2. Create the corresponding CSS that controls the style in the Ad Hoc Editor. Save the file to adhoc/stylesheets in your server installation. Again, refer to the existing CSS files to see how to create your own.

3. Edit the adhoc_messages.properties file (in WEB-INF/bundles) to add a label as the name of the style.

4. Edit the applicationContext-adhoc.xml file (in WEB-INF) to add the style’s thumbnail.

5. Test the new style in the editor and when run.

In many cases, it’s easier to copy and modify an existing style’s JRXML and CSS than to start from scratch.

### 3.3.3.3 Changing the Default Styles

Whether you have a custom style or rely on the existing styles in Ad Hoc, you can set which style is the default for each type of report. There are four defaults: one that applies to all report types and one for each type (table, chart, and crosstab). All can be changed. The defaults for the report types override the overall default. For instance, if the overall default is Corporate and the report type defaults are not supplied by the user, the Corporate style applies to all reports. If the default for chart reports is changed to Olive, the Olive style applies to chart reports and the Corporate style applies to all other reports.
Default styles for Ad Hoc reports are defined in the `<js-webapp>/WEB-INF/applicationContext-adhoc.xml` configuration file. This sample shows how to make Olive the default style for charts in the `<js-webapp>/WEB-INF/applicationContext-adhoc.xml` file:

```xml
<property name="chartDefaults">
  <map>
    <entry key="theme" value="olive"/>
    ...
  </map>
</property>
```

The following table shows the properties to change in the configuration file. Restart the server after changing them.

<table>
<thead>
<tr>
<th>Report Type</th>
<th>Property</th>
<th>Entry Key</th>
<th>Value As Installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>All types</td>
<td>defaultTheme</td>
<td>none</td>
<td>default</td>
</tr>
<tr>
<td>Table</td>
<td>tableDefaults</td>
<td>theme</td>
<td>default</td>
</tr>
<tr>
<td>Crosstab</td>
<td>crosstabDefaults</td>
<td>theme</td>
<td>default</td>
</tr>
<tr>
<td>Chart</td>
<td>chartDefaults</td>
<td>theme</td>
<td>default</td>
</tr>
</tbody>
</table>

### 3.3.4 Ad Hoc Editor Configuration

The following properties are among those that can be configured in the `WEB-INF/applicationContext-adhoc.xml` file.

The repository URI locations are relative to the organization so that these settings apply to all organizations in the server instance. For example, for a user in the default organization, the URI `/adhoc` actually refers to `/organizations/organization_1/adhoc`.

**Table 3-1 Configurable Properties in WEB-INF/applicationContext-adhoc.xml**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JrxmlScriptURI</td>
<td>The location in the file system of the state2jrxml.js script, which generates the JRXML report based on the current Ad Hoc Editor selections. By default, this file is located in the <code>/adhoc</code> folder of the repository.</td>
</tr>
<tr>
<td>realmsURI</td>
<td>The repository location where Topics should reside. The default is <code>/adhoc/topics</code>.</td>
</tr>
<tr>
<td>themesURI</td>
<td>The location in the file system where the JRXML files that define the report’s style when it is run. The default is <code>WEB-INF/adhoc/themes</code>.</td>
</tr>
<tr>
<td>defaultTheme</td>
<td>The name of the default style for Ad Hoc reports. This name must match a style defined in both a CSS and a JRXML file. The default is <code>default</code>.</td>
</tr>
<tr>
<td>aruFolder</td>
<td>The repository location where users are allowed to save their Ad Hoc reports. The default is <code>/</code>. This allows your users to save Ad Hoc reports anywhere. If you have a folder specifically for user content, specify this folder; for example, <code>/userreports</code>.</td>
</tr>
</tbody>
</table>
3.3.5 Scalability

When you open reports in the Ad Hoc Editor, the query’s entire result set is retrieved. If your Topics, Domains, or OLAP client connections return large result sets, your server instance requires more memory. Similarly, many actions users take in the Ad Hoc Editor (such as adding groups) require the server to re-sort the data on the server; if the query returns a large result set, the user may notice a delay when making such changes.

To reduce delays, decrease the memory requirement, and support more users, Jaspersoft recommends that the queries in Topics, Domains, and OLAP client connections return a reasonable amount of data for your hardware and system capacity.

If you encounter scalability or performance issues around Ad Hoc reporting, examine the server’s memory usage first. Then consider optimizing your Topic and Domains queries.

A Domain very often returns a large volume of data, which can impact performance of the Domain designer, the Ad Hoc Editor, and the final reports. If your Domain’s performance is slow, try the following remedial steps:

- Set your data policies and other Ad Hoc settings to reasonable values for your data and system configuration. In the case of data policies, you can configure the server to use less memory but perform more queries, which may perform better under certain circumstances. For more information, refer to the JasperReports Server Administrator Guide.
- The Java Virtual Machine (JVM) that the server runs in may need to be configured for higher memory usage. For information, refer to the documentation associated with your JVM. For more information, refer to the JasperReports Server Installation Guide.
- Use the smallest number of joins that creates the data islands you need. Complex join relationships can impact performance drastically.
- If your end-users experience time out messages or reports that seem to never complete, consider changing the governors defined for reports. For more information, refer to the JasperReports Server Administrator Guide.
- Use filters to narrow the data returned by Domains or Domain Topics to improve performance of reports based on large or complex Domains: limit the initial load time of a Domain by defining a filter in the Choose Data wizard. Set it to prompt to allow your users to edit it or remove it in the Ad Hoc Editor’s filter pane.
- When running a report that relies on a Domain, the server uses the filters and security defined for the particular report and user to limit the query. Running many reports with slightly different security or filters may cause problems with the server’s memory usage. In this case, similar data is duplicated in memory, which can impact performance. Increasing the memory allocated to the application server that hosts JasperReports Server can mitigate this issue.

CHAPTER 4  CUSTOM DATA SOURCES

JasperReports Server provides built-in support for many commonly used data sources, such as JDBC, JNDI, and JavaBeans, as described in the JasperReports Server Administrator Guide. However, JasperReports Server does not include all JasperReports Library data sources, and you may want to use a custom JasperReports Library data source. In either case, you can extend JasperReports Server to support additional data sources by adding files to your configuration.

In order to perform many of the tasks described in this section, you must have the administrator role and administrator access to the computer where JasperReports Server is installed.

This chapter contains the following sections:

- Data Sources in JasperReports Library
- Overview of the Custom Data Source Examples
- Prerequisites and Installation of the Examples
- Creating a Custom Data Source
- Installing a Custom Data Source

4.1 Data Sources in JasperReports Library

While a JasperReports Library data source is a different object from a JasperReports Server data source, they work together closely:

- A JasperReports Library data source is an implementation of the JRDataSource interface that provides data organized in rows and columns to the JasperReports Library filler; it produces a JasperPrint object. Each field declared in the JRXML corresponds to a column in the JRDataSource output.

- A JasperReports Server data source is a persistent object in the repository; it is typically created by stepping through a wizard. The data source stores properties that tell JasperReports Server how to create a JRDataSource (typically in collaboration with a JRQueryExecuter). These properties vary with the type of data source; for example, a JDBC data source needs a JDBC driver, URL, user, and password. A data source can be defined as a public repository object that can be used by any report unit (for example, the repository includes the /datasources/JserverJdbcDS if you installed the sample data), or as a local object defined during the creation of a specific report unit.

When JasperReports Server receives a request to run a report unit, it maps the report unit’s data source to an implementation of ReportDataSourceService, which returns a JRDataSource based on the data source’s persistent properties. The JRDataSource is used to fill the report and produce a JasperPrint object, from which the server generates HTML or other supported output formats.
Each JasperReports Server data source implementation must support the following features:

- Read and write persistent properties in the JasperReports Server repository.
- Provide a user interface for creating and editing instances that are integrated with the JasperReports Server web interface.
- Create a JRDataSource using the property values for a specific data source instance, or pass parameters to a JRQueryExecuter that produces the JRDataSource.

JasperReports Server’s built-in data sources rely on several Java classes, along with specialized Spring bean files, WebFlow configurations, message files, and JSP files. The custom data source framework provides the same functionality by using a Spring bean file, a message catalog, and a minimum of one Java file (more are required to support optional features).

**4.1.1 Query Executers**

A query executer is an implementation of the JRQueryExecuter interface in JasperReports Library. It interprets the queryString in the JRXML and produces a JRDataSource. JasperReports Library (either standalone or running in JasperReports Server) determines which query executer to use by looking at the language attribute of the queryString and looking up a query executer factory registered for that language.

JasperReports Server data sources can use two different methods to create a JRDataSource:

- The JasperReports Server data source can create the a JRDataSource directly, without a queryString in the JRXML; or
- The server can pass implementation-specific objects to the query executer through the report parameter map. The query executer then uses the objects from the parameter map, as well as the contents of the queryString, to create the JRDataSource.

Selecting the method to use depends on the nature of the data source, as well as whether you want to use a queryString to control your data source. A good example of a data source using a query executer is the JDBC data source: it passes a JDBC connection to the JDBC query executer, which it uses to pass the SQL queryString to the database.

The examples described in the following sections demonstrate both methods:

- The custom bean data source creates a JRDataSource directly, which returns a hard-coded list of JavaBeans.
- The webscraper data source can either create a JRDataSource directly, using the properties supplied by the data source instance, or it can get those properties from a queryString in the JRXML. In this case, a data source instance isn’t required. The sample reports for this data source each demonstrate one of these approaches.

**4.2 Overview of the Custom Data Source Examples**

Jaspersoft provides two example custom data sources:

- Custom Bean Data Source
- Webscraper Data Source

The examples are found in the <js-install>/samples/customDataSource directory where you installed JasperReports Server. Once you have deployed JasperReports Server to your application server, you can use Ant to build and deploy the examples.

**4.2.1 Custom Bean Data Source**

The custom bean data source implementation creates a data source from a collection of Java beans declared in the source code. Its Spring bean definition file is in <js-install>/samples/customDataSource/webapp/WEB-INF/applicationContext-sampleCDS.xml. Jaspersoft provides an example report that uses this data source; it is called simpleCDS.jrxml and is located in the <js-install>/samples/customDataSource/reports directory.

**4.2.2 Webscraper Custom Data Source**

The webscraper custom data source implementation fetches a web page, decodes its HTML, and extracts selected data that is turned into field values in the data source. Its Spring bean definition file is located in <js-install>/samples/customDataSource/webapp/WEB-INF/applicationContext-webscraperDS.xml.

The example reports for this data source read a web page from [http://www.craigslist.org](http://www.craigslist.org) and extract a list of items for sale.
The webscraper data source configuration includes these elements:

- **URL**: An HTTP URL that refers to the HTML page containing the desired content.
- **DOM path**: An XPath expression that locates HTML elements to be turned into rows in the data source.
- **Field paths**: XPath expressions for each field defined in the JRXML. JasperReports Server uses these paths to locate the field value in each row selected by the DOM path.

The implementation creates a data source by:

- Using the URL to issue a GET request for an HTML page.
- Using the DOM path to select XML elements from the converted response.
- Creating a new data source row for each selected element.
- Determining the context for each field based on its field path.

The data source takes two parameters: the URL of the web page and the XPath that determines how elements in the HTML page become rows in the data source. The parameters can either be specified by a data source definition in the repository or by a query string in the JRXML. JasperReports Server includes sample reports that each show one of these approaches:

- The `<js-install>/samples/reports/webscrapertest.jrxml` report has no query. Instead, it relies on an instance of the custom data source that you must create in the repository. Typical parameters to use with this data source are:
  
  **URL**: `http://sfbay.craigslist.org/search/car/eby?query=&neighborhood=62`
  
  **DOM Path**: `/html/body/blockquote[2]/p`

- The `<js-install>/samples/reports/webscraperQEtest.jrxml` example contains a `queryString` element that specifies the URL and the DOM path. It should be used without defining a data source instance, because JasperReports Server doesn’t run the query executor for this particular implementation if a data source is defined for the report unit.

### 4.3 Prerequisites and Installation of the Examples

#### 4.3.1 Java Development Kit

Because you must recompile the Java source files, you need the Java Development Kit (JDK). Ensure that the JAVA_HOME environment variable points to a full JDK installation.

#### 4.3.2 About Apache Ant

If you used an installer to install JasperReports Server, you have Ant installed already. Run Ant using the following command:

```bash
<js-install>/ant/bin/ant <ant-arguments>
```

If you installed JasperReports Server manually with a WAR file, you must download Ant from http://ant.apache.org. Ant 1.6.2 was used for testing, but earlier versions may also work.

#### 4.3.3 Installation

Each sample directory includes:

- **build.xml**: The Ant build file.
- **src**: Java source directory.
- **webapp**: A directory containing other files required by the examples, such as JSPs and Spring configuration files, which are copied directly to the JasperReports Server web application directory.
- **reports**: A directory containing example JRXML files that use the sample custom data sources.

**To install the samples in your JasperReports Server web application:**

1. At the command line, change directories to the custom data source sample directory (<js-install>/samples/customDataSource).
2. Edit build.xml and set the webAppDir property to the root of the JasperReports Server web application.
3. Run the Ant command (as described in 4.3.1, “Java Development Kit,” on page 45) with no arguments; this executes the default target, which is named deploy. The deploy target initiates these actions:
   - Compiles the Java source under the src directory.
   - Deploys the compiled Java class files to the web application.
   - Deploys files under the
     - webapp directory to the web application.
4. Restart the application server.

These steps only make the example custom data sources themselves available in JasperReports Server. To test the data sources, you must also create instances of the custom data sources in JasperReports Server, then upload the reports that accompany the samples.

### 4.4 Creating a Custom Data Source

A custom data source consists of Java code, a message catalog, and a Spring bean definition file that configures all the parts of the implementation with JasperReports Server. This section describes the implementation of a custom data source.

#### 4.4.1 Implementing the ReportDataSourceService Interface

A custom data source requires an implementation of the ReportDataSourceService interface, which sets up and tears down data source connections in JasperReports Server. It relies on:

- `void setReportParameterValues(Map parameterValues)`: called before running a report; it creates resources needed by JasperReports Library to obtain a JRDataSource, and adds them to the parameter map.
- `void closeConnection()`: cleans up any resources allocated in setReportParameterValues().

#### 4.4.2 Defining Custom Data Source Properties

A custom data source can define properties that help users configure each data source instance differently, in the same way that a JDBC data source has properties for JDBC driver class, URL, user name, and password. While implementing your ReportDataSourceService, Jaspersoft recommends that you consider which properties you’ll need.

There are two kinds of properties:

- **Editable properties** that must be string values. When you use the JasperReports Server data source wizard to create an instance of your custom data source, you can enter values for the editable properties using text fields. These values are persisted when you save the data source.
- **Hidden properties** that can be of any type. These property’s values are determined by the Spring configuration file: they are not persisted, nor are they visible in the data source wizard. Use them to give your ReportDataSourceService implementation access to a Spring bean instance.

---

**Files Used by a Custom Data Source Implementation**

<table>
<thead>
<tr>
<th>Type</th>
<th>Path (relative to web application directory)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring bean definition</td>
<td>WEB-INF/applicationContext-&lt;name&gt;.xml where &lt;name&gt; uniquely identifies your custom data source</td>
<td>Defines Spring beans needed to configure the data source.  Choose a unique name starting with applicationContext- and ending with .xml</td>
</tr>
<tr>
<td>Message catalog</td>
<td>WEB-INF/bundles/&lt;cat_name&gt;.properties where &lt;cat_name&gt; uniquely identifies your custom data source</td>
<td>Defines messages used by the data source implementation (this path is referenced in the Spring bean definition file).</td>
</tr>
<tr>
<td>Implementation classes</td>
<td>WEB-INF/lib or WEB-INF/classes</td>
<td>Any Java code required by the implementation.</td>
</tr>
</tbody>
</table>
For an example of both types of properties, see the custom bean data source definition in the XML example in section 4.4.5, “Defining the Custom Data Source in Spring,” on page 48.

These property values are set by the custom data source framework after it instantiates your ReportDataSourceService implementation. You need property setters and getters corresponding to each property name; for example, if you defined a property with the name foo, you need getFoo() and setFoo() methods.

### 4.4.3 Implementing Optional Interfaces

If you want to use the value of the queryString in the JRXML to obtain your data source, you must create implementations of the JRQueryExecuter and JRQueryExecuterFactory interfaces.

<table>
<thead>
<tr>
<th>Optional Interfaces</th>
<th>Method to Implement</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JRQueryExecutorFactory</td>
<td>createQueryExecuter(JRDataset dataset, Map parameters)</td>
<td>Returns a JRQueryExecuter for the given dataset and parameter map.</td>
</tr>
<tr>
<td>JRQueryExecutor</td>
<td>JRDataSource createDatasource()</td>
<td>Returns the actual data source based on the parameter map passed to the JRQueryExecuterFactory; most likely, you will create a JRDataSource implementation suitable for your data source.</td>
</tr>
<tr>
<td></td>
<td>close()</td>
<td>Called when the report filling process is done with the data source.</td>
</tr>
<tr>
<td></td>
<td>cancelQuery()</td>
<td>Called to clean up resources if the report filling process is interrupted.</td>
</tr>
<tr>
<td>CustomDataSourceValidator</td>
<td>validatePropertyValues(CustomReportDataSource ds, Errors errors)</td>
<td>Use this to provide validation in the JasperReports Server data source creation wizard. It checks parameters and calls errors.rejectValue() with the appropriate property name and error code (defined in a message catalog; for more information, refer to 4.4.4, “Creating the Message Catalog,” on page 47).</td>
</tr>
</tbody>
</table>

### 4.4.4 Creating the Message Catalog

The message catalog contains messages displayed by JasperReports Server’s data source wizard when creating and editing custom data source instances. The various types of messages are shown in the following table, along with message naming conventions:

<table>
<thead>
<tr>
<th>Messages about Instances of Custom Data Sources</th>
<th>Naming Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of the custom data source type</td>
<td>Cdsname.name (where cdsname is the value of the name property of the custom data source).</td>
</tr>
<tr>
<td>Name of the custom data source property</td>
<td>Cdsname.properties.propname (where propname is the name of the property that the user must define when creating a custom data source).</td>
</tr>
<tr>
<td>Validation messages</td>
<td>Cdsname.any.message.code (JasperReports Server does not enforce a convention, but Jaspersoft recommends starting with cdsname for consistency).</td>
</tr>
</tbody>
</table>
For example, the webscraper message catalog contains the following:

```java
webScraperDataSource.name=Web Scraper Data Source
webScraperDataSource.properties.url=URL
webScraperDataSource.properties.path=DOM Path
webScraperDataSource.url.required=A value is required for the URL
webScraperDataSource.path.required=A value is required for the DOM path
```

### 4.4.5 Defining the Custom Data Source in Spring

To configure your data source, you must add an instance of `CustomDataSourceDefinition` to the Spring bean definition file. This class has the following properties:

<table>
<thead>
<tr>
<th>Properties of <code>CustomDataSourceDefinition</code> Class</th>
<th>Required</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>factory</td>
<td>Yes</td>
<td>A fixed value of <code>ref=&quot;customDataSourceFactory&quot;</code> This bean manages all the custom data sources.</td>
</tr>
<tr>
<td>name</td>
<td>Yes</td>
<td>A unique name that identifies this data source to the custom data source framework. It is also used as a prefix for all messages in the message catalog. Choose the name that is not used by other custom data sources.</td>
</tr>
<tr>
<td>serviceClassName</td>
<td>Yes</td>
<td>A class name for your <code>ReportDataSourceService</code> implementation.</td>
</tr>
<tr>
<td>validator</td>
<td>—</td>
<td>An instance of your <code>CustomDataSourceValidator</code> implementation.</td>
</tr>
<tr>
<td>propertyDefinitions</td>
<td>—</td>
<td>Information describing each property used by the data source implementation, structured as a list of maps.</td>
</tr>
<tr>
<td>queryExecutorMap</td>
<td>—</td>
<td>Map with query languages (uses the language attribute of JRXML <code>queryString</code> element) as keys and <code>JRQueryExecuterFactory</code> class names as values.</td>
</tr>
</tbody>
</table>

The `propertyDefinitions` property is a list of maps, each one describing a property of the custom data source implementation. It includes these entry keys:

<table>
<thead>
<tr>
<th>Entry Keys for <code>propertyDefinitions</code> Property</th>
<th>Required</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Yes</td>
<td>Name of property that matches a Java Bean property in the <code>ReportDataSourceService</code> implementation; it is also used in message catalog keys.</td>
</tr>
<tr>
<td>default</td>
<td>—</td>
<td>A default value for the property.</td>
</tr>
<tr>
<td>hidden</td>
<td>—</td>
<td>If a property has the <code>hidden</code> entry key set to <code>true</code>, then its value is fixed to that of the <code>default</code> entry key. Such properties are not be editable in the JasperReports Server data source wizard, nor are they persisted. This is handy for making Spring beans accessible to <code>ReportDataSourceService</code> implementations.</td>
</tr>
</tbody>
</table>
The following XML defines a CustomDataSourceDefinition bean for the custom bean data source example:

```xml
<bean id="myCustomDataSource"
     class="com.jaspersoft.jasperserver.api.engine.jasperreports.util.CustomDataSourceDefinition">
    <property name="factory" ref="customDataSourceServiceFactory"/>
    <property name="name" value="myCustomDataSource"/>
    <property name="serviceClassName" value="example.cds.CustomSimplifiedDataSourceService"/>
    <property name="validator">
        <bean class="example.cds.CustomTestValidator"/>
    </property>
    <property name="propertyDefinitions">
        <list>
            <map>
                <entry key="name" value="foo"/>
            </map>
            <map>
                <entry key="name" value="bar"/>
                <entry key="default" value="b"/>
            </map>
            <map>
                <entry key="name" value="repository"/>
                <entry key="hidden" value="true"/>
                <entry key="default" value-ref="repositoryService"/>
            </map>
        </list>
    </property>
</bean>
```

4.4.6 Configuring the Message Catalog

To configure your message catalog, add a bean definition like the following to the Spring definition file that you created in “4.4.5, “Defining the Custom Data Source in Spring,” on page 48”:

```xml
<bean class="com.jaspersoft.jasperserver.api.common.util.spring.GenericBeanUpdater">
    <property name="definition" ref="addMessageCatalog"/>
    <property name="value" value="WEB-INF/bundles/cdstest"/>
</bean>
```

For the value property, substitute the location of your message catalog file, omitting the .properties extension. Setting the addMessageCatalog property precludes the need to edit the messageSource bean definition in applicationContext.xml. Note that, if you also supply localized versions of the message catalog that follow the Java conventions for naming resource bundles, users with other locales automatically see the localized strings when creating a new data source of this type.

4.5 Installing a Custom Data Source

To install your custom data source in JasperReports Server, add all the files it requires to the server web application directory. For the correct locations, refer to the table, “Files Used by a Custom Data Source Implementation,” on page 46. After adding the files, restart JasperReports Server.
When you create a new data source in JasperReports Server, the new custom data source type appears in the list of available data source types. If the new type is selected, JasperReports Server displays a form containing the list of properties you configured.

When the form is submitted, the parameter values are validated with your `CustomDataSourceValidator` implementation and appropriate validation messages are displayed. Once the data source is validated, save it to the repository. The data source can now be used in a report or analysis connection.

When defining `<queryString>` in JRXML, use a `language` setting that your custom data source supports.

When you add a report to the repository, you can define a local data source or you can select one of the data sources in the repository. In either case, you can use a data source based on your custom data source implementation. In the case of a data source in the repository, you must create it before adding the report. If, during the creation of your data source, the custom data source is not listed as an available data source type, the custom data source is not properly installed.
CHAPTER 5  QUERY-BASED INPUT CONTROLS

Query-based input controls display a dynamic set of values for the user to choose from. They are input control resources in the repository, but instead of being based on a datatype or a static list of values, they perform a query to retrieve a list of values. For example, a report could have a city parameter, and the query-based input control could display the list of cities that exist in your data. Because the queries use standard syntax, you can include filters in a WHERE clause. In the previous example, you could restrict the list of cities to a certain country.

By including parameters, you can also create cascading input controls. A cascading input control is one whose choices depend on the selection of a previous input control. For example, after the user selects a country, the values available for selecting a city are restricted to the chosen country. Cascading input controls are query-based controls that contain parameters returned by other controls.

Cascading input controls help make input controls easier to use and faster to display. Certain parameters in reports have a natural hierarchy, such as countries and cities or years and quarters, and the cascading input controls let the user find values based on this hierarchy. Instead of selecting cities from one large list that may need to scroll, users can make a selection from a smaller list where all choices are visible. Also, displaying long lists make the web page slow to load, so cascading input controls that reduce the size of the list make it faster to load. If there were an especially large number of cities, more cascading input controls could be used to reduce the list, such as region or state. The values for each control are loaded only when the previous input has been selected, making for a convenient and speedy user experience.

The parameter values determined by each cascading input control may or may not be used in the report. For example, if the report only shows data about a city, the country input control exists only to speed up the choice of city. However, if the report also shows information such as city average compared to country average for a given measure, the country parameter is also used in the report.

In order to perform many of the tasks described in this section, you must have the administrator role.

Creating standard input controls is documented in the *JasperReports Server Administrator Guide*. This chapter explains how to create input controls that are based on a query and parameterized by the results of another control, thus creating the cascading effect.

This chapter contains the following sections:

- Creating a Query-based Input Control
- Built-in Parameters for Query-based Input Controls
- Domain-based Queries
- Cascading Input Controls
5.1 Creating a Query-based Input Control

In this first example, we create a query-based input control that returns a long list of all cities for the user to choose from.

1. Log in as an administrator.
2. Browse the repository and select the folder where you want to create the query-based input control.
3. Right-click the folder and select Add Resource > Input Control. The Add Input Control dialog appears:

![Adding an Input Control - Naming](image1)

4. Select the type of query-based input control from the type drop-down list. This choice determines how the input control appears to users, either as a drop-down list, a set of radio buttons, a multi-select list, or a set of check boxes. In this example, we choose a single-select query-based input control.
5. Specify the prompt text, parameter name, optional description, and appearance options in the same manner as when defining a regular input control.
6. Click Next. Because we selected one of the query-based types, the Locate Query page appears:

![Adding an Input Control - Locating the Query](image2)
If you have a suitable query resource defined in the repository, you could select it here as an external reference. In this example, we’ll define a query resource locally inside the input control resource.

7. Click **Next** to define the local query resource. The query naming dialog appears:

![Query Naming Dialog](image)

**Figure 5-3 Adding an Input Control - Naming the Query**

Although the query resource is not visible in the repository, it may still have a name, ID and optional description within the query resource. However, the values for these fields are not important.

8. Enter any name, and the ID is filled in automatically. Then click **Next**. The data source link page appears:

![Data Source Link Page](image)

**Figure 5-4 Adding an Input Control - Linking to a Data Source**

As with all query resources, the query resource inside the input control may optionally link to a data source, either in the repository, or its own internally defined one. If no data source is linked, the query in the input control uses the same data source as report. In this example, we take the default selection of not linking to a data source.
9. Click **Next**. The query definition page appears:

![Figure 5-5 Adding an Input Control - Defining the Query](image)

10. Select the query language, in this example SQL, and enter a query string. The SELECT statement should contain the names of all fields used in the display, value, or filter for the input control. In this example, the query returns three fields, country, state, and city, and the country field is used to limit the values to a single country. The ORDER BY clause ensures that the values from the query are sorted alphabetically when they appear in the input control.

For an example in a different query language, see section 5.3, “Domain-based Queries,” on page 56.

11. Click **Save** to complete the query definition. The parameter values page appears:

![Figure 5-6 Adding an Input Control - Setting Parameter Values](image)

On the parameter values page, you define which field in the results of the query are displayed, and which field contains values that will become the parameter value when chosen.

a. First, specify the value column, which is the field whose value is passed to the report. The data type of the field must match the type of the corresponding parameter in the report.

b. Next, specify the visible columns, which are the fields whose values appear in the input control that the user will choose from. In the simplest case, enter same field as the value column. If you add multiple fields to the visible
columns, the input control will display the fields together, in the order listed, separated by a vertical bar (|). In the example in Figure 5-6, the user may see and choose from:

Los Angeles | CA  
San Francisco | CA  
Denver | CO  

Only the city value (without the state) will be passed to the report. Showing additional field in this way can help users find the value they want in long lists of results.

The value and display columns may also be entirely different, for example, displaying the full name of a sales representative, but using the employee ID as the value returned by the input control. The only restriction is that all fields used in the value or display list must be selected by the query.

5.2 Built-in Parameters for Query-based Input Controls

The LogInUser and LogInUsername parameters are always available for query input controls; they are always available to reports, as well, even if an input control isn’t defined for them. The standard parameters are also provided for reports if they are defined as parameters in the JRXML.

Table 5-1 Built-in Parameters for Query-based Input Controls

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogInUser</td>
<td>User</td>
<td>The user that is currently logged in. This parameter isn’t available in query input controls, but is used as parameter to the report.</td>
</tr>
<tr>
<td>LogInUsername</td>
<td>String</td>
<td>The user name of the current user.</td>
</tr>
<tr>
<td>LogInUserFullName</td>
<td>String</td>
<td>The full name of the current user.</td>
</tr>
<tr>
<td>LogInUserEmail</td>
<td>String</td>
<td>The email address of the current user.</td>
</tr>
<tr>
<td>LogInUserEnabled</td>
<td>Boolean</td>
<td>Indicates whether the current user is enabled.</td>
</tr>
<tr>
<td>LogInUserExternallyDefined</td>
<td>Boolean</td>
<td>Indicates whether the current user is authenticated externally.</td>
</tr>
<tr>
<td>LogInUserTenantId</td>
<td>String</td>
<td>In the commercial editions, the name of the organization of the current user.</td>
</tr>
<tr>
<td>LogInUserRoles</td>
<td>Collection&lt;String&gt;</td>
<td>The roles assigned to the current user. This is helpful for parameters that use $X.</td>
</tr>
<tr>
<td>LogInUserAttributes</td>
<td>Map&lt;String, String&gt;</td>
<td>The profile attributes of the logged-in user. This parameter isn’t usable in query input control, but it is used as parameter to the report. If the user has no attributes, the parameter is an empty map.</td>
</tr>
<tr>
<td>LogInUserAttributeNames</td>
<td>Collection&lt;String&gt;</td>
<td>The names of the profile attributes of the logged-in user. This is helpful for parameters that use $X. If the user has no attributes, the parameter is an empty map.</td>
</tr>
<tr>
<td>LogInUserAttributeValue</td>
<td>Collection&lt;String&gt;</td>
<td>The values of the profile attributes of the logged-in user. This is helpful for parameters that use $X. If the user has no attributes, the parameter is an empty map.</td>
</tr>
<tr>
<td>LogInUserAttribute_&lt;attribute-name&gt;</td>
<td>String</td>
<td>For the logged-in user, the value of the attribute matching the name passed as &lt;attribute-name&gt; (like att1). If there is no match, the parameter is empty. This parameter is only available if it is defined in a query or as a report parameter.</td>
</tr>
</tbody>
</table>
5.3 Domain-based Queries

In the case of reports that use a Domain as the data source, any query-based input controls must contain a query against the Domain. When defining the query as shown in Figure 5-5 on page 54, set the query language to **Domain**.

The query language Domain ("sl") is selected when opening Domain-based queries created in JasperServer 3.5 or earlier. It is used only for backward compatibility and should not be selected for new Domain-based queries.

Domain queries have their own special syntax, the same that is used in the Domain design. A Domain-based query references fields, called items, by their item IDs, along with any set IDs that determine the path of the item within the Domain. For example, if you want your query input control to return a list store cities, where the field with ID `ej_store_store_city` is nested in the set with ID `expense_join_store`, you would use the following Domain query:

```xml
<query>
  <queryFields>
    <queryField id="expense_join_store.ej_store_store_city" />
  </queryFields>
</query>
```

The list contained inside the `<queryFields>` tag in a Domain query is equivalent to the fields given in the SELECT statement of an SQL query. Given the query above, you can create an input control for a Domain-based report that lets the user select a city as a parameter to the report.

Sometimes, you may want the input control to display more information than the actual value returned. As with standard query-based input controls, you can select more fields, and then display those fields in your input control. For example, to make the list of cities unambiguous, you could include the state and country in your display. In that case, the Domain-based query must also retrieve those items:

```xml
<query>
  <queryFields>
    <queryField id="expense_join_store.ej_store_store_city" />
    <queryField id="expense_join_store.ej_store_store_state" />
    <queryField id="expense_join_store.ej_store_store_country" />
  </queryFields>
</query>
```

Then, when specifying your visible query columns, as shown in Figure 5-9, “The COUNTRY Input Control,” on page 60, you would add the 3 fields to the list in the order you want them to appear. When specifying fields in the list of visible query columns, use the full ID of the field, including any set IDs. For example, the following list of fields:

```
expense_join_store.ej_store_store_country
expense_join_store.ej_store_store_state
expense_join_store.ej_store_store_city
```

will create a list of values such as the following for users to choose from (the separator | is added automatically):

USA | CA | Los Angeles
USA | CA | San Francisco
USA | OR | Portland
USA | WA | Redmond
Finally, the Domain-based query also has the option to filter the query results, as shown in the following example:

```xml
<query>
  <queryFields>
    <queryField id="expense_join_store.ej_store_store_city" />
    <queryField id="expense_join_store.ej_store_store_country" />
    <queryField id="expense_join_store.ej_store_store_state" />
  </queryFields>
  <queryFilterString>expense_join_store.ej_store_store_country == 'USA' and expense_join_store.ej_store_store_state == 'CA'</queryFilterString>
</query>
```

The `<queryFilterString>` tag contains a DomEL (Domain Expression Language) expression that references the full ID of the fields, including any set IDs. For more information about DomEL, see the JasperReports Server User Guide. The `<queryFilterString>` tag in a Domain query is equivalent to the WHERE clause of an SQL query. The list of fields in the `<queryFields>` tag must include all fields being referenced in the filter string.

### 5.4 Cascading Input Controls

A cascading input control is one whose values depend on the selection made in a previous input control. Cascading input controls are created by using parameters in the query string of a related input control. In other words, the parameter defined by an input control may be used in another input control.

In the query-based example of cities and states such as:

- Los Angeles | CA
- San Francisco | CA
- Denver | CO

the query may still generate a list of hundreds of cities to scroll through. Even though each city is easy to identify with the state, scrolling through a long list is time consuming. With cascading input controls, this example would have two input controls, one for the state and one for the city:

- When input controls are displayed, the query for the state input control returns an alphabetical list of unique state names.
- When the user selects a state, the query for the city input control is triggered and returns the list of cities for that state. The cities are displayed in the input control, and when the user selects one and submits it, the city name is passed as a parameter to the report.

The user makes two selections from much shorter lists, which is easier and quicker than using one long list of city and state names. The second input control is empty, showing no selections, until clicking on the first of the cascading input controls. If the user selects a different state in the first control, the list of cities in the second control updates accordingly.

Parameter substitution in query input controls follows the same approach as for JasperReports queries. Queries of all types of query connections can use parameter substitution, and $P, $P! and $X (for SQL queries) parameters are supported. For more information on using $P, $P! and $X to build dynamic queries, refer to the JasperReports Ultimate Guide and the iReport Ultimate Guide.

In almost all cases, the parameters appear in the filter (WHERE) clause of the query. Single-select query input controls return single values that are referenced with the $P syntax, and multi-select query input controls return collections that are handled by the $X syntax.

> When defining these parameters in a report, don’t use a `defaultValueExpression` element. Due to a limitation in JasperReports Server, these parameters are null when a `defaultValueExpression` is provided.
5.4.1 Parameters in Input Control Queries

The example in this section shows how to create cascading input controls for selecting a country and a city. This is done by writing the query of the second input control (city) with a syntax that references a parameter name. The syntax is the same used by JasperReports in the report queries. A parameter is referenced using the following convention:

\$P{parameter name}

So if we have an input control called COUNTRY, the query to get the cities from a hypothetical table called ACCOUNTS looks like this:

\select city from ACCOUNTS where country = \$P{COUNTRY}

When the user selects a country from the COUNTRY input control, the result is used to perform the query of the CITY input control, and the CITY input control is refreshed to show the result.

There are two additional ways to use a parameter in a cascading input control query. The first is used when the value held by parameter_name is not a simple value, but a chunk of the query (or in extreme cases even the whole query). It has the syntax:

\$P!{parameter_name}

With the \$P!{} syntax, the value of the parameter is treated as raw text. The server replaces the placeholder with the value of the referenced parameter without performing extra checking and value escaping, as is done when using the plain \$P{} syntax.

Secondly, there is the \$X{} syntax that is used when the value of a parameter is a collection. In the country/cities example, we can allow the user to pick any number of countries, and show all the cities in the selected countries. Now, the cities are selected in a multi-select input control that returns a collection, and the \$P{} syntax is insufficient for substituting a collection into an SQL query. The \$X{} syntax appears as follows:

\select city from ACCOUNTS
where \$X{IN, country, COUNTRIES}

When the user selects the values Canada, Mexico, and USA in the COUNTRIES multi-select input control, the \$X{} syntax translates into the following query for the CITIES input control:

\select city from ACCOUNTS
where country IN ('USA','Canada','Mexico')

\$X{} takes three positional arguments:
- First is the collection operator, either IN or NOT IN.
- Second is the table column that is being compared.
- Third is the parameter that provides the collection of values, in other words, the name a multi-select input control.

The number of parameters that can be used in a query is arbitrary, just as the number of input controls that can be defined in a JasperReport is arbitrary. In addition to the standard input control parameters, a cascading input control query can use the built-in parameters described in Table 5-1, “Built-in Parameters for Query-based Input Controls,” on page 55.

5.4.2 Step-by-Step Example

In this example, we’ll create a simple report that displays all the accounts of a city, using the SugarCRM sample database shipped with JasperReports Server. This example uses iReport to create a report and the JasperReports Server plug-in to create the input control resources in the repository.

We start by creating a report with a parameter called CITY and the following report query:

\select * from accounts where billing_address_city = \$P{CITY}

In the detail band, we add three fields: name, shipping_address_city, and shipping_address_country. Then we publish the report on the server, using the Publish tool of the JasperReports Server plug-in in iReport.
Now define the input controls. Right-click the JasperReport node in the Repository Navigator and add the first input control by selecting **Add > Input Control**. This input control shows the list of countries in which accounts are present. It is not a cascading input control, but its value is used in the next control: the one that selects the city.

Set the name of this first input control to COUNTRY (the display name can be “Country”). Set the Input Control type to **Single Select Query** (this because we want to get the countries using an SQL query, but since this is just a common input control, we may use any other type of input control, like a list of values or even a multiple-select list of values).

Edit a local resource for the query, set a name for it (“query”) and set the query language to **SQL**.
The query is just a simple query to select the countries. For instance:

```sql
SELECT DISTINCT shipping_address_country
FROM ACCOUNTS
ORDER BY shipping_address_country
```

To complete the local query resource, set the repository resource /datasources/JServerJdbcDS as the query’s data source. Finally, in the **Value and Visible Columns** tab, set the Value Column to `shipping_address_country` and make it (the only) visible column. The first input control, which selects the country, is now ready.

Now that we have an input control named COUNTRY, we can reference the COUNTRY parameter in any query-based input control. This is what we are going to do with the second input control. Set its name to CITY. Its definition is similar to the COUNTRY control, so the type must be **Single Select Query**. The query resource must be of type **SQL**; it’s used in the COUNTRY parameter’s **where** condition:

```sql
SELECT DISTINCT shipping_address_city
FROM ACCOUNTS
WHERE shipping_address_country = $P{COUNTRY}
ORDER BY shipping_address_city
```

This time the column to be used in the Value and Visible Columns field is `shipping_address_city`.

When you run the JasperReport, if everything has been correctly configured, the dialog box in **Figure 5-10 on page 60** appears. It consists of the two simple input controls, and the CITY control is not populated until the user selects a country.
In JasperReports Server, access control is determined by two separate mechanisms, authentication and authorization. Authentication is the process of verifying user credentials to give access to the server. Authorization is the process of checking permissions granted to users and roles to access resources within the server. More generally, authentication determines who can access JasperReports Server, and authorization determines what they see.

For example, a user might bookmark the repository page; if the user’s session has timed out when he clicks the bookmark, the server prompts for the login name and password. The authentication mechanism compares these to the existing user accounts and if a match is found, the user is logged in and redirected to the repository page. The user session includes any roles stored in the user account, and these roles are processed by the authorization mechanism to determine what the user is allowed to see in the repository. Roles determine much of what users see, including the menu options, pages, folders, reports, repository objects, and even the data they can view.

This chapter contains the following sections:

- Authentication
- Authorization
- Authorization Example

### 6.1 Authentication

To perform authentication, JasperReports Server integrates Spring Security, part of the Spring framework. Spring is an open source security solution for assembling J2EE components by way of configuration files. This section describes how Spring Security performs authentication, its default behavior, and some options you have for customizing it.

The default configuration of Spring Security represents the most common use cases, but Spring Security’s flexibility can provide you with very fine-grained control of the security system. However, since Spring Security can be configured in so many ways, you may find its configuration daunting. This section is only meant as an introduction to get you started in thinking about customizing JasperReports Server’s security system. Refer to section 6.1.3, “Further Documentation,” on page 63 for more information.

### 6.1.1 Spring Security Filter Chain

The main entry point for authentication in Spring Security is the `securityFilter`, which is configured in WEB-INF/web.xml. As with all J2EE servlet filters, the `securityFilter` is invoked for every HTTP request for paths in this web application, whether the request comes from users or automated agents. The `securityFilter` is essentially a proxy for Spring Security’s `FilterChainProxy` class, which handles the actual logic that performs authentication. In JasperReports Server, this class is instantiated as a Spring bean defined in the WEB-INF/applicationContext-security.xml file. For
implementations with security requirements that differ from the default behavior, the FilterChainProxy can be configured to support other authentication schemes. Finally, Spring Security allows the server to accept authentication from an external authentication mechanism.

The configuration of the filterChainProxy bean determines how the server enforces authentication on its web pages:

```xml
<bean id="filterChainProxy"
     class="org.springframework.security.util.FilterChainProxy">
   <property name="filterInvocationDefinitionSource">
     <value>
       CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
       PATTERN_TYPE_APACHE_ANT

       /xmla=httpSessionContextIntegrationFilter,basicProcessingFilter,
        JIAuthenticationSynchronizer,anonymousProcessingFilter,
        basicAuthExceptionTranslationFilter,filterInvocationInterceptor

       /services/**=httpSessionContextIntegrationFilter,basicProcessingFilter,
        JIAuthenticationSynchronizer,anonymousProcessingFilter,
        basicAuthExceptionTranslationFilter,filterInvocationInterceptor

       /**=httpSessionContextIntegrationFilter,authenticationProcessingFilter,
        basicProcessingFilter,JIAuthenticationSynchronizer,anonymousProcessingFilter,
        exceptionTranslationFilter,filterInvocationInterceptor
     </value>
   </property>
</bean>
```

Each expression has the format URIpattern=filter1,filter2,... so that a list of processing filters is defined for any URI exposed by the server. Each filter is given the request and response context and has a chance to take action before control is passed to the next filter. You can even define and add your own filter if you need to implement a custom authentication schema, although the packaged options are sufficient in almost every case.

You’ll notice that the /xmla and /services/** expressions are configured with one set of filters, while all other URLs have a different set of filters. The expressions are configured this way because the first two (that represent the XML for Analysis servlet and the web services, respectively) are designed to receive SOAP requests from other servers, while all other server URLs are designed for humans using web browsers; for example, only humans are asked to proceed to the login page.

The filter beans have the following functions:
- httpSessionContextIntegrationFilter – Stores the SecurityContext in the HttpSession.
- authenticationProcessingFilter – Redirects browsers to the login page, then handles the username and password credentials entered by the user.
- basicProcessingFilter – Handles HTTP BASIC style authentication, where the username and password are passed as an HTTP header.
- JIAuthenticationSynchronizer – Notifies JasperReports Server which user has logged in; this is not a standard Spring Security class.
- anonymousProcessingFilter – Marks the user as anonymous if no other authentication occurred.
- exceptionTranslationFilter – Sets a 403 Forbidden HTTP response code if the user is not authenticated to view the requested page.
- filterInvocationInterceptor – Integrates the filter chain with other Spring Security components.

### 6.1.2 Authentication Processing Filter

As seen in the previous section, a request for JasperReports Server pages from a user is processed through the filterChainProxy where it is handled by the authenticationProcessingFilter. This filter gathers the user’s credentials from the login page and processes them with the authenticationManager bean.
The `authenticationManager` bean is configured with a list of authentication providers who can verify the credentials. The default configuration is as follows:

```xml
<bean id="authenticationManager" class="org.springframework.security.providers.ProviderManager">
  <property name="providers">
    <list>
      <ref local="daoAuthenticationProvider"/>
      <ref local="anonymousAuthenticationProvider"/>
    </list>
  </property>
</bean>
```

The property list includes the following:

- `daoAuthenticationProvider` – Compares the username and password (passed from the login form) to those stored in the server’s private database. This is the default form of authentication.
- `anonymousAuthenticationProvider` – Grants anonymous access to the login page itself.

If you use a Lightweight Directory Access Protocol (LDAP) directory, you can configure an LDAP provider bean by adding it to the list of providers in the `authenticationManager` bean. The LDAP provider bean must be of the class `org.springframework.security.providers.ldap.LdapAuthenticationProvider`.

When JasperReports Server uses an external authentication service such as LDAP, JIAuthenticationSynchronizer ensures that any user that is authenticated successfully is created if it does not already exist in the server. Roles defined in the external authentication service are also created if they do not exist in the server. To grant access to users, these roles must be associated with repository resources, such as reports and analysis views.

Assuming the credentials are valid, the `daoAuthenticationProvider` finds a match in the user database and creates the user session under the given username. It also reads the roles from the user account information and stores them in the user session. The combination of username and assigned roles can then be used by authorization mechanism every time the user requests access to a resource, such as a report, in the repository.

### 6.1.3 Further Documentation

For more background on how Spring Security works and can be configured, including extensions for external authorization and single sign-on, refer to the following documents:

- *JasperReports Server External Authentication Cookbook*
- *Spring Security Reference Documentation*
- Spring Webflow 2.0 has quite different syntax for flow files than previous versions, so if a user has custom flow, it needs to be upgraded. Spring offers a special tool, WebFlowUpgrader, which helps to upgrade the flows. For more information, see the *Spring Security chapter on upgrading from 1.0*.

### 6.2 Authorization

In JasperReports Server, all resources are stored in the repository in a hierarchical structure of folders. Each resource in the repository has a set of permissions explicitly granting certain kinds of access. Folders also have permissions, and all contents of a folder inherit these permissions if they do not explicitly define their own. Often, many resources need the same permissions, and therefore it is easier to manage permissions on folders.

Permissions can grant access to users or to roles. Roles are groups of users created for the purpose of simplifying authorization. Often, it is easier to manage permissions for groups of users, and then manage role membership separately. However, if necessary, permissions can be granted to specific users.

The following sections describe how users, roles, and repository objects interact in the authorization mechanism. Section 6.3, “Authorization Example,” on page 68 demonstrates how these entities are created and modified, and how authorization
works in practice. For procedural documentation about managing user, roles, and repository objects, see the *JasperReports Server Administrator Guide*.

### 6.2.1 Repository Design for Access Control

The repository is the main storage facility for resources in JasperReports Server. Resources are represented as objects in the repository, contained in folders. Folders give structure to the repository and let you organize your resources in a logical manner. Folders can be nested to any level to support any organization of resources. The following section provides some guidelines for designing your repository.

Access control and the organization of the repository are inter-dependent. It is important to understand their interactions before you configure them.

Approach configuration of the repository and its access control with the following guidelines in mind:

- Determine how best to organize the repository. Define folders to group the repository objects that require similar permissions. This simplifies access control maintenance. A straight-forward repository organization also encourages reuse of report units and resources.
- Abstract out the similarities of your users to create roles that can be combined to grant the required level of access.
- Revisit the repository organization scheme you originally devised and modify it according to your findings regarding your role requirements.
- Create the roles for the permissions you determined were needed, then create the folders of the repository.
- Populate the repository with the report units and resources that define your reports.
- Set the role-level permissions for each folder, and in special cases, for report units and resources. You may also want to specify user-level access in some cases, as well.
- Test your access control model by logging into the server as various users with different roles using Login as User. Correct any problems you find.
- As new objects are added to the repository, be sure to define their permissions.
- When you add new roles, be sure to restrict the access they grant before assigning them to users.
- If your instance hosts multiple organizations, edit the Template Folder to include the resources each organization should have. Take this step before adding your additional organizations, so that they are populated with the correct resources.
- As your repository grows, delete report units and resources that are no longer in use. If you organize your repository to maximize reuse, you may not need to clean up your repository often.

### 6.2.2 Roles

Roles are a mechanism for grouping users for the purpose of assigning permissions in bulk. Administrators assign roles to each user, and permissions in the repository name the roles to which they grant access.

When you define your roles, start by defining one role for each unique type of user in your enterprise. For example, you might have roles for administrators, DBAs, developers, accountants, executives, and other end users. Once you determine the various sets of rights you need to define, you can sometimes compare them and abstract out sub-sets of permissions. Then assign each user a set of roles to provide the exact level of access needed. This approach simplifies the maintenance of access control.

Consider an example in which there are three different user types: users that can view revenue reports, users that can view cost reports, and users that can view both. You could define two roles: Role A, which is granted access to revenue reports, and Role B, which is granted access to cost reports. Users who only need to view revenue reports would be assigned Role A, users who need to view cost reports would be assigned Role B, and users who need to view both revenue and cost reports would be granted both roles.

If a user is assigned roles in which access rights conflict, access is granted. For example, consider a user with two roles: one role revokes access to view revenue reports and the other grants that same access. The user would have access to the revenue reports.
If a user does not have a role, he cannot log in; a user must at least have ROLE_USER, ROLE_ADMINISTRATOR, or ROLE_SUPERUSER in order to access the repository. JasperReports Server has the following default roles:

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROLE_ADMINISTRATOR</td>
<td>This role grants all object-level access, as well as granting access to the administrator home page and the Manage menu (except for the Ad Hoc Settings and Analysis Settings). This role is automatically granted to the jasperadmin user (who administers each organization) and superuser.</td>
</tr>
<tr>
<td>ROLE_ANONYMOUS</td>
<td>This modifiable role is automatically granted to the anonymoususer and, to any agent accessing the server without logging in; by default it only grants access to the login page.</td>
</tr>
<tr>
<td>ROLE_DEMO*</td>
<td>This role grants access to the SuperMart demo Home page, sample reports, and, if you have Jaspersoft OLAP, sample analysis views. These objects are available only if you installed the sample data when you installed JasperReports Server.</td>
</tr>
<tr>
<td>ROLE_ETL_ADMIN</td>
<td>This role was included in previous versions of the server to support integration with Talend Information Suite Enterprise Edition (TIS EE). It only appears if you have upgraded from a version of the server earlier than 3.1 (then called JasperServer). ROLE_ETL_ADMIN enables users of the commercial editions to access the Talend Integration Suite Enterprise Edition (TIS EE) administrator’s login page (on the Home page, click Manage &gt; JasperETL). The command appears only for users who have the ETL administrator role. For details, see the JasperReports Server Administrator Guide.</td>
</tr>
<tr>
<td>ROLE_PORTLET*</td>
<td>This role is assigned to users who are created automatically by the server when a portal such as Liferay contacts it with an authentication request. If the specified user name does not exist in the server, it is created, assigned the password of the user in the portal, and given ROLE_PORTLET and ROLE_USER.</td>
</tr>
<tr>
<td>ROLE_SUPERMART_MANAGER*</td>
<td>This role manages permissions for the sample data. It demonstrates data security features available in commercial editions of JasperReports Server.</td>
</tr>
<tr>
<td>ROLE_SUPERUSER*</td>
<td>This role grants all object-level access, as well as granting access to the administrator Home page and the Manage menu, including the Ad Hoc Settings and Analysis Settings. This role is automatically granted to the superuser (who administers the entire system).</td>
</tr>
<tr>
<td>ROLE_USER</td>
<td>Every non-administrator user that logs into the server must have this role. This role is automatically granted to every new user.</td>
</tr>
</tbody>
</table>

* JasperReports Server commercial editions only.

For more information about creating and assigning roles, refer to the JasperReports Server Administrator Guide.

### 6.2.3 Users

Everyone who accesses JasperReports Server needs a user account that defines his online session. Administrators assign roles to user accounts to determine their access to data and functionality. By default, the server contains the following users:

<table>
<thead>
<tr>
<th>User Name (Default Password)</th>
<th>Default Roles</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anonymousUser (none)</td>
<td>ROLE_ANONYMOUS</td>
<td>Special end user; included in minimal server setup. By default, this user cannot log in.</td>
</tr>
<tr>
<td>CaliforniaUser* (CaliforniaUser)</td>
<td>ROLE_USER ROLE_SUPERMART_MANAGER*</td>
<td>Special end user; included in sample data.</td>
</tr>
</tbody>
</table>
### 6.2.4 Assigning Permissions

Every repository object and folder either defines or inherits a permission for every role and every user. The permission determines the level of access that is granted to the members of the role or directly to the user.

Permissions are set by administrators through the repository interface. For example, the following illustration shows the interface to view and set permissions on a folder. You can set a separate permission for each role and for each user. If no permission is defined explicitly, the permission inherited from the enclosing folder applies:

*JasperReports Server commercial editions only.*

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>demo*</td>
<td>ROLE_USER ROLE_DEMO</td>
<td>Special end user; included in sample data.</td>
</tr>
<tr>
<td>jasperadmin</td>
<td>ROLE_ADMINISTRATOR</td>
<td>Administrator user, or organization admin; included in minimal server setup.</td>
</tr>
<tr>
<td>joeuser</td>
<td>ROLE_USER</td>
<td>Standard end user; included in sample data.</td>
</tr>
<tr>
<td>superuser*</td>
<td>ROLE_SUPERUSER* ROLE_ADMINISTRATOR</td>
<td>Special system administrator user.</td>
</tr>
</tbody>
</table>

*Figure 6-1 Example of all Permissions on a Folder*
A permission that appears with an asterisk (*) is inherited from its parent folder, which may in turn be inherited from a higher folder. Permissions that are greyed out cannot be modified, for example the administrator cannot remove his own access to a file (otherwise it would become impossible to manage). Possible permissions are the same for roles as for users and include:

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Access</td>
<td>Users or roles with this permission cannot see or access this resource.</td>
</tr>
<tr>
<td>Execute Only</td>
<td>Users or roles with this permission cannot see this resource, but they may run a report that accesses the resource.</td>
</tr>
<tr>
<td>Read Only</td>
<td>Users or roles with this permission can read but not update or delete the resource.</td>
</tr>
<tr>
<td>Delete + Read</td>
<td>Users or roles with this permission can delete and read the resource.</td>
</tr>
<tr>
<td>Write + Delete + Read</td>
<td>Users or roles with this permission can update, read, and delete the resource.</td>
</tr>
<tr>
<td>Administer</td>
<td>Users or roles with this permission can change the permissions defined for this resource, as well as read, update, and delete it.</td>
</tr>
</tbody>
</table>

Through a combination of role-based permissions and explicit user-based permissions, a user can have several permissions defined for a single resource. In this case, the most permissive is granted to the user. For example, if a user belongs to a role that has Read Only access to a folder, but the user has also specifically been granted Write+Delete+Read permission, the person has Write+Delete+Read access to the resource.

For security purposes, when a new user or role is added, its permission at the root of the repository is No Access, which is thus inherited on every folder and object. You must explicitly grant the user or role access to folders or resources by setting their permissions in the repository.

### 6.2.5 Design Recommendations

Before you define access control for JasperReports Server, consider these tips:

- If you plan to have a large number of objects in the repository, model your roles and repository organization on paper before configuring them in the server. It’s helpful to refer to a visual representation of the access control model when you create it.
- Map your roles along meaningful lines that will be easy to apply to new objects. Job functions within your company can provide some guidance; for example, you may want to create a separate role that provides access to financial reports that should only be available to your Accounting department. By granting this role view permissions on the appropriate reports and assigning it to your accounting users, you create an easily-extensible organization: new Accounting users will be assigned this role, and new accounting report units and resources will grant view permissions to the role.
- Create folders to group reports that have similar access requirements, then grant access at the folder level. This simplifies maintenance and gives you default locations for additional reports in the future.
- Take a business-oriented approach to access control, and ensure that users’ level of access is based on their job functions.
- Choose a repository organizing principle that makes sense for your application and business model. The obvious choice is often best, as it is usually the most intuitive for your users.
- The relationships between objects in the repository and the roles you define determine the level of access your users have. When modeling access control, focus on these relationships.
- Remember that your repository and access control model will evolve through system usage. A straight-forward repository organization scheme is easier to maintain and extend.
- Abstract out common privileges between user types and create roles that can be combined to grant the proper access to each user.
- Do not over-complicate your object permissions and roles. You can create a very complicated access model using very simple roles and assigning each user complimentary roles. The simpler your roles and the fewer object-level customizations you make, the easier it will be to troubleshoot and maintain access control. Note that there is a trade-off to consider, as well: having a large number of very simple roles can be cumbersome to maintain. Strike a balance between simplicity of role permissions and proliferation of roles.
- The majority of your access control model should be implemented through roles. Ensure that the roles you define will generally grant your users the correct permissions.
• Minimize the need for customization and use inheritance to your advantage by defining access for your top-level folders that is sensible for the majority of the repository. Setting most of your object’s access to None Defined will then define sensible access throughout the repository. Then customize the objects that cannot use the inherited set of permissions.
• If you must define user-level access control on objects, do so sparingly. While such customizations shouldn’t impact performance, their maintenance may be tedious and prone to errors.
• When you create a new role, you must manually grant access to any folders and objects that users with the new role should see.
• Don’t customize the access granted to ROLE_USER, unless the change should apply to every one of your users. While the server properly enforces whatever access you define for this role, your maintenance will be easier if you create new roles to augment the ROLE_USER role rather than directly editing ROLE_USER.
• When you add a new object or folder, it inherits the access control policies of its parent folder. Be sure that the inherited privileges are reasonable and necessary for the new object, and change them if they aren’t.

6.3 Authorization Example

The example presented in this section can help you understand the authorization mechanism in JasperReports Server. It describes a simplified repository structure, users, roles, and objects for a fictitious company called MyCompany.

This example does not explain data-level authorization. For an example of securing the data returned by Domains, refer to Chapter 7, “Securing Data in a Domain,” on page 77. For an example of securing the data returned by an analysis view, refer to the Jaspersoft OLAP Ultimate Guide.

If you’d like to follow along in your own test installation of JasperReports Server, you can download and import a ZIP file that includes the example’s users, roles, and objects. The reports in the example use sample data that is optional when installing the server. You do not need the sample data to demonstrate authorization.

6.3.1 Downloading and Importing the Authorization Example

If you decide to import the example, Jaspersoft recommends that you import it into a test installation of JasperReports Server.

The example is available in a ZIP archive called js-authorizations-example-<ver>.zip that can be downloaded from one of two locations:
• On the Jaspersoft Technical Support Portal under the Documentation Downloads section for the commercial editions.
• From this direct link on JasperForge.org.

The version of the zip is currently 3.5, but it has also been tested to work with JasperReports Server 4.2. The following procedure describes how to import the ZIP file’s contents using the command-line import utility.

To import the example:
1. JasperReports Server can be running or stopped when using the import utility. If it is running, you do not need to stop it, you can import resources on the fly.
2. Copy the js-authorization-example.zip to a temporary location on your JasperReports Server host computer.
   For simplicity, the following steps assume that you saved the file directly in the <js-install>/buildomatic directory where the import utility is located.
3. At the command line, change directories to the location of the import utility. For example:
   
   Windows:  cd "C:\Program Files\jasperserver[-pro]-<ver>\buildomatic"
   Linux:  cd /home/<user>/jasperserver[-pro]-<ver>/buildomatic
4. Run the import command, and specify the example ZIP. For example:
   
   Windows:  js-import.bat --input-zip js-authorization-example.zip
   Linux:  js-import.sh --input-zip js-authorization-example.zip
The import command lists the roles, users, folders, and resources that were successfully imported. It may return warning messages; you can ignore them. Unless the import utility returns errors, your repository now includes the example objects.

Refer to the JasperReports Server Administrator Guide for information on running the import utility.

6.3.2 Understanding the Example

MyCompany represents a fictional company with a set of users organized into departments; users in most departments cannot view other departments’ reports:

- Sales users can only view Sales reports.
- Shipping users can only view Shipping reports.
- Accounting oversees the other departments, so its users can view all reports in any department.

Let’s look at the general structure of the example, and see whether MyCompany’s authorization requirements are met.

To review the example:

1. If the server isn’t running, start it, and login as jasperadmin; the default password is jasperadmin.
2. Click View > Repository and expand the folders to see the contents of the new /Reports/MyCompany folder.
   Since we are logged in as the administrator, we can view everything in the default organization. The My Company folder contains a separate folder for each department:

   ![My Company Example Folders](image)

   **Figure 6-2  My Company Example Folders**

3. Click Manage > Roles to view the roles that are defined at MyCompany:

   ![My Company Example Roles](image)

   **Figure 6-3  My Company Example Roles**

In commercial editions, The MyCompany roles are defined in the default organization, which has the ID organization_1. In addition to the default roles, MyCompany has defined a role for each department: ROLE_ACCOUNTING, ROLE_SALES, and ROLE_SHIPPING.
4. Click **Manage > Users** to see the users defined at MyCompany.

![User List]

**Figure 6-4  My Company Example Users**

In addition to the default users, MyCompany has six users, two per department. In this example, the users’ first name is their user ID and their last name indicates their department, for example Sally Accounting works in the Accounting department.

Now we can test the permissions by logging in as each user to see the folders they can access.

5. Click Xiang’s user name and click **Login as User**.

   The login name at the top-right shows that you are logged in as “jasperadmin as Xiang Sales.” You can now see the JasperReports Server and its repository as the user Xiang would see it, including all permissions and restrictions.

6. Click **View > Repository**, expand the /Reports/My Company folder, and select the Sales Department Folder.

![Repository View]

**Figure 6-5  Xiang Can See the Three Reports in the Sales Department Folder**

Xiang is a member of the Sales department and has ROLE_SALES. The permissions only allow him to see the Sales Department folder and view the three sales reports it contains.

When you run the reports in this example, they all produce the same output. This example demonstrates only access permissions, and the reports are not meant to represent actual sales, shipping, or accounting reports.
7. Click **Logout** to return to the administrator’s account.
8. Select Friedrich among the users and click **Login as User**.
9. Click **View > Repository**, expand the /Reports/My Company folder again, and select the Shipping Department Folder.

   Friedrich is a member of the Shipping department and has ROLE_SHIPPING. He can only see the Shipping department folder and view the three shipping reports it contains.

10. Click **Logout** to return to the administrator’s account again.
11. Finally, select Sally among the users and click **Login as User**.
12. Click **View > Repository**, expand the /Reports/My Company folder, and select the Accounting Department Folder.

   Sally can see all three department folders.
Sally is a member of the Accounting department and has ROLE_ACCOUNTING. The permissions allow her to see all three example folders: her own department folder (Accounting) as well as those of the Sales and Shipping departments. She can run any report in any of these folders.

13. Click **Logout** to return to the administrator’s account again.

Let’s look at the user accounts, roles, and folders in this example to understand how they are set up.

14. Click **Manage > Roles** and select ROLE_ACCOUNTING.

---

**Figure 6-8 ROLE_ACCOUNTING Users**

Krishnan and Sally, both from Accounting, have this role. Hovering over their names gives a tool tip with their full name and organization. This role assignment defines the two users as members of the Accounting department and gives them all the permissions assigned to ROLE_ACCOUNTING.

15. Click **View > Repository** and expand the /Reports/My Company folder to see the department folders.

16. Right-click the Accounting Department Folder and select **Permissions...** from the context menu.

**Figure 6-9 Permissions for My Company Accounting Folders**

For the Accounting Department Folder, most roles are set to No Access. However, the ROLE_ACCOUNTING role is set to Read Only, which explains why Sally could see this folder but the users in the other departments could not.

17. Click **Cancel**, then right-click the Sales Department Folder, and select **Permissions...** from the context menu.
The Sales Department Folder gives read permission to both ROLE_SALES and ROLE_ACCOUNTING. This explains why Sally could also see the sales reports but Friedrich in shipping could not.

In both cases, administrators automatically have access as well, which is why the folders are also visible in this procedure when logged in as jasperadmin.

6.3.3 Modifying the Example

As in many authorization cases, the My Company example focuses on folder permissions. Because of inherited permissions, all reports in the example folders are secured with the same policy as the folder itself, so we can define the access policy once on the folder and avoid managing permissions for each individual report.

In this section, we’ll create a user with unique authorization requirements. A new employee in the Shipping department needs special access to Sales data, as well as the usual access to reports in his own department.

Since the new user is the only one in the company with this unique need, it doesn’t make sense to modify an existing role or create a new one. Instead, we’ll assign the user to the Shipping role, and grant him the additional permissions he needs. However, unlike the other permissions in this example, he will not have access to a whole folder, but only the specific report he needs.

To modify the example with a special Shipping user:

1. Login as jasperadmin.
2. Click Manage > Users and click Add User...
3. Enter the user information as shown in Figure 6-11 below:
   - User name: Giovanni Shipping.
   - User ID: automatically filled in from the user name, but edit this to Giovanni.
   - Email: leave blank.
   - Password and confirmation: use any value.
   - Leave User is enabled checked.
4. Click Add User to organization_1.
   Giovanni appears in the list of users on the users page. Let’s assign him to the Shipping department by assigning the corresponding role to him.
5. Select Giovanni and click Edit.
6. In the list of Roles Available, double click ROLE_SHIPPING to assign it to Giovanni.
7. Click Save in the user properties to save the new role assignment.
   From the current permissions on the department folders, we know that Giovanni, like Friedrich, can now access the Shipping Department Folder and all the reports it contains. But Giovanni also needs special access to sales data for the southern region.
8. Click View > Repository, expand the /Reports/My Company folder, and select the Sales Department Folder.
9. In the right-hand list of resources, right-click the South Region Sales Report and select Permissions... from the context menu. The Assign Permissions by Role page appears. Notice that ROLE_SHIPPING has no access to this report.
10. Click User near the top of the permissions dialog and set Giovanni’s permission to Read-only.
Remember that permissions are additive, so Giovanni will have access based on this permission by user, even if he is not granted access by his roles.

11. Click **Apply** to save your changes and return to the repository.

12. Click **Manage > Users**, click Giovanni’s name, and click **Login as User**.

13. Click **View > Repository** and expand the /Reports/My Company folder.

   We did not change the permissions on the Sales Department Folder, so Giovanni cannot see that folder or its contents. To find the sales report that he has access to, Giovanni must search the repository.

14. Enter **south** in the search field at the top of the page and click the search icon.

   A search applies to all resources in the repository, regardless of the permission on the folders. As shown in the following figure, Giovanni has permission to view and run the South Region Sales Report, even though he cannot browse the Sales Department Folder.

![Figure 6-13 My Company Example User With Special Permissions](image)

This example shows how easy it is to set up permissions to protect reports on the basis of roles and folder permissions. It also shows that you can just as easily change permissions to handle special access cases on a per-user basis, a per-resource basis, or both.
CHAPTER 7   SECURING DATA IN A DOMAIN

This section describes functionality that can be restricted by the software license for JasperReports Server. If you don’t see some of the options described in this section, your license may prohibit you from using them. To find out what you’re licensed to use, or to upgrade your license, contact Jaspersoft.

You may need to restrict access to the data in a Domain when it is accessed by different people. For example, you may allow managers to analyze data across their department but only allow individual contributors to see data related to themselves. For this purpose, Domains support security files.

When Domain security is properly configured, a user only sees the data that the organization wants them to see. To define this access, write data access filtering rules (access grants) in XML and upload them as a new security file using the Domain designer. These rules are powerful and flexible, and can be based on a number of aspects, such as user roles or profile attributes.

The power of this solution is best presented as an example business case. This section describes a fictional company’s implementation of Domains in JasperReports Server—from both a business perspective and an implementation perspective. For details about the basics of Domains, refer to the JasperReports Server User Guide.

This chapter describes a number of tasks that only administrative users can perform.

Also note that it assumes a working knowledge of these third party technologies:
- Groovy is used when defining the filter expression in this example.
- SQL and database knowledge are required when working with the profile attribute table.

The examples in this chapter assume that you have a JasperReports Server version 3.7.1 or newer.
Version 3.7.1 (then called JasperServer) changed the value of applySecurityFilterInMemory and added a new DomEL function (testProfileAttribute). For more information, refer to section 7.4.3, “Alternative Filter Expressions using the testProfileAttribute Function,” on page 98. If you are running version 3.7.0, update to the latest version before testing the filter expressions in this chapter.

This chapter includes the following sections:
- Business Case
- Implementation
- Domain and Security Recommendations
- Reference Material
7.1 Business Case

CZS is an up-and-coming consumer electronics company with operations in the U.S. and Japan. CZS uses JasperReports Server to track sales data, such as sales revenue and operating cost.

The CZS Sales organization employs the following personnel:

- Rita is the regional sales manager in the Western U.S. She uses the Sales Numbers Domain to create reports that track sales trends in her region.
- Pete is a Sales Rep selling televisions in Northern California. He uses reports based on the same Domain to track his quarterly progress.
- Yasmin is a Sales Rep selling cell phones in Northern California. She uses reports based on the same Domain to track her quarterly progress.
- Alexi is the regional sales manager in Kansai, Japan. He uses reports based on the same Domain to track sales trends in his region.

CZS stores its data in a MySQL database. The data is exposed by the Sales Numbers Domain, which displays information about CZS’s consumer electronics sales across the world. It is filtered depending on each employee’s cities of operation and product. In addition, only managers can access cost information.

7.2 Implementation

This section describes how CZS addressed their business case using a Domain. It describes the specific steps CZS took to secure their data for users with certain profile attributes and roles.

7.2.1 Overview of CZS’s Process

<table>
<thead>
<tr>
<th>Steps</th>
<th>Described in Section …</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Defined a Domain that included the following fields from their JDBC data source: city, state, product department, sales amount, cost amount, and unit sales.</td>
<td>7.2.2, “Sales Numbers Domain,” on page 79</td>
</tr>
<tr>
<td>2. Identified and created access roles. CZS identified two roles: one for managers, and another for sales reps. Both are granted access to the Sales Numbers Domain.</td>
<td>7.2.3, “Roles,” on page 80</td>
</tr>
<tr>
<td>3. Created the users who represent their employees and assigned appropriate roles to each.</td>
<td>7.2.4, “Users,” on page 80</td>
</tr>
<tr>
<td>4. Identified and inserted the profile attributes that determine user’s access to data in the Domain. For this example, CZS identified these attributes: Cities and ProductDepartment.</td>
<td>7.2.5, “Security Files and Profile Attributes,” on page 81</td>
</tr>
<tr>
<td>5. Prepared to test their security implementation by enabling logging and creating an example report.</td>
<td>7.2.6, “Preparation,” on page 86</td>
</tr>
<tr>
<td>6. Iteratively created, uploaded, and tested an XML file that defines the access granted to users based on the attributes defined in step 4.</td>
<td>7.2.7, “Creating a Security File,” on page 87</td>
</tr>
<tr>
<td>7. Tested the Domain as various users.</td>
<td>7.2.8, “Testing and Results,” on page 90</td>
</tr>
</tbody>
</table>

After articulating their business case (as above), CZS implemented the Sales Numbers Domain; the following sections describe these steps in more detail.
7.2.2 Sales Numbers Domain

CZS is primarily interested in the volume and revenue of their sales, as well as their operational cost. These metrics are represented in the Domain as fields: unit sales, store sales, and store cost. The Domain also includes fields to establish context for the sales data, such as product department, city, and state. The following figures show the configuration of this Domain in the designer.

![Figure 7-1 Tables Tab in the Domain Designer](image1)

![Figure 7-2 Joins Tab in the Domain Designer](image2)
The XML representation of this Domain design is shown in section 7.4.1, “Domain Design in XML Format,” on page 92.

7.2.3 Roles

Domain security can reference a user’s roles to determine the access permissions to grant. CZS defined these roles:

- ROLE_SALES_MANAGER is assigned to Sales Managers.
- ROLE_SALES_REP is assigned to Sales Reps.

CZS grants each role access to view the Sales Numbers Domain. For details about creating roles and assigning privileges, refer to the JasperReports Server Administrator Guide. The following shows CZS’s ROLE_SALES_MANAGER:

7.2.4 Users

CZS created a user for each of their employees. It assigned roles based on each employee’s level of responsibility:

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexi, Rita</td>
<td>ROLE_SALES_MANAGER</td>
</tr>
<tr>
<td>Pete, Yasmin</td>
<td>ROLE_SALES_REP</td>
</tr>
</tbody>
</table>
For details about creating users, refer to the JasperReports Server Administrator Guide.

The following figure shows the configuration of Rita’s user account:

Notice Rita’s profile attributes listed below her roles. CZS implemented two of them (Cities and ProductDepartment) to use in defining row-level security for her. The next section explains this concept.

This example focuses on the Cities and ProductDepartment profile attributes. The other attributes depicted in the figure above are used in the data security example in the Jaspersoft OLAP Ultimate Guide.

### 7.2.5 Security Files and Profile Attributes

A Domain’s security file contains item and resource access grants that specify access grants based on certain aspects of a user, such as roles. Typically, access grants check a user’s roles and grant access to the columns and rows available to that role. An additional approach is to define profile attributes for each user that can limit the access grants within her roles. A profile attribute is a name-value pair defined at the user level that corresponds to some data in a Domain. This section illustrates both kinds of access grant.

#### 7.2.5.1 Column-level and Row-level Security

A Domain’s security file consists of access definitions of two types:

- **Row-level access**, which determines the rows in the data source that can be displayed to a specific user.

  Consider a table that includes values for the cities where products were sold. You could define a resource access grant that finds users to which a city has been defined as a profile attribute and grants access to only rows each user’s specific city. For example, the following resource grant gives access to users whose Cities profile attribute is San Francisco. The principle expression determines the users to whom the resource access grant applies, that is, users whose Cities profile attribute is San Francisco. The filter expression determines the rows to display, that is, those rows where the store_city field is San Francisco:

```xml
<resourceAccessGrant id="Jointree_1_row_access_grant_2">
  <principalExpression><![CDATA[authentication.getPrincipal().getAttributes().any
      {it.getAttrName() in ['Cities'] && it.getAttrValue() in ['San Francisco'] }]]></principalExpression>
  <filterExpression>store.store_city in ('San Francisco')</filterExpression>
</resourceAccessGrant>
```
Column-level access, which determines the columns in the data source that can be displayed to specific users.

Consider a table that includes employee contact and salary information; you could define item group access grants that check the user’s role and grant access to the salary field only if the user has the Human Resources role. For example, the following code sample grants ROLE_SALESREP users revokes the default access for that role then grants access to sales information only. The principle expression determines the users to whom the item group access grant applies, that is, users with the ROLE_SALESREP role. The item access grants determine the specific access of the users, that is, all role-specific access is revoked then access to the StoreSales and StoreCost item is granted:

```xml
<itemGroupAccessGrant id="Jointree_1_item_group_access_grant_2" access="granted">
    <principalExpression>authentication.getPrincipal().getRoles().any
        { it.getRoleName() in ['ROLE_SALESREP'] }</principalExpression>
    <itemAccessGrantList id="Jointree_1_grant2_item_group_items"
        defaultAccess="denied">
        <itemAccessGrants>
            <itemAccessGrant id="Jointree_1_grant2_items_grant1" itemId="StoreSales"
                access="granted" />
            <itemAccessGrant id="Jointree_1_grant2_items_grant2" itemId="UnitSales"
                access="granted" />
        </itemAccessGrants>
    </itemAccessGrantList>
</itemGroupAccessGrant>
</itemGroupAccessGrants>
```

### 7.2.5.2 Understanding Profile Attributes and Variable Substitution

A profile attribute is a name-value pair defined at the object level. In this example, it is set on the user and corresponds to a value in the data returned by the Domain. The server uses attributes like these to determine the access to grant to a user, based on her role. Variable substitution in the Domain security file can refer to these attributes at runtime.

For example, take Rita and Alexi. Both have the same role and the same access to the Sales Numbers analysis view, but CZS doesn’t want them to see the same data—Rita should see data about California and Alexi should see data about Osaka. Without profile attributes, this would only be possible if CZS’s access roles were defined along geographic lines. Instead, CZS defines profile attributes that indicate the users’ geographic affiliations within the same role: Rita is responsible for California and Alexi is responsible for Osaka.

This example shows a typical resource access grant that finds users with the Cities profile attribute, then grants access to ones whose attribute is San Francisco:

```xml
<resourceAccessGrant id="Jointree_1_row_access_grant_2">
    <principalExpression><![CDATA[authentication.getPrincipal().getAttributes().any
        { it.getAttributeName() in ['Cities'] && it.getAttributeValue() in ['San Francisco'] ]]]>
        </principalExpression>
    <filterExpression>store.store_city in ('San Francisco')</filterExpression>
</resourceAccessGrant>
```
However, using this approach, CZS would have to write a resource access grant for each city where they have stores:

```xml
<resourceAccessGrant id="Jointree_1_row_access_grant_2">
  <principalExpression><![CDATA[authentication.getPrincipal().getAttributes().any
    { it.getAttrName() in ['Cities'] && it.getAttrValue() in ['San Francisco'] }]]></principalExpression>
  <filterExpression>store.store_city in ('San Francisco')</filterExpression>
</resourceAccessGrant>

<resourceAccessGrant id="Jointree_1_row_access_grant_3">
  <principalExpression><![CDATA[authentication.getPrincipal().getAttributes().any
    { it.getAttrName() in ['Cities'] && it.getAttrValue() in ['Los Angeles'] }]]></principalExpression>
  <filterExpression>store.store_city in ('Los Angeles')</filterExpression>
</resourceAccessGrant>

...  
```

The complexity of such an implementation could require pages of XML code and would be cumbersome to maintain, since it would have to handle every possible profile attribute value separately. Therefore, to avoid this complexity, CZS can define profile attributes for their users that describe each user in terms of geography and product. The access grant definition can detect these profile attributes and compare their values to values in the data returned from the Domain. The principle expression determines each user’s profile attribute value and the filter expression compares that value to values in the Domain data.

If we were interested in securing the City field, a resource access grant that used variable substitution would be similar to the following:

```xml
<resourceAccessGrant id="Jointree_1_row_access_grant_4">
  <principalExpression><![CDATA[authentication.getPrincipal().getAttributes().any
    { it.getAttrName() in ['Cities'] }]]></principalExpression>
  <filterExpression>store.store_city in (groovy('authentication.getPrincipal()
    .getAttributes().find{ it.attrName == "Cities" }.attrValue'))
</filterExpression>
</resourceAccessGrant>
```

The principle expression is simple; it checks the Cities attribute of the logged-in user. The filter expression checks the user’s Cities profile attribute as well, but it compares this value with the values in the Domain’s store_city field. The Domain then returns all the rows that match the user’s Cities profile attribute.

Note the use of the Groovy expression language in the filter expression. Groovy is a dynamic language written for the Java Virtual Machine (JVM). It is used to write expressions and scripts for the Java platform. For more information, see [http://groovy.codehaus.org](http://groovy.codehaus.org). For another way to use profile attributes, see section 7.4.3, “Alternative Filter Expressions using the testProfileAttribute Function,” on page 98.

The following sections describe how to define profile attributes. The Domain security file that CZS created, including profile attributes, is shown in section 7.4.2, “Domain Security File,” on page 96.
7.2.5.3  CZS’s Profile Attributes

CZS determined that they need to describe their users in terms of product lines that they sell and the cities where they sell them. CZS also set attributes to secure data revealed in the analysis views (country, region, and state). Thus, each CZS user is assigned five profile attributes in addition to the users’ roles:

Table 7-1  Profile Attributes of All CZS Users

<table>
<thead>
<tr>
<th>User</th>
<th>Profile Attributes</th>
<th></th>
<th>Cities</th>
<th>Product/Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rita</td>
<td>USA</td>
<td>West</td>
<td>CA</td>
<td>San Francisco, Los Angeles, Sacramento Television, Wireless Devices</td>
</tr>
<tr>
<td>Pete</td>
<td>USA</td>
<td>West</td>
<td>CA</td>
<td>San Francisco             Television</td>
</tr>
<tr>
<td>Yasmin</td>
<td>USA</td>
<td>West</td>
<td>CA</td>
<td>San Francisco             Wireless Devices</td>
</tr>
<tr>
<td>Alexi</td>
<td>Japan</td>
<td>Kansai</td>
<td>Osaka</td>
<td>Osaka, Sakai              Wireless Devices</td>
</tr>
</tbody>
</table>

* This profile attribute is used in the analysis view data security example in the Jaspersoft OLAP Ultimate Guide.

The security file shown in section 7.4.2, “Domain Security File,” on page 96 refers to two of these profile attributes:

- The Cities profile attribute corresponds to the City field in the Geography item group in the Sales Numbers Domain.
- The Product/Department attribute corresponds to the Department field in the Product item group in the Sales Numbers Domain.

Each user’s attributes determine the data returned to him by the Domain, based on an access grant definition that refers to profile attributes. For example, Rita’s attribute value for Cities is San Francisco, Los Angeles, Sacramento while Pete’s is San Francisco. Thus, Pete sees less data than Rita does.

Profile attributes are defined in a table in the JasperReports Server’s private database. To update this table, you must use database tools, such as the PostgreSQL command line, or custom code.

At a high level, CZS took the following steps in populating and utilizing the profile attribute table:

<table>
<thead>
<tr>
<th>Steps</th>
<th>Described in Section…</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Retrieved each user ID by querying the JasperReports Server database. This information is necessary to identify the user when adding or updating a profile attribute.</td>
</tr>
<tr>
<td>2.</td>
<td>Inserted the required profile attributes into the table for each user. This defines the profile attributes referred to by variable substitution in the security file.</td>
</tr>
<tr>
<td>3.</td>
<td>Prepared for testing by enabling logging and creating a test report.</td>
</tr>
<tr>
<td>4.</td>
<td>Created an security file (in XML) that refers to these attributes.</td>
</tr>
<tr>
<td>5.</td>
<td>Used the access grant definition in a Domain.</td>
</tr>
</tbody>
</table>
7.2.5.4 Retrieving User IDs

In order to populate the profile attribute table, CZS needed to get the userID for each user who needs access to the Sales Numbers Domain. For example, to retrieve Rita’s userID, they ran the following SQL query:

```sql
postgres=# select * from jiuserrole where
  -> roleId=(select id from jirole where rolename='ROLE_SALES_MANAGER') and
  -> userId=(select id from jiuser where username='Rita');

+--------+--------+
| roleId | userId |
+--------+--------+
| 24     | 31     |
+--------+--------+
1 row in set (0.00 sec)
```

PostgreSQL returns Rita’s userId; it is 31. CZS then used this information to insert her profile attributes.

7.2.5.5 Inserting Attributes into the Profile Attribute Table in SQL

The profile attribute table (JIProfileAttribute) has five columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The primary key.</td>
</tr>
<tr>
<td>attrName</td>
<td>The name of the attribute. Use this name when referring to an attribute in an access grant definition.</td>
</tr>
<tr>
<td>attrValue</td>
<td>The value of this attribute for this user.</td>
</tr>
<tr>
<td>principalobjectclass</td>
<td>The class of the user object. The only valid values are:</td>
</tr>
<tr>
<td></td>
<td>com.jaspersoft.jasperserver.api.metadata.user.domain.impl.hibernate.RepoUser</td>
</tr>
<tr>
<td></td>
<td>com.jaspersoft.jasperserver.api.metadata.user.domain.impl.hibernate.RepoRole</td>
</tr>
<tr>
<td></td>
<td>Note that, while profile attributes can be defined for users and roles, repository objects (such as folders and resources) do not support them.</td>
</tr>
<tr>
<td>principalobjectid</td>
<td>The ID of the user or role whose attribute to set.</td>
</tr>
</tbody>
</table>

Depending on your repository database server and operating system, the table and column names may be case sensitive.

To insert a Cities attribute for Rita, CZS ran the following SQL statement:

```sql
INSERT INTO JIProfileAttribute
(attrName,attrValue,principalobjectclass,principalobjectid)
values('Cities','San Francisco,Los Angeles,Sacramento',"com.jaspersoft.jasperserver.api.metadata.user.domain.impl.hibernate.RepoUser",31);
```

This statement creates a Cities attribute of San Francisco, Los Angeles, Sacramento for the user with the userID of 31 (Rita).
To insert a ProductDepartment attribute for Rita, CZS ran the following SQL statement:

```
INSERT INTO JIProfileAttribute
(attrName, attrValue, principalobjectclass, principalobjectid)
values("ProductDepartment", "Televisions, Wireless Devices", "com.jaspersoft.jasperserver.api.metadata.user.domain.impl.hibernate.RepoUser", 31);
```

This statement creates a ProductDepartment attribute of Televisions, Wireless Devices for Rita.

To insert Rita’s other attributes (Country, Region, and State), CZS ran the following SQL statement:

```
INSERT INTO JIProfileAttribute
(attrName, attrValue, principalobjectclass, principalobjectid)
values("Country", "USA", "com.jaspersoft.jasperserver.api.metadata.user.domain.impl.hibernate.RepoUser", 31);

INSERT INTO JIProfileAttribute
(attrName, attrValue, principalobjectclass, principalobjectid)
values("Region", "West", "com.jaspersoft.jasperserver.api.metadata.user.domain.impl.hibernate.RepoUser", 31);

INSERT INTO JIProfileAttribute
(attrName, attrValue, principalobjectclass, principalobjectid)
values("State", "CA", "com.jaspersoft.jasperserver.api.metadata.user.domain.impl.hibernate.RepoUser", 31);
```

Rita now has the attributes shown in Table 7-1, “Profile Attributes of All CZS Users,” on page 84.

7.2.6 Preparation

Before creating a security file, CZS prepared for the implementation by:

- Enabling Logging
- Creating a Test Report

7.2.6.1 Enabling Logging

To assist in the iterative creation of their security file, CZS enabled more verbose logging to help troubleshoot problems with the Sales Numbers Domain and security file. Such logging features are disabled by default to minimize the size of logs. They should be enabled in test environments when defining security.

To enable Domain security logging:

1. Locate and open the log4j.properties file and scroll to the bottom of the file. This file is found in the WEB-INF folder; if you use Tomcat as your application server, the default path to this location is: `<js.install>/apache-tomcat/webapps/jasperserver-pro/WEB-INF`.

2. Add the following lines after the last line in the file:

```
log4j.logger.com.jaspersoft.commons.semantic.dsimpl.JdbcTableDataSet=DEBUG, stdout, fileout
log4j.logger.com.jaspersoft.commons.util.JSControlledJdbcQueryExecutor=DEBUG, stdout, fileout
```

3. Save the file.

Information about Domains and their security will now be written to the log and to the console.

The additional information written to the log can be very verbose, and your log files will grow more quickly with these properties enabled. You can manage your logs in the file system; they are found in the WEB-INF/logs folder under your JasperReports Server installation. For more information, refer to the log4j documentation, which is available at: http://logging.apache.org/log4j/docs/manual.html

Because these options are so verbose, Jaspersoft recommends that they only be used during debugging; these options should be disabled in production environments.

### 7.2.6.2 Creating a Test Report

CZS created an Ad Hoc crosstab based on the Sales Numbers Domain to assist in testing the security file as they created each access grant. The report displays store sales amount, store sales cost, and store units sold for all cities and departments.

![Figure 7-6 Administrator's View When Creating CZS Ad Hoc Crosstab](image)

Each user’s limited view of this report is shown in section 7.2.8, “Testing and Results,” on page 90.

### 7.2.7 Creating a Security File

This section shows the security file CZS created for the Sales Numbers Domain in mall portions. For the entire example XML file, refer to section 7.4.2, “Domain Security File,” on page 96.

**Note** the comments in the XML examples in this section; for example: <!-- Comment -->. It’s good practice to comment the access grants you define, and to format your XML neatly. Jaspersoft recommends that you use an XML editor when creating security files. See also section 7.3, “Domain and Security Recommendations,” on page 91.

#### 7.2.7.1 Creating Resource Access Grants for Cities

With an understanding of the discussion in section 7.2.5.2, “Understanding Profile Attributes and Variable Substitution,” on page 82, CZS created a security file that corresponded with the Domain design file they exported from the designer. They began with XML that defines a principal expression and filter expression that grant access to users based on their Cities profile attribute:

```xml
<resourceAccessGrant id="Jointree_1_row_access_grant_20">
  <principalExpression>
    <![CDATA[authentication.getPrincipal().getAttributes().any{ it.getAttrName() in ['Cities'] }]]>
  </principalExpression>
  <filterExpression>
    store.store_city in (groovy('authentication.getPrincipal().getAttributes().find {
      it.attrName == "Cities"}.attrValue'))
  </filterExpression>
</resourceAccessGrant>
```
The principal expression uses the Groovy language to check the user’s Cities attribute, and the filter expression users Groovy to limit the data returned to rows where the City column’s value matches the logged-in user’s profile attribute value.

In testing this resource access grant, CZS found that users with a value of San Francisco for their Cities attribute only saw data pertaining to San Francisco, as shown in Figure 7-9 on page 91. However, some of CZS’s users are responsible for multiple cities, and this access grant did not work properly for them, because its filter expression didn’t handle profile attributes that are actually collections: in our example case, comma-separated lists of values. For example, Rita is responsible for San Francisco, Sacramento, and Los Angeles; her profile attribute is set to Sacramento, San Francisco, Los Angeles.

To accommodate such cases, CZS wrote the following resource access grant; the Groovy in its filter expression disassemble the profile attribute values before comparing it to the data in the City field:

```xml
<resourceAccessGrant id="Jointree_1_row_access_grant_20">
  <principalExpression><![CDATA[authentication.getPrincipal().getAttributes().any
    { it.getAttrName() in ['Cities'] }]]></principalExpression>
  <filterExpression>store.store_city in (groovy('authentication.getPrincipal().getAttribute()
    .find{ it.attrName == "Cities" }.attrValue.split("",").collect
    {'''' + it + '''' }.join("",").replaceFirst("''","").replaceFirst("''$","")'))
</filterExpression>
</resourceAccessGrant>
```

Because CZS defined all their profile attributes in the same manner, they were able to use a similar resource access grant to determine data access for users based on their ProductDepartment profile attribute.

The resulting security file included these two resource access grants (see the complete file in section 7.4.2, “Domain Security File,” on page 96):

```xml
<resourceAccessGrant id="Jointree_1_row_access_grant_20">
  <principalExpression><![CDATA[authentication.getPrincipal().getAttributes().any
    { it.getAttrName() in ['Cities'] }]]></principalExpression>
  <filterExpression>store.store_city in (groovy('authentication.getPrincipal().getAttribute()
    .find{ it.attrName == "Cities" }.attrValue.split("",").collect
    {'''' + it + '''' }.join("",").replaceFirst("''","").replaceFirst("''$","")'))
</filterExpression>
</resourceAccessGrant>
```

```xml
<resourceAccessGrant id="Jointree_1_row_access_grant_30">
  <principalExpression><![CDATA[authentication.getPrincipal().getAttributes().any
    { it.getAttrName() in ['ProductDepartment'] }]]></principalExpression>
  <filterExpression>product_class.product_department in (groovy('authentication.getPrincipal().getAttribute()
    .find{ it.attrName == "ProductDepartment" }.attrValue.split("",").collect
    {'''' + it + '''' }.join("",").replaceFirst("''","").replaceFirst("''$","")'))
</filterExpression>
</resourceAccessGrant>
```
Since version 3.7.1, JasperReports Server includes a new DomEL function (`testProfileAttribute`) that can be used in the place of these Groovy expressions. For more information, refer to section 7.4.3, “Alternative Filter Expressions using the testProfileAttribute Function,” on page 98.

Access grant IDs must be unique within the scope of the security file.

You can define several similar resource access grants for each resource defined in your Domain. By default, the server assumes access grants should be chained together with a logical AND. You can force the server to use a logical OR by setting the `orMultipleExpressions` property to TRUE. For example:

```
<resourceAccessGrant id="Example_access_grant" orMultipleExpressions="true">
    The scope of this property is the entire resource access grant tag.
```

### 7.2.7.2 Creating Item Group Access Grants for Sales Data

To ensure that sales representatives don’t have access to cost information, CZS also added item group access grants; the first grants full access to managers and the administrator:

```xml
<itemGroupAccessGrant id="Jointree1_item_group_access_grant_MNG" access="granted">
    <principalExpression>authentication.getPrincipal().getRoles().any
        { it.getRoleName() in ['ROLE_ADMINISTRATOR','ROLE_SALES_MANAGER'] } ...
    </principalExpression>
</itemGroupAccessGrant>
```

CZS then added an item group access grant that grants limited access to sales reps; the following XML grants access to the Sales Amount and Sales Units fields while revoking access to the Sales Cost field:

```xml
<itemGroupAccessGrant id="Jointree1_item_group_access_grant_REP" access="granted">
    <principalExpression>authentication.getPrincipal().getRoles().any
        { it.getRoleName() in ['ROLE_SALES_REP'] } ...
    </principalExpression>
    <itemAccessGrantList id="Jointree1_grant2_item_group_items" defaultAccess="denied">
        <itemAccessGrants>
            <itemAccessGrant id="Jointree1_grant2_items_grant1" itemId="StoreSales" access="granted" />
            <itemAccessGrant id="Jointree1_grant2_items_grant2" itemId="UnitSales" access="granted" />
        </itemAccessGrants>
    </itemAccessGrantList>
</itemGroupAccessGrant>
```

### 7.2.7.3 Uploading the Security File

CZS uploaded and tested the security file each time they added a new access grant. You can upload the security file when you add or edit a Domain.
7.2.8 Testing and Results

Finally, CZS tested the Domain as various users by clicking the Login as User button on the Manage Users page.

To test the access granted to users on data in the Domain:
1. Log in as administrator (jasperadmin) if necessary.
2. Click Manage > Users.
3. In the list of user names, click the name of the user you want to test.
4. In the User page, click Log in as User.
   The selected user’s Home page appears.
5. Click View > Reports.
6. In the list of report, click the test report you created when defining your security file.
   The report appears.
7. Test the report to ensure that it only displays the data this user should see. Also test that you have not restricted data that the user should be able to view. The figures below show CZS’s results.
8. Click Logout to return to the administrator view.

When viewing the test report created from the Sales Numbers Domain:
- Rita can see all data pertaining to California and the three Californian cities where CZS has offices (Los Angeles, Sacramento, and San Francisco):

<table>
<thead>
<tr>
<th></th>
<th>Los Angeles</th>
<th>Sacramento</th>
<th>San Francisco</th>
<th>Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Televsions</td>
<td>$5,065.10</td>
<td>$4,023.88</td>
<td>$4,314.26</td>
<td>$14,403.24</td>
</tr>
<tr>
<td></td>
<td>$2,024.97</td>
<td>$1,952.52</td>
<td>$1,096.05</td>
<td>$5,073.54</td>
</tr>
<tr>
<td></td>
<td>2588</td>
<td>2422</td>
<td>2120</td>
<td>7122</td>
</tr>
<tr>
<td>Wireless Devices</td>
<td>$39,305.74</td>
<td>$39,187.46</td>
<td>$36,699.97</td>
<td>$115,193.17</td>
</tr>
<tr>
<td></td>
<td>$15,677.91</td>
<td>$12,308.10</td>
<td>$14,712.26</td>
<td>$42,198.27</td>
</tr>
<tr>
<td></td>
<td>13558</td>
<td>18294</td>
<td>16553</td>
<td>58205</td>
</tr>
<tr>
<td>Totals</td>
<td>$44,579.84</td>
<td>$44,011.34</td>
<td>$41,014.23</td>
<td>$129,606.41</td>
</tr>
<tr>
<td></td>
<td>$17,692.58</td>
<td>$17,542.74</td>
<td>$16,407.31</td>
<td>$51,642.62</td>
</tr>
<tr>
<td></td>
<td>20929</td>
<td>20716</td>
<td>19113</td>
<td>60758</td>
</tr>
</tbody>
</table>

Figure 7-8 Rita’s view of the CZS Test Report
- Pete can only see Television data about San Francisco; the Cost measure displays zeroes because he is denied access to that field:

<table>
<thead>
<tr>
<th>Sales Data by City</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Francisco</td>
</tr>
<tr>
<td>Televisions</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Totals</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Figure 7-9 Pete’s view of the CZS Test Report

- Yasmin can only see Wireless Devices data about San Francisco; the Cost measure displays zeroes because she is denied access to that field:

<table>
<thead>
<tr>
<th>Sales Data by City</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Francisco</td>
</tr>
<tr>
<td>Wireless Devices</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Totals</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Figure 7-10 Yasmin’s view of the CZS Test Report

- Alexi can see all data pertaining to Osaka and the two Japanese cities where CZS has offices (Osaka and Sakai):

<table>
<thead>
<tr>
<th>Sales Data by City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Osaka</td>
</tr>
<tr>
<td>Wireless Devices</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Totals</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Figure 7-11 Alexi’s view of the CZS Test Report

### 7.3 Domain and Security Recommendations

When defining a Domain and its security, keep these recommendations in mind:

- A Domain should cover a large subject area and include data with multiple uses. Define joins to create data islands that each contain related information; the data islands themselves can contain completely unrelated data. For example, you could include both human resources and sales data in a single Domain; different users would see only the information relevant to their job responsibilities. For an example of this type of Domain, refer to the SuperMart example that can be installed with JasperReports Server.

- When defining a Domain, don’t create too many item groups, and avoid very deep structures with many levels. Such complexity makes the Domain harder to use.

- Logging can help you troubleshoot any problems you encounter while implementing Domain security. For more information, refer to section 7.2.6.1, “Enabling Logging,” on page 86.
• Refer to http://groovy.codehaus.org for information on the Groovy expressions that Domain security files support. Note that, while the server does validates Groovy expressions, the validation is very light weight: it doesn’t detect all improperly formed expressions.

• If the names of tables and fields in your data source change, you can edit the Domain design XML file so that the resource names match the new names in the database. Then, upload the new version of the file; your reports that rely on the Domain will work properly without being updated individually. If you have defined a security file for this Domain, you must also edit the resource names in the security file, as well.

• Start with the simplest item or resource grant, and when that works, expand upon it. Start simple and iterate until you have the full set of access grants needed. Follow good troubleshooting practices, such as only changing a single aspect of the security file before testing the results of the change.

• Use an XML editor (such as XMLBuddy) to create your security file. While the server validates the schema against its own XML definition, a typical XML editor can identify such issues as tags that aren’t properly closed. For example, open the security file with Internet Explorer; if it returns errors, use them to identify and correct your XML.

• Once your Domain is created, create several Domain Topics that focus on specific aspects of the Domain or specific data your end-users will want to review regularly. To do so, click Create > Ad Hoc Report, select your Domain, and use the Data, Filters, and Display pages to customize the contents and the way it is displayed, then use the Topics page to save the new Domain Topic.

For tips on improving the performance of Domains and reports that rely on them, refer to section 3.3.5, “Scalability,” on page 42.

7.4 Reference Material

7.4.1 Domain Design in XML Format

The CZS-sales-Domain.xml file defines a Domain that returns data from the sales_fact_2006 table stored in a MySQL database. It includes the three fields and CZS is interested in displaying, as well as data the corresponds to the profile attributes described in the security file.

```xml
  <itemGroups>
    <itemGroup description="Product family and department" descriptionId="" id="Products" label="Products" labelId="" resourceId="JoinTree_1">
      <items>
        <item description="Family" descriptionId="" id="Family" label="Family" labelId="" resourceId="JoinTree_1.product_class.product_family" />
        <item description="Department" descriptionId="" id="Department" label="Department" labelId="" resourceId="JoinTree_1.product_class.product_department" />
      </items>
    </itemGroup>
    <itemGroup description="Geography" descriptionId="" id="Geography" label="Geography" labelId="" resourceId="JoinTree_1">
  </itemGroups>
</schema>
```
<items>
  <item description="City" descriptionId="" id="City" label="City" labelId=""
    resourceId="JoinTree_1.store.store_city" />
  <item description="Country" descriptionId="" id="Country" label="Country"
    labelId="" resourceId="JoinTree_1.store.store_country" />
  <item description="State" descriptionId="" id="State" label="State"
    labelId="" resourceId="JoinTree_1.store.store_state" />
</items>
</itemGroup>
</itemGroup>
</resources>
<jdbcTable datasourceId="czsdata" id="product" tableName="product">
  <fieldList>
    <field id="brand_name" type="java.lang.String" />
    <field id="gross_weight" type="java.lang.Double" />
    <field id="net_weight" type="java.lang.Double" />
    <field id="product_class_id" type="java.lang.Integer" />
    <field id="product_id" type="java.lang.Integer" />
    <field id="product_name" type="java.lang.String" />
    <field id="recyclable_package" type="java.lang.Boolean" />
    <field id="shelf_depth" type="java.lang.Double" />
    <field id="shelf_height" type="java.lang.Double" />
    <field id="shelf_width" type="java.lang.Double" />
    <field id="SKU" type="java.lang.Long" />
    <field id="SRP" type="java.math.BigDecimal" />
    <field id="units_per_case" type="java.lang.Short" />
  </fieldList>
</jdbcTable>
<jdbcTable datasourceId="czsdata" id="product_class" tableName="product_class">
  <fieldList>
    <field id="product_category" type="java.lang.String" />
    <field id="product_class_id" type="java.lang.Integer" />
    <field id="product_department" type="java.lang.String" />
    <field id="product_family" type="java.lang.String" />
    <field id="product_subcategory" type="java.lang.String" />
  </fieldList>
</jdbcTable>
```xml
<jdbcTable datasourceId="czsdata" id="product" tableName="product">
  <fieldList>
    <field id="brand_name" type="java.lang.String" />
    <field id="gross_weight" type="java.lang.Double" />
    <field id="net_weight" type="java.lang.Double" />
    <field id="product_class_id" type="java.lang.Integer" />
    <field id="product_id" type="java.lang.Integer" />
    <field id="product_name" type="java.lang.String" />
    <field id="recyclable_package" type="java.lang.Boolean" />
    <field id="shelf_depth" type="java.lang.Double" />
    <field id="shelf_height" type="java.lang.Double" />
    <field id="shelf_width" type="java.lang.Double" />
    <field id="SKU" type="java.lang.Long" />
    <field id="SRP" type="java.math.BigDecimal" />
    <field id="units_per_case" type="java.lang.Short" />
  </fieldList>
</jdbcTable>

<jdbcTable datasourceId="czsdata" id="sales_fact_2006" tableName="sales_fact_2006">
  <fieldList>
    <field id="customer_id" type="java.lang.Integer" />
    <field id="product_id" type="java.lang.Integer" />
    <field id="promotion_id" type="java.lang.Integer" />
    <field id="store_cost" type="java.math.BigDecimal" />
    <field id="store_id" type="java.lang.Integer" />
    <field id="store_sales" type="java.math.BigDecimal" />
    <field id="time_id" type="java.lang.Integer" />
    <field id="unit_sales" type="java.math.BigDecimal" />
  </fieldList>
</jdbcTable>

<jdbcTable datasourceId="czsdata" id="store" tableName="store">
  <fieldList>
    <field id="coffee_bar" type="java.lang.Boolean" />
    <field id="first_opened_date" type="java.sql.Timestamp" />
    <field id="last_remodel_date" type="java.sql.Timestamp" />
    <field id="region_id" type="java.lang.Integer" />
    <field id="store_city" type="java.lang.String" />
    <field id="store_country" type="java.lang.String" />
    <field id="store_fax" type="java.lang.String" />
    <field id="store_id" type="java.lang.Integer" />
    <field id="store_manager" type="java.lang.String" />
    <field id="store_name" type="java.lang.String" />
    <field id="store_number" type="java.lang.Integer" />
    <field id="store_phone" type="java.lang.String" />
    <field id="store_postal_code" type="java.lang.String" />
    <field id="store_sqft" type="java.lang.Integer" />
    <field id="store_state" type="java.lang.String" />
    <field id="store_street_address" type="java.lang.String" />
    <field id="store_type" type="java.lang.String" />
  </fieldList>
</jdbcTable>
```
Securing Data in a Domain

```xml
<jdbCTable datasourceId="czsdata" id="JoinTree_1" tableName="product">
  <fieldList>
    <field id="product_class.product_category" type="java.lang.String" />
    <field id="product_class.product_class_id" type="java.lang.Integer" />
    <field id="product_class.product_department" type="java.lang.String" />
    <field id="product_class.product_family" type="java.lang.String" />
    <field id="product_class.product_subcategory" type="java.lang.String" />
    <field id="sales_fact_2006.customer_id" type="java.lang.Integer" />
    <field id="sales_fact_2006.product_id" type="java.lang.Integer" />
    <field id="sales_fact_2006.promotion_id" type="java.lang.Integer" />
    <field id="sales_fact_2006.store_cost" type="java.math.BigDecimal" />
    <field id="sales_fact_2006.store_id" type="java.lang.Integer" />
    <field id="sales_fact_2006.store_sales" type="java.math.BigDecimal" />
    <field id="sales_fact_2006.time_id" type="java.lang.Integer" />
    <field id="sales_fact_2006.unit_sales" type="java.math.BigDecimal" />
    <field id="product.brand_name" type="java.lang.String" />
    <field id="product.gross_weight" type="java.lang.Double" />
    <field id="product.net_weight" type="java.lang.Double" />
    <field id="product.product_class_id" type="java.lang.Integer" />
    <field id="product.product_id" type="java.lang.Integer" />
    <field id="product.product_name" type="java.lang.String" />
    <field id="product.recyclable_package" type="java.lang.Boolean" />
    <field id="product.shelf_depth" type="java.lang.Double" />
    <field id="product.shelf_height" type="java.lang.Double" />
    <field id="product.shelf_width" type="java.lang.Double" />
    <field id="product.SKU" type="java.lang.Long" />
    <field id="product.SRP" type="java.math.BigDecimal" />
    <field id="product.units_per_case" type="java.lang.Short" />
    <field id="store.coffee_bar" type="java.lang.Boolean" />
    <field id="store.first_opened_date" type="java.sql.Timestamp" />
    <field id="store.grocery_sqft" type="java.lang.Integer" />
    <field id="store.last_remodel_date" type="java.sql.Timestamp" />
    <field id="store.meat_sqft" type="java.lang.Integer" />
    <field id="store.region_id" type="java.lang.Integer" />
    <field id="store.store_city" type="java.lang.String" />
    <field id="store.store_country" type="java.lang.String" />
    <field id="store.store_fax" type="java.lang.String" />
    <field id="store.store_id" type="java.lang.Integer" />
    <field id="store.store_manager" type="java.lang.String" />
    <field id="store.store_name" type="java.lang.String" />
    <field id="store.store_number" type="java.lang.Integer" />
    <field id="store.store_phone" type="java.lang.String" />
    <field id="store.store_postal_code" type="java.lang.String" />
    <field id="store.store_sqft" type="java.lang.Integer" />
  </fieldList>
</jdbCTable>
```
### 7.4.2 Domain Security File

The CZS-sales-security.xml file is based on the CZS-sales-domain.xml Domain design file, and defines access for users with Cities and ProductDepartment profile attributes.
Securing Data in a Domain

<!-- Row level for Product Dept -->
<resourceAccessGrant id="Jointree_1_row_access_grant_30">
  <principalExpression><![CDATA[authentication.getPrincipal().getAttributes().
  any{ it.getAttrName() in ['ProductDepartment'] }]]></principalExpression>
  <filterExpression>product_class.product_department in
  (groovy('authentication.getPrincipal().getAttributes().find{ it.attrName ==
  "ProductDepartment" }.attrValue.split("","`).collect{"''" + it +
  "''"}.join("","`).replaceFirst("''","").replaceFirst("\$","")'))
  </filterExpression>
</resourceAccessGrant>
</resourceAccessGrantList>

<!-- Column level security -->
<!-- What access do roles/users have to the fields in an item group? -->
<!-- Column level for managers and admin -->
<!-- Column level for sales reps -->
</securityDefinition>
7.4.3 Alternative Filter Expressions using the testProfileAttribute Function

CZS’s filter expressions rely on complex Groovy expressions to identify the cities and product departments to display for each user. The DomEL function testProfileAttribute can replace these expressions with simpler ones.

The testProfileAttribute function is not available in versions of JasperReports Server earlier than 3.7.1.

7.4.3.1 The testProfileAttribute Function

The testProfileAttribute function takes two parameters:

testProfileAttribute(table_ID.field_name, 'profileAttribute')

where:

- table_ID.field_name is the table name and field name of the field you’re comparing to a profile attribute.
- profileAttribute is the name of the user profile attribute.

With this method, you can create much simpler filter expressions to check user profile attributes. For example, CZS could use the following city and product department access grants instead of those shown in section 7.4.2, “Domain Security File,” on page 96:

```xml
<resourceAccessGrantList id="JoinTree_1_List" label="ListLabel" resourceId="JoinTree_1">
    <resourceAccessGrants>
        <!-- Row level for Cities -->
        <resourceAccessGrant id="Jointree_1_row_access_grant_20">
            <principalExpression><![CDATA[authentication.principal.attributes.any
                {it.attrName in ['Cities']}]]></principalExpression>
            <filterExpression>testProfileAttribute(store.store_city, 'Cities')</filterExpression>
        </resourceAccessGrant>
        <!-- Row level for Product Dept -->
        <resourceAccessGrant id="Jointree_1_row_access_grant_30">
            <principalExpression><![CDATA[authentication.getPrincipal().getAttributes().
                any(it.getAttrName() in ['ProductDepartment'])]]></principalExpression>
            <filterExpression>testProfileAttribute(product_class.product_department,
                'ProductDepartment')</filterExpression>
        </resourceAccessGrant>
    </resourceAccessGrants>
</resourceAccessGrantList>
```

The first filter expression tests the store.store_city field against the Cities profile attribute; the second tests the product_class.product_department field against the ProductDepartment profile attribute. They would replace the corresponding Groovy expressions in section 7.4.2, “Domain Security File,” on page 96.
7.4.3.2 Editing testProfileAttribute for the Delimiter in the SQL Profile Attribute Table

testProfileAttribute is defined in the file applicationContext-semanticLayer.xml:

```xml
<entry key="testProfileAttribute">
  <value>
    def testAttrName = args[1].value
    def testField = sqlArgs[0]
    def attrVal = authentication.principal.attributes.find{it.attrName == testAttrName}? .attrValue
    testField + " in (" + attrVal?.split(",").collect{"'" + it.trim() + "'"}.join("'" + ")" + ")"
  </value>
</entry>
```

The default delimiter in JIProfileAttribute, the SQL profile attribute table, is a comma, as shown in the tables in section 7.2.5.5, “Inserting Attributes into the Profile Attribute Table in SQL,” on page 85.

If you use a different delimiter in SQL, you must edit testProfileAttribute so that it recognizes your delimiter. To do this, edit the following line in the applicationContext-semanticLayer.xml file:

```java
    testField + " in (" + attrVal?.split(",").collect("" + it.trim() + ").join("","")" + ")"
```

Change the comma to your delimiter; for example:

```java
    testField + " in (" + attrVal?.split("your_delimiter").collect("" + it.trim() + ").join("" + ")"
```
CHAPTER 8 APPLICATION SECURITY

Application security protects the JasperReports Server web application from unwarranted changes, malicious intrusions, and malware. This chapter explains measures you can take in order to provide such protection on the Tomcat server. These measures do not offer 100% protection—no measures can guarantee that except possibly pulling the plug—but they do create an acceptable level of protection, and they are a foundation upon which more thorough measures can be built.

What follows is not a complete tutorial on securing your web application. To do so would take volumes. Instead, we have written instructions on four basic components of a secure Tomcat environment; we also added related code in the default installation. These additions demonstrate the basic procedures you should follow, but you may have to adapt the procedures to your installation.

The tutorials assume the following system configuration:

- JasperReports Server 4.1
- JDK 1.6.0.18
- Apache Tomcat 6.0.26
- Web App Deployed Name: jasperserver

For more information on the components of a secure environment, see the OpenSSL web site and the Java documentation.

The chapter includes the following sections:

- Using SSL in the Web Server
- Disabling Unused HTTP Verbs
- Setting the Secure Flag on Cookies
- Setting httpOnly for Cookies
- Using a Protection Domain Infrastructure
- Final Note

8.1 Using SSL in the Web Server

Secure Sockets Layer (SSL) is a widely-used protocol for secure network communications. It encrypts network connections at the Transport Layer and is used in conjunction with HTTPS, the secure version of the HTTP protocol.

This section shows how to install SSL on Tomcat 6.0 and to configure JasperReports Server to use only SSL in Tomcat:

1. Setting Up an SSL Certificate
2. Enabling SSL in the Web Server
3. Configuring JasperReports Server to Use Only SSL
8.1.1 Setting Up an SSL Certificate

To use SSL, you need a valid certificate in the Tomcat keystore. In the Java Virtual Machine (JVM), certificates and private keys are saved in a keystore. This is the repository for your keys and certificates. By default, it is implemented as a password-protected file (public keys and certificates are stored elsewhere).

If you already have a suitable certificate, you can import it into the keystore, using the import switch on the JVM keytool utility. If you don’t have a certificate, you can use the keytool utility to generate a self-signed certificate (one signed by your own certificate authority). Self-signed certificates are acceptable in most cases, although certificates issued by certificate authorities are even more secure. And they do not require your users to respond to a security warning every time they login, as self-signed certificates do.

The following command is an example of how to import a certificate. In this case, it is a self-signed certificate imported into a PKCS12 keystore using OpenSSL:

```bash
openssl pkcs12 \-export \-in mycert.crt \-inkey mykey.key \-out mycert.p12
\-name tomcat \-CAfile myCA.crt \-caname root \-chain
```

Next in this example, you create key.bin, the keystore file, in the Tomcat home folder. Use one of these commands:

<table>
<thead>
<tr>
<th>OS</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>%JAVA_HOME%\bin\keytool -genkey -alias tomcat -keyalg RSA -keystore %CATALINA_HOME%\conf\key.bin</td>
</tr>
<tr>
<td>Unix</td>
<td>$JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA -keystore $CATALINA_HOME/conf/key.bin</td>
</tr>
</tbody>
</table>

The basic install requires certain data. With the above commands, you are prompted for the data:

- Enter two passwords twice. The default for both is “changeit”. If you use the default, be sure to set better, stronger passwords later.
- Specify information about your organization, including your first and last name, your organization unit, and organization. The normal response for first and last name is the domain of your server, such as jasperserver.mycompany.com. This identifies the organization the certificate is issued to. For organization unit, enter your department or similar-sized unit; for organization, enter the company or corporation. These identify the organization the certificate is issued by.
- Keytool has numerous switches. For more information about it, see the Java documentation.

8.1.2 Enabling SSL in the Web Server

Once the certificate and key are saved in the Tomcat keystore, you need to configure your secure socket in the SCALALINA_BASE/conf/server.xml file, where SCALALINA_BASE represents the base directory for the Tomcat instance. For your convenience, sample `<Connector>` elements for two common SSL connectors (blocking and non-blocking) are included in the default server.xml file that is installed with Tomcat. They are similar to the code below, with the connector elements commented out, as shown.

```xml
<!-- Define a SSL HTTP/1.1 Connector on port 8443
    This connector uses the JSSE configuration, when using APR, the
    connector should be using the OpenSSL style configuration
    described in the APR documentation -->

<!-- Define a SSL Connector on port 8443
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
    maxThreads="150" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS" />
-->
To implement a connector, you need to remove the comment tags around its code. Then you can customize the specified options as necessary. For detailed information about the common options, consult the [Tomcat 6.0 SSL Configuration HOW-TO](#). For detailed information about all possible options, consult the [Server Configuration Reference](#).

The default protocol is HTTP 1.1; the default port is 8443. The port is the TCP/IP port number on which Tomcat listens for secure connections. You can change it to any port number (such as the default port for HTTPS communications, which is 443). However, note that on many operating systems, special setup that is outside the scope of this document is necessary if you run Tomcat on port numbers lower than 1024.

### 8.1.3 Configuring JasperReports Server to Use Only SSL

At this point, the JasperReports Server web application runs on either protocol (HTTP and HTTPS). You can test the protocols in your web browser:

- **HTTP:** http://localhost:8080/jasperserver[-pro]/
- **HTTPS:** https://localhost:<SSL port>/jasperserver[-pro]/

The next step, then, is to configure the web application to enforce SSL as the only protocol allowed. Otherwise, requests coming through HTTP are still serviced.

Edit the file `<js-webapp>/WEB-INF/web.xml`. Near the end of the file, make the following changes inside the first `<security-constraint>` tag:

- Comment out the line `<transport-guarantee>NONE</transport-guarantee>`.
- Uncomment the line `<transport-guarantee>CONFIDENTIAL</transport-guarantee>`.

Your final code should be like the following:

```xml
<security-constraint>
    <web-resource-collection>
        <web-resource-name>JasperServerWebApp</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <!-- SSL not enforced -->
        <!-- <transport-guarantee>NONE</transport-guarantee> -->
        <!-- SSL enforced -->
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

In the code, the term CONFIDENTIAL forces the server to accept only SSL connections through HTTPS. And because of the URL pattern `/*`, all web services must also use HTTPS. If you need to turn off SSL mode, you can set the transport guarantee back to NONE or delete the entire `<security-constraint>` tag.

### 8.2 Disabling Unused HTTP Verbs

It is prudent to disable all unused HTTP verbs so that they cannot be used by intruders.

In the default JasperReports Server installation, the following HTTP verbs are not used, but they are allowed. However, to facilitate your disabling the verbs, they are listed in a single block of code in `<js-webapp>/WEB-INF/web.xml`. As in the code immediately above, the URL pattern `/*` applies the security constraint to all access to the server, including web service requests.

The list is commented out by default because it has not been exhaustively tested with all system configurations and platforms.
After uncommenting the security constraint, your final code should be like the following:

```xml
<!-- This constraint disables the listed HTTP methods, which are not used by JS -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>RestrictedMethods</web-resource-name>
    <url-pattern>/</url-pattern>
    <http-method>HEAD</http-method>
    <http-method>CONNECT</http-method>
    <http-method>COPY</http-method>
    <http-method>LOCK</http-method>
    <http-method>MKCOL</http-method>
    <http-method>OPTIONS</http-method>
    <http-method>PATCH</http-method>
    <http-method>PROPPATCH</http-method>
    <http-method>SEARCH</http-method>
    <http-method>TRACE</http-method>
    <http-method>UNLOCK</http-method>
  </web-resource-collection>
</security-constraint>
```

### 8.3 Setting the Secure Flag on Cookies

To require that JasperReports Server cookies be sent only via secure connections, call `setSecure(true)` on the [Servlet API](https://docs.oracle.com/javaee/8/api/javax/servlet/http/Cookie.html) class. This only acts on the cookies; providing a secure connection is up to the client application, although it is currently generally taken to mean providing an HTTPS connection.

At present, JasperReports Server uses only two cookies: time zone and locale (the session cookie is handled by the application server). We do not set the secure flag on them because we don't want to force you to use secure connections. If you want all cookies to be secure, not just the JasperReports Server ones, you have to secure those cookies throughout the rest of the system. For example, Spring Security and Tomcat also create cookies.

### 8.4 Setting httpOnly for Cookies

The application server that hosts JasperReports Server handles the session cookie. To prevent malicious scripts on a client from accessing the session cookie, and thus the user connection, you should set the application server to use httpOnly cookies. This tells the browser that only the server may access the cookie, not scripts running on the client.

Since Apache Tomcat 6.0.19 or 5.5.28, you can enable `useHttpOnly` in the file `<tomcat>/conf/context.xml`:

```xml
<Context useHttpOnly="true">
  ...
</Context>
```

### 8.5 Using a Protection Domain Infrastructure

Legitimate code can be used to introduce harmful measures into the web application. For instance, calls for disk access and calls to `System.exit` can be hidden in classpaths when running a report. An effective measure against such intrusions is to
implement a protection domain. In Tomcat, in order to implement a protection domain you have to enable the Tomcat Security Manager then edit its parameters according to the requirements of your server environment.

The `ProtectionDomain` class encloses a group of classes whose instances have the same permissions, public keys, and URI. A given class can belong to one and only one ProtectionDomain. For more information on `ProtectionDomain`, see the Java documentation.

### 8.5.1 Enabling the JVM Security Manager

Enabling the Security Manager restricts permissions at the application server level. By default, all permissions at that level are disallowed, so legitimate permissions must be added specifically. You must add permissions for JasperReports Server. Doing so does not interfere with server operations because JasperReports Server security restrictions occur on other levels.

Add the enabling code for the Security Manager in the file `<apache-tomcat>/conf/catalina.policy` file. ProtectionDomains can be enabled, as defined in `<js-webapp>/WEB-INF/applicationContext.xml`, `reportsProtectionDomainProvider` bean.

To enable the Security Manager and give JasperReports Server full permissions there, add the following code fragment at the end of catalina.policy:

```java
// These permissions apply to the JasperReports Server application
grant codeBase "file:${catalina.home}/webapps/jasperserver[-pro]/-" {
    permission java.security.AllPermission;
}
```

After enabling the manager, you should add the security parameter to your Tomcat startup command. For example:

```shell
<apache-tomcat>\bin\startup -security
```

If you did not add the permissions properly, you will receive errors like the following:

```
Feb 9, 2010 12:34:05 PM org.apache.catalina.core.StandardContext listenerStart
SEVERE: Exception sending context initialized event to listener instance of class
org.springframework.web.context.ContextLoaderListener
java.security.AccessControlException: access denied (java.lang.RuntimePermission
accessDeclaredMembers)
    at java.security.AccessControlContext.checkPermission(Unknown Source)
    at java.security.AccessController.checkPermission(Unknown Source)
    at java.lang.SecurityManager.checkPermission(Unknown Source)
    at java.lang.SecurityManager.checkMemberAccess(Unknown Source)
    at java.lang.Class.checkMemberAccess(Unknown Source)
    at java.lang.Class.getDeclaredMethods(Unknown Source)
```

### 8.5.2 Restoring Disallowed Permissions

The file `<js-webapp>/WEB-INF/applicationContext.xml` defines the permissions that are allowed for `java.security.Class`. You might have to use the file to add permissions that enabling the Security Manager has disallowed. On the application level, only specified permissions are granted now, so any application-level permissions you were using have been disallowed. You must write code that restores them.
To help you restore necessary permissions, the following commented sample code is provided in the applicationContext.xml file. For instance, to add permission for read/write access to the /temp folder, you would uncomment the code for the bean class java.io.FilePermission:

```xml
<bean id="reportsProtectionDomainProvider" class="com.jaspersoft.jasperserver.api.engine.jasperreports.util.PermissionsListProtectionDomainProvider">
    <property name="permissions">
        <list>
            <!-- no permissions by default -->
            <!-- sample permission: read and write to temp folder -->
            <!-- <bean class="java.io.FilePermission">
            <!-- <constructor-arg value="${java.io.tmpdir}${file.separator}*/"/>
            <!-- <constructor-arg value="read,write"/>
            <!-- </bean>-->\n            <!-- all permissions can be granted if desired -->
            <!-- <bean class="java.security.AllPermission"/>-->\n        </list>
    </property>
</bean>
```

8.5.3 Additional Customizations for Previous Versions of Tomcat

For Tomcat versions 5.5.28 and earlier and 6.0.20 and earlier, you also need to add permissions for Groovy scripts in the catalina.policy file and in the protection domain for reports.

In `<apache-tomcat>/conf/catalina.policy`, permissions to read the JasperReports Server classpath needs to be granted to Groovy scripts (which use /groovy/script as their codebase). This change applies to calculated fields in Ad Hoc, which use Groovy to evaluate expressions:

```xml
grant codeBase "file:/groovy/script" {
    permission java.io.FilePermission "${catalina.home}${file.separator}webapps
        ${file.separator}jasperserver-pro${file.separator}WEB-INF${file.separator}classes-${", "read";

    permission java.io.FilePermission "${catalina.home}${file.separator}webapps
        ${file.separator}jasperserver-pro${file.separator}WEB-INF${file.separator}lib
        ${file.separator}*, "read";
}
```

In `<js-webapp>/WEB-INF/applicationContext.xml`, the same permissions need to be added to reportsProtectionDomainProvider. This change grants access to reports that use the Groovy language, plus reports that need to load additional classes from the JasperReports Server web application:

```xml
<bean id="reportsProtectionDomainProvider" class="com.jaspersoft.jasperserver.api.engine.jasperreports.util.PermissionsListProtectionDomainProvider">
    <property name="permissions">
        <bean class="java.io.FilePermission">
            <constructor-arg value="${catalina.home}${file.separator}webapps
                ${file.separator}jasperserver-pro${file.separator}WEB-INF${file.separator}classes-${"/"}/">
            <constructor-arg value="read"/>
        </bean>
    </property>
</bean>
```
Also, for a Tomcat bug found in 5.5.26, 5.5.27 and 6.0.16, and fixed in 5.5.28 and 6.0.18, the following configuration change is required for JasperReports Server to start properly.

In <apache-tomcat>/conf/catalina.policy, find the section that starts with:

```xml
<bean class="java.io.FilePermission">
  <constructor-arg value="file:${catalina.home}/bin/tomcat-juli.jar"/>
</bean>
</property>
</bean>
```

Add the following line in that section:

```xml
permission java.io.FilePermission "${catalina.base}${file.separator}webapps
 ${file.separator}jasperserver-pro${file.separator}WEB-INF${file.separator}lib${file.separator}*", "read";
```

### 8.6 Final Note

The example instructions in this chapter apply only to the indicated versions of Tomcat. They have not been tested on any other server. We offer them as a useful model for implementing security on your server, regardless of its type.

Additional measures that you might take include disabling unnecessary applications and resources, encrypting usernames and passwords, closing unused ports, and avoiding memory leaks.

Jaspersoft is committed to application security; we remain alert to advances in the art.
CHAPTER 9  JASPERREPORTS SERVER PLUG-IN FOR iREPORT

iReport Designer is the most widely used advanced visual designer for JasperReports. The JasperReports Server plug-in for iReport transforms iReport into a client with which you can access the server’s repository, navigate it, manage its content, and run report units. You can create, edit and manage folders, report units, JRXML sources, and all the related resources, such as images, fonts, and input controls. The additional Domain plug-in helps you create reports that return data from a Domain.

The JasperReports Server plug-in uses the web services API to access the repository. For more information, refer to the web services documentation provided with JasperReports Server.

This chapter describes:
- Getting Started
- Folders and Resources
- Data Sources
- File Resources
- Report Units

9.1  Getting Started

9.1.1  Installation and Startup

Depending on the distribution you installed from, you start up iReport differently:
- If you installed iReport as part of a JasperReports Server installation, start iReport by selecting Start > All Program > JasperReports Server x.x > Start iReport (where x.x is your version of JasperReports Server).
- If you installed iReport from the stand-alone iReport Professional distribution, iReport by selecting Start > All Program > Jaspersoft > iReport-Professional-x.x.x > iReport-Professional-x.x.x (where x.x.x is your version of iReport).

To open the JasperReports Server Plug-in, click Window > JasperReports Server Repository. iReport displays the plug-in as a tree view in a panel on the left, with the Report Inspector.
The Domain plug-in is also available; note that it is only compatible with commercial editions of JasperReports Server.

9.1.2 Server Connection Configuration

By default, the plug-in doesn’t display any servers; they must be added manually. To add one, click . The server configuration window appears.

Specify the correct value for each field and click Save to add the server to the list.
Note that the **Organization** field is only supported in commercial editions of JasperReports Server since version 3.5 (then called JasperServer), and is only required if your instance hosts multiple organizations. For more information on organizations, refer to the *JasperReports Server Administrator Guide*.

Expand the server icon to display and explore its repository.

### 9.1.3 Common Connection Errors

<table>
<thead>
<tr>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(401) Bad credential</td>
<td>Invalid user name or password provided.</td>
</tr>
<tr>
<td>The AXIS engine could not find a target service to invoke!</td>
<td>The web services URL provided is incorrect.</td>
</tr>
<tr>
<td>java.net.ConnectException: Connection refused: connect</td>
<td>The server is down, or the host or port specified in the web services URL is incorrect.</td>
</tr>
</tbody>
</table>

### 9.2 Folders and Resources

When you view the repository in iReport, you can navigate its folders by double-clicking them.

Right-click an object in the docking panel to open the context menu for the object. The menu provides access to all the available actions that you can take, such as viewing object properties, refreshing a folder’s content, and deleting objects.

Working in the repository through the JasperReports Server Plug-in is much like working directly in the repository.

While resources can be located anywhere in the repository, Jaspersoft recommends that you organize the repository in such a way that you can store similar resources together. Think carefully about the organization of your repository; such planning can facilitate maintenance and authorization by grouping similar objects together. For more information, refer to section 6.2.1, “Repository Design for Access Control,” on page 64.

Right-click a resource in the Repository Navigator panel and select **Properties** from the context menu to view its details. A resource must have a name and a label (that is, a display name); you can also specify a description. These properties appear on the **General** tab. A resource’s other properties are determined by its type; for example, a data source resource’s properties also includes whether it is a JNDI, JDBC, or Bean type data source. Such properties appear on other tabs in the Properties window.
Figure 9-4 shows the general properties of a query resource. For instructions on creating a query, see the *JasperReports Server Administrator Guide*.

![General Tab in the Properties Dialog](image)

The *Save* button is only active when:
- You have made changes since opening it; and
- All required fields are populated.

Since the ID of a resource is part of its URI (also called a resource path; for example /datasources/ProfileData Source, or /reports/samples/AllAccounts), the **ID** field cannot contain spaces or special characters. However, note that these characters are supported in the **Label** field because the label is only used for display.

9.3 Data Sources

Users generally define data sources in iReport when working with reports in iReport. For these reports to run in JasperReports Server, the server must have similar data sources. If you aren’t using the plug-in, you must test the reports in iReport, then login to the server to upload them and test them again on the server. With the plug-in, you can run the report on the server from within the plug-in. Since the server and iReport can have different access to the database (or other data source), Jaspersoft recommends running the report unit on the server rather than running it locally. This yields more reliable test results. In some cases, such as when the data source is a Domain, the report must be tested by way of the server, either by running it through the plug-in or by uploading it to the repository and then running it.

If needed, you can create data sources on the server through the plug-in. You can also view and edit the definition of a data source on the server if you have the required permission, usually if you’ve connected as an administrator (jasperadmin).

Creating a data source through the plug-in is very similar to working on the server:
1. In the Repository Navigator, right-click the folder where you want the data source and select **Add > Data Source**.
2. Enter an ID and a name for the data source on the general tab.
3. On the Data Source Details tab, select the type of data source you want to add:
   - JDBC Data Source
   - JNDI Data Source
   - Bean Data Source
   The information you need to enter for each is the same as when working on the server. For details about data sources, see the *JasperReports Server Administrator Guide*.
4. For JDBC data sources, you also have the option of importing a data source definition from those you have defined in iReport. Click **Import from iReport**, select the data source you want to import from, and enter the database password if necessary.
5. Click **Save** when done to create the data source on the server.
The plug-in also supports importing a data source from the server. It is often convenient to have the same data source within iReport for previewing a report during development. Use the context menu for data source objects in the Repository Navigator to import a data source definition from the server.

After importing a JasperReports Server data source, it appears as the default data source in iReport’s list of connections/data sources.

When you import a JDBC connection, iReport stores its password in clear text in a configuration file (located in the user’s home directory; for example: C:\Documents and Settings\<user>\Application Data\ ireportpro\iReport version>\config\Preferences\com\jaspersoft\ireport\jasperserver.properties.

Ensure that this folder is properly safeguarded against unauthorized access.

In addition to the data source, reports can also rely on the Domains. Domains provide a metadata layer on top of other data sources defined in the repository. iReport can generate reports from Domains using the Domain plug-in. See the JasperReports Server User Guide for details.

Remember that you shouldn’t use a locally-defined data source when testing reports to run on JasperReports Server. JRXML files created for JasperReports Server can contain special URLs (using the repo: syntax) that refer to repository resources, such as images and subreports. Because iReport can’t resolve the special URLs, the report fails when run locally.
9.4 File Resources

When you create a new file resource, the plug-in prompts for a resource ID, a display name, an optional description, and a file to upload. You can also upload file resources automatically when you add them to an existing report in the repository. For more information, see section 9.5.4, “Adding Images, Style Templates, and Subreports,” on page 117.

To edit an existing resource, right-click it and select Properties from the context menu. For fonts and images, the properties window includes a preview.

The Resource tab in the properties window enables you to share files between the server and your local computer.
- To replace the file on the server with a local instance, select the Replace the resource with this file check box on the context menu, and use the Browse button to locate and select the new file.
- To download a file that is stored on the server, click the Export button on the Properties window’s Resource tab.

Click Save to save your changes.

JasperReports Server won’t let you upload class files directly. To upload new classes, you must first package them, either by zipping them up or using the Java Development Kit’s JAR tool to create a JAR file.

Once you upload the JAR file, you must link it to the report unit that relies on it. You can also define the resource directly in the report unit.

9.5 Report Units

iReport displays a report unit as a folder; think of the report unit as a container for all the pieces you need to create the output you want. At a minimum, a report unit needs a JRXML (called the main JRXML) and a data source that fills it.

A report’s resources are either locally defined or externally defined:
- A locally defined resource resides in the report unit itself, and can only be used by this one report.
- An externally defined resource resides elsewhere in repository; the report unit references the resource using a pointer to its location.

A report unit can include every resource it requires; it could just as easily reference each resource externally, if each were defined in the repository. The decision of whether to define local or external resources revolves largely around the nature of the resource itself:
- If the resource is unique to the report, define it locally. For example, most JRXML source files and JAR files that contain report-specific scriptlets.
- If the resource would be useful in a number of reports, upload it to the repository separately and reference it externally. For example, you might externally define resources such as data sources, commonly-used JAR files, your logo, or fonts.
A report unit contains the main JRXML, the data source (only visible if it is locally defined), and two subfolders, Input Controls and Resources, which contain the input controls used to prompt users for the information and resources the report requires (both locally and externally defined).

### 9.5.1 Creating a JasperReport on the Server

This section walks you through the creation of a JasperReport on the server through the plug-in. We’ll use the iReport wizard to create a JRXML file that relies on a JDBC connection that’s already defined in the repository. While more complex reports might require you to start a new JRXML file from scratch, we’ll step through the wizard.

1. Be sure that the data source you plan to use on the server is available as an active connection in iReport. We’ll use the Jserver Jdbc data source connection in the /Data Sources repository folder; this is only created if you chose to install the sample data during installation. This connection points to the JasperReports Server SugarCRM sample database.

2. To create a new report:
   1. In the Repository Navigator, right-click the Jserver Jdbc data source data source in the repository and select Import JDBC Connection Definition.
   2. Select File > New and scroll down among the templates to select Simple Blue.
   3. Click Launch Report Wizard to start the new report with this template.
   4. Specify a name and location, and click Next.
   5. Select the data source you imported in step 1.
   6. Enter the following in the Query (SQL) field:

   ```sql
   SELECT accounts.id AS accounts_id,
        accounts.name AS accounts_name,
        accounts.industry AS accounts_industry,
        accounts.email1 AS accounts_email1
   FROM
        accounts accounts
   ```

   7. Click Next.
   8. Select all the fields that the query returns, and click Next.
   9. Click Next to skip the definition of grouping for the report.
   10. At the end of the wizard, click Finish.

   The report appears in the design area. You may need to click on the Report Inspector to make it visible, because it shares the same panel as the Repository Navigator. For more information about the wizard, refer to the iReport Ultimate Guide.

11. Enter a report title such as My Accounts and a description such as Accounts by Industry. Save the report and preview it to see the values it contains.

   The report is ready to be uploaded to the server.
To upload the new report to JasperReports Server:

1. In iReport’s Repository Navigator, right-click the folder where you want to place the report, and click **Add > JasperServer Report**.

2. When you are prompted for a name and a label (display name) for the new report unit, enter them and click **Next**.

3. When you are prompted to select the main JRXML file, select the report you just created in the wizard by clicking **Get source from current opened report**.

4. Click **Next**.

5. When prompted to locate a data source, select From the repository, and from the drop-down, choose the data source you exported in step 1 of the previous procedure (/datasources/JServerJdbcDS), and click **Finish**.

The new report unit appears in the folder of the repository tree. You can expand the report unit, though this simple report only has a main JRXML component.
7. You can also login to JasperReports Server, and click View > Reports to locate the report and run it.

This example used a very simple report: it has no extra resources (such as images or fonts), nor does it include a subreport, input controls, or offer drill down functionality. These more advanced subjects, are discussed in the following sections:

- 9.5.4, “Adding Images, Style Templates, and Subreports,” on page 117
- 9.5.5, “Using Input Controls,” on page 119
- 9.5.6, “Using Drill Down Functionality,” on page 121

### 9.5.2 Editing the JRXML File

If you have sufficient permissions, you can edit any JRXML file in the repository by double-clicking it in the JasperReports Server plug-in tree view. The file is downloaded to a temporary file found in the `<user home>/Local Settings/Temp/jstmp` folder (in Windows).

To save the file in the repository, right-click the JRXML file in the JasperReports Server plug-in and select Replace with current JRXML.

### 9.5.3 Running a Report

When you are using the plug-in, Jaspersoft recommends that you avoid testing your changes in the local iReport instance (by using the Preview button). It relies on an iReport data source, which might not be available in the repository. Instead, test your reports directly on your instance of JasperReports Server by selecting the report unit that you want to test, then click Run JasperServer Report in the plug-in tool bar. Select the report unit rather than its main JRXML.

Testing your reports on your instance of JasperReports Server has several advantages:

- The report is executed in the environment where your users will access it, and it will use the exact data source defined on the server, even if the data isn’t directly accessible by iReport.
- It allows you to test reports that refer to externally-defined resources using the special `repo:` syntax; the server finds the referenced resource, and no error is returned (as is usually the case when testing such reports in iReport without the plug-in), since the resource can be found by the server engine. For more information on the `repo:` syntax, see section 9.5.4, “Adding Images, Style Templates, and Subreports,” on page 117.
- When testing reports that rely on input controls, iReport prompts you with them before running the report on the server.

### 9.5.4 Adding Images, Style Templates, and Subreports

Reports in JasperReports Server can reference several kinds of external objects:

- Images. File resources used in reports.
- Style templates. Files of type JRTX that define a set of styles for use in multiple reports.
- Subreports. Any JRXML files in your report other than the main JRXML file. Subreports reference the same data source as the main JRXML but not necessarily the same resources.

You can point to these files using a URI, classpath, or repository reference (for example, `repo:/images/JRLogo`). For reports that are uploaded to the server’s repository, Jaspersoft recommends that you also store the images, style templates, and
subreports in the repository and that you use the \texttt{repo:} syntax to reference them in the reports. For more information about referencing resources, see the \textit{JasperReports Server Administrator Guide}.

The JasperReports Server plug-in for iReport makes it easy to manage the resources in the repository through iReport.

There are two ways to reference resources using iReport:

- Add the resources to the repository ahead of time, as described in section 9.4, “File Resources,” on page 114, then drag them from the iReport Repository Navigator to the design area. The resources appears in your report layout and iReport automatically creates the expressions that reference them. For subreports, JasperReports Server compiles the JRXML sources and locates the subreports’ generated \\texttt{.jasper} files.

  Adding the references ahead of time makes them available for use in other reports.

- If you do not add the resources before you upload the JRXML file to the repository, the plug-in detects all the resources that the JRXML file references and reminds you to upload them. It also recommends rewriting the expressions that reference them in \texttt{repo:} syntax, as shown in Figure 9-13.

  When you upload the resources in this way, they are saved in the report unit and are unavailable for other reports.
9.5.5 Using Input Controls

A report can include input controls that determine the data in the report and its appearance. For example, you might include an input control to allow users to select the name of a customer from a drop-down list; when the report is run, it only includes data pertaining to that customer. Such input controls are mapped by their names to parameters in the report itself, which must match. JasperReports Server provides a default interface for input controls but you can specify your own JSP to show, as described in section 11.5.1, “Customizing the Input Controls Form,” on page 168.

JasperReports Server provides a rich set of highly configurable input controls. You can define as many as needed. They can be mandatory or optional, and can be made read only, so the user understands the report’s parameters but cannot change them.

When defining input controls, you must specify a type of control that is suitable for the parameter’s data. For example, it doesn’t make sense to define a Boolean type input control for a Date parameter.

The types of input control that JasperReports Server supports can be grouped in three basic categories:

- **Boolean.** The simplest type. Generally represented as a check box, these input controls return a java.lang.Boolean object to the report engine in response to a user’s selection.

- **Single value.** Require you to specify a datatype resource. Generally represented as a free-form text box; the user’s entry is validated against the datatype resource you specify. Note that you can manage datatype resources the same way you manage other resources; you can define them locally or externally to the input control.

- **List-based.** Refer to a resource in the repository. These controls offer the user a set of options, from which he can select one (single-select) or more (multi-select). If only a single selection is allowed, the input control is generally represented as a drop-down or a set of radio buttons. Multi-select input controls are generally represented as a scrollable list or a set of check boxes. The available options come from a repository resource that is either a static list or a query (SQL statement).
9.5.5.1 Datatypes

Datatypes are used with single-value input controls.

The only required information for a datatype is the datatype (Text, Number, Date, or Date/Time). Its optional information includes:

- Pattern. Similar to a format mask: the user’s entry must match it.
- Minimum and maximum. Set thresholds for valid values. While minimum and maximum are generally used with numbers, you can also use them with strings. The report engine uses the Java Comparable interface to check these constraints. The check boxes determine whether the constraint is inclusive. For instance, for a percent field, you might specify a minimum of 0 and a maximum of 100. If you do not want to accept 0 percent, you would select the minimum check box. If you want to accept 100 percent, you would clear the maximum check box.

<table>
<thead>
<tr>
<th>Java Class Type</th>
<th>Datatype That Is Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>Text</td>
</tr>
<tr>
<td>java.math.BigDecimal</td>
<td>Number</td>
</tr>
<tr>
<td>java.util.Date</td>
<td>Date and Date/Time</td>
</tr>
</tbody>
</table>

9.5.5.2 List-Based

List-based input controls are populated using lists and queries.

- The static list is simply a set of labels and values that map to data returned by the report.

- A query is identified by a query language and some text. By default, a query in iReport runs against the same data source as the report. However, you can specify a different data source than the report uses. This feature provides you a great deal of flexibility; the input values can come from a separate source.
Once the query is defined, you must indicate which columns in the query result to use as the item value, and which columns to display as the label when the input control is rendered.

A static list of values always returns an item value with the class type java.lang.String. However, when the list comes from a query, the type is unpredictable; you’ll need to know the type returned by the query before creating the input control.

9.5.5.3 Multi-Select Input Controls

For multi-select input controls, you can use the IN and NOT IN clause functions:

$X{ IN, COLUMN, PARAMETER NAME}

For example:

```sql
SELECT * FROM ORDER WHERE $X{IN, CITY, myCities}
```  

where:

- CITY is the column name.
- myCities is the parameter.

This clause function is exploded in the following:

```sql
SELECT * FROM ORDER WHERE CITY IN ( 'city1', 'city2', ... )
```

The parameter myCities must be a non-null Java Collection.

9.5.6 Using Drill Down Functionality

One of the most interesting features of JasperReports Server is the ability to connect reports to create drill-down and drill-up functionality. In this case, your users can click elements in the report to navigate to other reports that are generated dynamically. You can configure this functionality for any report element that supports links, such as text fields, images, and charts.

To create such a link using iReport, simply drag the report unit icon from the JasperReports Server plug-in repository tree view, and drop it on the report element in the design area. The element’s Properties window appears, displaying the Link tab.
The plug-in automatically sets the Hyperlink Type to the correct value (ReportExecution), and it adds a parameter (_report) that points to the report to generate when the user clicks the element. The plug-in also creates a list of the target report’s parameters. For each of these input parameters, you should specify an expression that indicates the correct input control in the repository (similar to what is done in JasperReports with subreports, crosstabs, or subdataset runs).

Report connections only work when reports are run in the JasperReports Server web interface (in a browser).

In addition to the report input parameters, the following optional parameters can be manually added along with value expressions:

- **_output** – Specifies the format in which the target report should open. Values for this parameter are keys for the report exporters configured in JasperReports Server. By default, the server recognizes the following output types: pdf for PDF, xls for Excel, rtf for RTF, csv for CSV and swf for the server’s Flash report viewer. When this parameter is not specified, the default format is HTML displayed in the default report viewer.

- **_page** – Specifies the initial page that should be displayed when opening the target report. By default the first page is displayed. This parameter is only effective when HTML output is used for the drill-down.

- **_anchor** – Specifies the name of an anchor from the target report at which the report should open. This parameter is only effective when _page is not specified and when the output format is HTML.

### 9.5.7 iReport and the Ad Hoc Editor

You can use the JasperReports Server plug-in for iReport to browse your server’s repository to locate and edit reports created with the Ad Hoc Editor. For example, you can add or remove columns.

Note that, because the Ad Hoc Editor relies on a different data sorting and aggregation mechanism, some changes you make in iReport will be lost when you open the report in the Ad Hoc Editor. In particular, changes to grouping and sorting made in iReport are lost when the report is opened in the Ad Hoc Editor. You can, in most cases, add and remove columns without ill effect.

Changes that don’t affect data are typically supported, but may be ignored. Modifying visual elements, such as changing the color scheme, adding images, changing fonts, or moving elements is typically supported. These changes appear when the report is run, but if the report is opened in the Ad Hoc Editor, such changes are lost when the report is saved.
CHAPTER 10 JASPERREPORTS SERVER APIs

One of the main goals of JasperReports Server is to expose a set of reusable application programming interfaces (APIs) that are easy to understand, extend, and customize. This facilitates adapting JasperReports Server to the unique requirements of different deployments.

This chapter describes the main entry points into the JasperReports Server APIs, the functionality they provide, and how they are leveraged in the product. JasperReports Server supports the following APIs:

- Java APIs
- Ad Hoc Launcher Java API
- Repository HTTP API
- Ad Hoc Editor HTTP API
- Dashboard Designer HTTP API
- Web services APIs (see the JasperReports Server Web Services Guide)

10.1 Java APIs

This section describes some of the important Java interfaces available in JasperReports Server, including commercial editions and Jaspersoft OLAP modules.

10.1.1 The Public JasperReports Server API

A subset of the Java classes and interfaces in JasperReports Server has been designated as the public JasperReports Server API. These classes are marked with an @JasperServerAPI annotation, as demonstrated in the example below.

```java
package com.jaspersoft.jasperserver.api.metadata.jasperreports.domain;

import com.jaspersoft.jasperserver.api.JasperServerAPI;
import com.jaspersoft.jasperserver.api.metadata.common.domain.DataSource;

@JasperServerAPI
public interface ReportDataSource extends DataSource {
    ...
}
The JavaDoc for the JasperReports Server API classes can be downloaded from the Support Portal (for the commercial editions) or from JasperForge (for the community project).

Classes included in the public API are more likely to be stable from release to release, so Java developers should use them in preference to other classes which are not part of the API. Developers should note, however, that the public API is a small subset of all JasperReports Server classes; it doesn’t provide all of the functionality that developers may need, in which case you must create and use other classes. The public API will continue to be expanded to provide APIs for new features, and future releases may fill some current gaps. Also note that this section does not cover all JasperReports Server API classes.

These JasperReports Server Java APIs are a contract between JasperReports Server (including Jaspersoft OLAP) and other applications and services that are exposed as Java interfaces. If the APIs change in the future, the changes will be gradual.

10.1.1.1 Accessing API Implementations Using the Spring Framework

Many of the implementations of the API interfaces are singletons, usually services, which are instantiated by the Spring Framework. The Spring bean configuration files control how these singletons are created and configured, so it is important to understand the files before writing Java code that will run in JasperReports Server using the API.

The following is a brief overview of Spring 2.5 and is not meant to cover all the possible ways to configure Spring. For more information on Spring, refer to its reference documentation at http://static.springsource.org/spring/docs/2.5.x/reference/index.html.

The Spring configuration files use XML to define Java singleton instances, called beans. In the JasperReports Server web application, these files are located under the WEB-INF directory. Their file names begin with applicationContext and end in .xml, for example, the file WEB-INF/applicationContext-adhoc.xml contains Ad Hoc-related beans:

- Each instance of a singleton is defined by a `<bean>` element.
- Its type is specified by the `class` attribute.
- Its reference ID is specified by the `id` attribute.
- Properties of the instance are set with the `<property>` element:
  - The `name` attribute corresponds to a Java property that follows JavaBean conventions. For example, a property with name `abc` should have a getter method `getAbc()` and a setter method `setAbc()`.
  - Using the `value` attribute, properties can be set with a constant value.
  - Using the `ref` attribute, properties can be set with a reference to another bean.

Below is part of a definition from a sample custom data source implementation. It demonstrates all of the conventions above; the original file is samples/customDataSource/webapp/WEB-INF/applicationContext-hibernateDS.xml in the JasperReports Server distribution.

```xml
<bean id="hibernateDataSource"
  class="com.jaspersoft.jasperserver.api.engine.jasperreports.util.
  CustomDataSourceDefinition">
  <!-- this property is always the same; it registers the custom ds -->
  <property name="factory" ref="customDataSourceServiceFactory"/>
  <!-- name used in message catalog and elsewhere -->
  <property name="name" value="hibernateDataSource"/>
</bean>
```

To add your own instances to the server, you first need information about the specific enhancement that you want to implement. This determines the Java implementations that are required. A good example is creating a custom data source, which is documented in the JasperReports Server Administrator Guide for the commercial editions or the JasperReports Server Community Project User Guide. Samples of custom data sources are located in the JasperReports Server distribution under samples/customDataSource.

Once you have a Java class that you want instantiated along with JasperReports Server, you can deploy it by modifying the webapp directory as follows:

- Add your compiled Java class files to WEB-INF/classes, or create a JAR and add it to WEB-INF/lib.
• Create a new Spring bean file under WEB-INF, using the naming convention described above.
• Add a `<bean>` element for each object instance you want to create.
• For each property you want to set, you must have a public setter and getter.
• For each API implementation you want to access:
  • Add a setter and getter to your implementation whose types match the Java type of the API.
  • Find out the ID of the API instance you want; some IDs are listed in the table below.
  • Add a `<property>` element to your bean with a `ref` attribute whose value is the ID of the API instance.

As an example of a reference to another bean, please refer to the factory property in the Spring file excerpt above. The CustomDataSourceDefinition instance uses the `factory` property to refer to a singleton implementation of CustomReportDataSourceServiceFactory, which has a bean ID of custom-DataSourceServiceFactory:

• The CustomDataSourceDefinition implementation defines a factory JavaBean property by implementing the following setter and getter:
  • `public void setFactory(CustomReportDataSourceServiceFactory factory)`.
  • `public CustomReportDataSourceServiceFactory getFactory()`.
• The `<bean>` element contains a `<property>` element with name set to `factory` and `ref` set to `customDataSourceServiceFactory`.

The following table contains the APIs described in the rest of this section, along with their corresponding bean IDs and descriptions of their functions.

<table>
<thead>
<tr>
<th>API</th>
<th>Bean ID</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepositoryService</td>
<td>repositoryService</td>
<td>Search, retrieve, and modify persistent objects in the repository.</td>
</tr>
<tr>
<td>EngineService</td>
<td>engineService</td>
<td>Run reports and handle report metadata.</td>
</tr>
<tr>
<td>ReportDataSourceService and</td>
<td>n/a</td>
<td>Implement data sources used for running reports and other purposes.</td>
</tr>
<tr>
<td>ReportDataSourceServiceFactory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OlapConnectionService</td>
<td>olapConnectionService</td>
<td>Manage OLAP-specific repository and model runtime.</td>
</tr>
<tr>
<td>OlapManagementService</td>
<td>olapManagementService</td>
<td>Manage OLAP server run time.</td>
</tr>
<tr>
<td>ObjectPermissionService</td>
<td>objectPermissionService</td>
<td>Search, retrieve, and modify metadata repository object permissions.</td>
</tr>
<tr>
<td>UserAuthorityService</td>
<td>userAuthorityService</td>
<td>Manage internal users and roles.</td>
</tr>
<tr>
<td>ReportSchedulingService</td>
<td>reportSchedulingService</td>
<td>Manage report schedules.</td>
</tr>
</tbody>
</table>

10.1.2 Repository API

It’s easy to populate the repository (using metadata or output content) and subsequently exploit it. This functionality relies on a limited set of interfaces and classes.

10.1.2.1 Object Model and Service

The `com.jaspersoft.jasperserver.api.metadata.common.service.RepositoryService` interface is central to accessing the metadata repository. It exposes various methods to store, lookup, and retrieve content from the repository. The repository is hierarchical; it is very similar to a file system. However, instead of files, the repository stores resources in a metadata representation of a tree structure.

All resources must have a name, label (display name), description, and type; the names must be unique within a folder. Resources reference their parent folder and are uniquely identified by their absolute URI. This URI consists of the full folder path within the repository, suffixed with the resource name. For example, the URI for the ContentFiles folder created when you run the installer is `/ContentFiles`. 
From the object model perspective, all resources are instances of the `com.jaspersoft.jasperserver.api.metadata.common.domain.Resource` interface and represent various entities that constitute the metadata (such as reports, data sources, datatypes, analysis views, and fonts), or generated content (such as generated report output in PDF or XLS format).

Even folders are special types of resources. The `com.jaspersoft.jasperserver.api.metadata.common.domain.Folder` interface, which represents folders, directly inherits from `com.jaspersoft.jasperserver.api.metadata.common.domain.Resource`.

All interfaces that represent the main object model of the repository have convenience class implementations in the `com.jaspersoft.jasperserver.api.metadata.common.domain.client` package; they have the `Impl` suffix added to their corresponding interface name. These implementations are shown in the examples that follow when we instantiate folders and resources.

### 10.1.2.2 Working with Folders

With the most minimal setup (manual WAR file deployment), the repository includes a single folder by default; it serves as the repository’s root directory. In this setup, Jaspersoft recommends that you create sensible folders (within root) to hold all your repository resources.

If you use one of the installers, the root directory includes a number of standard folders. You can use them as-is or create your own structure, depending on your needs. The following code creates a folder in /root:

```java
import com.jaspersoft.jasperserver.api.common.domain.ExecutionContext;
import com.jaspersoft.jasperserver.api.metadata.common.service.RepositoryService;
import com.jaspersoft.jaspersoft.jasperserver.api.metadata.common.domain.client.FolderImpl;
...
ExecutionContext context = ...; // gets the instance of the ExecutionContext
// interface, or receives it as a parameter in the // current method
RepositoryService repositoryService = ...; // gets the instance of the // RepositoryService interface
...
Folder myFolder = new FolderImpl();
myFolder.setName("examples");
myFolder.setLabel("Examples");
myFolder.setDescription("Folder containing various resources to use as examples.");
repositoryService.saveFolder(context, myFolder);
```

Note that the code doesn’t specify a parent for the new folder. In this case, the server assumes that the new resource should reside in /root.

The following code creates a new subfolder in the /examples folder created immediately above:

```java
Folder imagesFolder = new FolderImpl();
imagesFolder.setName("images");
imagesFolder.setLabel("Images");
imagesFolder.setDescription("Folder containing image resources to use in the examples.");
imagesFolder.setParentFolder("/examples");
repositoryService.saveFolder(context, imagesFolder);
```

The following code gets the /examples/images subfolder and changes its description:

```java
Folder imagesFolder = repositoryService.getFolder(context, "/examples/images");
imagesFolder.setDescription("Example Images Folder");
repositoryService.saveFolder(context, imagesFolder);
```
The existence of a folder can be verified using the `folderExists` method, as shown here:

```java
repositoryService.folderExists(context, "\examples\images");
```

Removing a folder from the repository is also easy. It needs only one method call that identifies the folder by its absolute URI. For example:

```java
repositoryService.deleteFolder(context, "\examples\images");
```

Just as the server’s web interface lets you explore the repository and manage it, the API includes methods (exposed by the `RepositoryService`) that allow you to get a list of subfolders and manage a given folder’s content. The API includes one method that gets the list of subfolders and another method that gets the list of other types of child resources. For example, the following code returns a list of folders:

```java
List folders = repositoryService.getSubFolders(context, "\reports");
if (folders.isEmpty()) {
    System.out.println("No folders found under /reports");
} else {
    System.out.println(folders.size() + " folder(s) found under /reports");
    for (Iterator it = folders.iterator(); it.hasNext();) {
        Folder folder = (Folder) it.next();
        System.out.println("Subfolder: " + folder.getName());
    }
}
```

### 10.1.2.3 Repository Resources

Adding a new resource in the repository differs from adding a folder, despite the fact that folders are themselves resources. Unlike folders, which are simple in structure and behavior, other types of resources might need to be initialized in a special way, and the initialization logic would probably reside in a service. As a result, we created a unique API for managing repository resources; you can use it regardless of type and internal structure.

New resource instances are created by making a special request to the `RepositoryService`, not by direct instantiation. This can be seen in the following example, where we create a new image resource and put it in the repository by loading it from an image file on disk:

```java
FileResource img = (FileResource) repositoryService.newResource(context, FileResource.class);
img.setFileType(FileResource.TYPE_IMAGE);
img.setName("logo.gif");
img.setLabel("Logo Image");
img.setDescription("Example Logo Image");
img.readData(new FileInputStream("C:\Temp\MyImages\logo.gif"));
img.setParentFolder("\examples\images");
repositoryService.saveResource(context, img);
```
To retrieve a resource from the repository, you could call the following method on the `RepositoryService` instance:

```java
// retrieve a data source resource from the repository
Resource resource = repositoryService.getResource(context,
        "datasources/mydatasource");
if (resource == null) {
    throw new RuntimeException("Resource not found at /datasources/mydatasource"); }
if (resource instanceof JdbcReportDataSource)
    JdbcReportDataSource datasource = (JdbcReportDataSource) resource;
    System.out.println("JDBC data source URI: " + datasource.getConnectionUrl());
else if (resource instanceof JndiJdbcReportDataSource)
    JndiJdbcReportDataSource datasource = (JndiJdbcReportDataSource) resource;
    System.out.println("JNDI data source name: " + datasource.getJndiName());
else {
    throw new RuntimeException("Was expecting /datasources/mydatasource to be a datasource");
}
```

You can save or persist a resource in the repository by calling the following (as already seen above where we created the image resource) on the `RepositoryService` instance:

```java
public void saveResource(ExecutionContext context, Resource resource);
```

Removing a resource from the repository is done by calling `repositoryService.deleteResource`:

```java
try {
    repositoryService.deleteResource(context, "/reports/myreport");
    System.out.println("Resource /reports/myreport deleted");
} catch (Exception e) {
    System.err.println("Not able to delete resource /reports/myreport");
    e.printStackTrace();
}
```

### 10.1.2.4 Content Files

Content resources are specially-created resource objects that hold binary data. The data is usually the result of using some of the BI tools available in JasperReports Server, such as the report-generating services, which produce PDF and XLS output. The output can be stored in the repository for later use, especially if the reports were generated in the background as scheduled jobs.

Creating a content resource and adding it to the repository is similar to what we've seen in the previous section, where we created an image resource:

```java
ContentResource pdfResource = new ContentResourceImpl();
pdfResource.setFileType(ContentResource.TYPE_PDF);
pdfResource.setName("report.pdf");
pdfResource.setLabel("PDF Report");
pdfResource.setDescription("Example PDF File");
pdfResource.readData(new FileInputStream("C:\Temp\MyReports\report.pdf"));
pdfResource.setParentFolder("/examples");
repositoryService.saveResource(context, pdfResource);
```
Retrieving the binary data of a content resource from the repository is achieved by using the `getContentResourceData` method of the `RepositoryService`, as follows:

```java
FileResourceData fileResourceData = repositoryService.getContentResourceData(context, "/examples/report.pdf");
byte[] pdfContentBytes = fileResourceData.getData();
```

### 10.1.2.5 Repository Search

You get the list of child resources within a given folder by using filter criteria. The server expects an instance of the `com.jaspersoft.jasperserver.api.metadata.view.domain.FilterCriteria` class as a parameter in the method call; the list of returned resources matches certain the selected filter conditions.

The only required condition for a `FilterCriteria` instance is that the returned resources’ parent folder must match a given folder. For example:

```java
FilterCriteria filterCriteria = FilterCriteria.createFilter();
filterCriteria.addFilterElement(FilterCriteria.createParentFolderFilter(folderURI));
List resources = repositoryService.loadResourcesList(context, filterCriteria);
```

The `loadResourcesList` method returns a list of `ResourceLookup` objects that contain basic resource attributes (such as the name and label). To retrieve the full resource definition, you must use the `getResource` method. Further filtering can be applied to get a refined list of resources based on a given resource type, or other conditions. For example, the following retrieves all the images and JRXMLs in a folder:

```java
FilterCriteria filterCriteria = FilterCriteria.createFilter(FileResource.class);
filterCriteria.addFilterElement(FilterCriteria.createParentFolderFilter(folderURI));
FilterElementDisjunction fileTypeDisj = filterCriteria.addDisjunction();
fileTypeDisj.addFilterElement(FilterCriteria.createPropertyEqualsFilter("fileType", FileResource.TYPE_IMAGE));
fileTypeDisj.addFilterElement(FilterCriteria.createPropertyEqualsFilter("fileType", FileResource.TYPE_JRXML));
List resources = repositoryService.loadResourcesList(context, filterCriteria);
```

To develop a more detailed understanding of the filter criteria, please refer to the API Javadoc.

### 10.1.3 Report Data Source Service API

JasperReports Server comes with built-in support for JDBC, JNDI, Mondrian, and XML/A data sources for reporting purposes. Each of these custom data sources has an implementation of `com.jaspersoft.jasperserver.api.metadata.jasperreports.service.ReportDataSourceService` interface. This service is responsible for setting up and tearing down data source connections in the server.

The `setReportParameterValues(Map parameterValues)` is called before running a report and creates the resources needed by JasperReports to obtain a `JRDataSource`, then it adds them to the parameter map.

The `closeConnection()` method cleans up any resources allocated in `setReportParameterValues()`.

The custom data source API enables easy integration of a new `ReportDataSourceService` implementation.

You can find further details on creating and configuring custom data sources in the *JasperReports Server Administrator Guide* or the *JasperReports Server User Guide* for the Community Project.

### 10.1.4 Report Scheduling API

Reports on the server can be executed asynchronously using the Report Scheduling API. Asynchronous report execution involves defining the report job and using the report scheduling service to schedule it.
A report job definition consists of:

- Report attributes. Each job must be linked to a single JasperReport on the server. If applicable, the job must also contain values for the report input parameters.

- Scheduling attributes. Instruct the scheduler when to execute the job. A report job can be a one-time job that can be launched immediately or at a specified moment, or a recurring job that runs repeatedly at specified times.

  Two types of recurrence are supported by default:
  
  - Simple recurrence can be used to schedule a job to repeat at fixed time intervals, such as every 4 hours, every 2 days, or every week. The job start date attribute is used to specify the moment of the first occurrence. The user can specify the number of times the job should occur or an end date for the job.
  
  - Calendar recurrence can be used to schedule a job to repeat at specified calendar moments, such as at 8 PM every work day or on the first of every month.

- Output attributes. Instruct the scheduling service on what to do with the report output. The job creator has to specify the report output formats and the repository location where the report output is saved. It can also specify one or more addresses to which email notifications is sent. The notifications can include the report output.

A report job definition is an instance of the `com.jaspersoft.jasperserver.api.engine.scheduling.domain.ReportJob` bean class. To instantiate a new report job definition, you would use code similar to the following:

```java
ReportJob job = new ReportJob();
job.setLabel("foo"); //set the job label
job.setDescription("bar"); //set the job description
```

The job source is created as a sub-bean:

```java
ReportJobSource source = new ReportJobSource();
source.setReportUnitURI("/test/reportURI"); //set the report to run
Map params = new HashMap();
params.put("param1", new Integer(5));
params.put("param2", "value2");
source.setParametersMap(params); //set the report input parameter values
job.setSource(source); //set the job source
```

The job trigger is used to specify when the job should occur. The basic `com.jaspersoft.jasperserver.api.engine.scheduling.domain.ReportJobTrigger` bean type is abstract; two concrete types extend it: `com.jaspersoft.jasperserver.api.engine.scheduling.domain.ReportJobSimpleTrigger` and `com.jaspersoft.jasperserver.api.engine.scheduling.domain.ReportJobCalendarTrigger`.

For example, to create a job that fires 20 times every 10 days you would use code similar to this:

```java
Date startDate = ...;
ReportJobSimpleTrigger trigger = new ReportJobSimpleTrigger();
trigger.setStartDate(startDate);
trigger.setOccurrenceCount(20);
trigger.setRecurrenceInterval(10);
trigger.setRecurrenceIntervalUnit(ReportJobSimpleTrigger.INTERVAL_DAY);
job.setTrigger(trigger);
```
Next, you need to specify the job output attributes:

```java
job.setBaseOutputFilename("foo"); // the base output file name
job.addOutputFormat(ReportJob.OUTPUT_FORMAT_PDF); // output PDF
job.addOutputFormat(ReportJob.OUTPUT_FORMAT_HTML); // and HTML
```

```java
ReportJobRepositoryDestination repositoryDestination = new ReportJobRepositoryDestination();
repositoryDestination.setFolderURI("/test/scheduled");
// the repository folder where to output the files
repositoryDestination.setSequentialFilenames(true);
// append a timestamp to the file names
job.setContentRepositoryDestination(repositoryDestination);
```

Optionally, you can instruct the reporting scheduler to send a notification once the job completes:

```java
ReportJobMailNotification mailNotification = new ReportJobMailNotification();
mailNotification.addTo("john@smith.com"); // the recipient
mailNotification.setSubject("Scheduled report"); // the subject
mailNotification.setMessageText("Executed report.\n"); // the message body
mailNotification.setResultSendType(ReportJobMailNotification.RESULT_SEND_ATTACHMENT);
// send the report output as attachments
job.setMailNotification(mailNotification);
```

The scheduling service is used to schedule a report job and provide information about existing jobs. The built-in scheduling service implementation consists of a Hibernate-based component that persists job definitions and a Quartz scheduler that schedules and fires the jobs.

Once a report job definition is created, it is passed to the scheduling service:

```java
ReportSchedulingService schedulingService = ...;
ReportJob job = ...
schedulingService.scheduleJob(executionContext, job);
```

The `ReportSchedulingService` is not a Java object that is included in the API; instead, you must inject a `ReportSchedulingService` instance (that is, define a bean called `reportSchedulingService` using Spring). Your bean should include custom code that produces the desired behavior. For more information on defining a bean, see section 10.1.1.1, “Accessing API Implementations Using the Spring Framework,” on page 124.

The `getScheduledJobs` of the scheduling service can be used to retrieve the list of scheduled jobs for a report unit. The list consists of instances of `com.jaspersoft.jasperserver.api.engine.scheduling.domain ReportJobSummary` that contain basic job attributes, plus runtime attributes such as job status and previous/next fire times.

To load the full report job definition you would use the `getScheduledJob` method. The job definition can be altered then updated using the `updateScheduledJob` service method. The scheduling service also contains methods for the removal of one or several existing report jobs: `removeScheduledJob` and `removeScheduledJobs`.

Caution is advised when updating the job trigger, since the original Quartz trigger is dropped and a new trigger that corresponds to the updated attributes is created.
10.1.5 Users and Roles API

Access to all JasperReports Server functionality is based on assigned user-level and role-level permissions. Thus, managing users and roles is a critical aspect of the public API.

The `com.jaspersoft.jasperserver.api.metadata.user.service.UserAuthorityService` interface has methods for creating, modifying, and removing users and roles. The API manipulates only these two types of entities for which public interfaces are available:

- Users are represented by the `com.jaspersoft.jasperserver.api.metadata.user.domain.User` interface.
- Roles are represented by the `com.jaspersoft.jasperserver.api.metadata.user.domain.Role` interface.

A new user can be defined in a few easy steps:

```java
User workingUser = userAuthService.newUser(null);
workingUser.setTenantId("organization_1");
workingUser.setUsername("john");
workingUser.setPassword("changeme");
workingUser.setFullName("John Doe");
workingUser.setEnabled(true);
workingUser.setExternallyDefined(false);
userAuthService.putUser(null, workingUser);
```

The `setTenantId` method specifies the organization the user belongs to. However, note the following:

- If you are using commercial editions of JasperReports Server, you should use the method in most cases, but if your instance hosts only a single instance, this method should set most user’s organization to the default (`organization_1`).
- If this is a special administrative user (similar to superuser) that shouldn’t be affiliated with an organization, do not call `setTenantId`.

To get the user information from the database, the `getUser` method can be called by providing the user name.

Users can be removed from the database by name with the `deleteUser` method.

Equivalent methods for managing roles are available in the `UserAuthorityService`. You can assign users to roles using the following two methods:

```java
public void addRole(ExecutionContext context, User aUser, Role role);
public void removeRole(ExecutionContext context, User aUser, Role role);
```

Additional methods for finding users with specific roles are available; you can find details about them if you consult the Javadoc for the `UserAuthorityService` interface.

10.1.6 Object Permissions API

An object permission represents the right to perform a certain action on a repository resource by a user. Access to all functionality relies on a security mechanism that checks if the user has the right to perform the current action on the given object or resource. The `com.jaspersoft.jasperserver.api.metadata.user.service.ObjectPermissionService` interface grants or revokes permissions on an object to users and roles; it has methods for creating and removing permissions on objects and recipients.

An object permission is represented by an implementation instance of the `com.jaspersoft.jasperserver.api.metadata.user.domain.ObjectPermission` interface and holds information about the recipient (User or Role instance), the type of permissions granted to the user (combination of numeric constants defined in the Spring Security class `org.springframework.security.acl.basic.SimpleAclEntry`), and the resource to which they apply.
The following illustrates how to create an object permission on an image resource within the repository:

```java
Resource resource = ..an existing repository resource..
ObjectPermission permission = objectPermissionService.newObjectPermission(null);
permission.setURI(resource.getURIString());
permission.setPermissionRecipient(user);
permission.setPermissionMask(SimpleAclEntry.READ);
objectPermissionService.putObjectPermission(null, permission);
```

Revoking object permissions can be done with one of the following public methods exposed by this service:

```java
public void deleteObjectPermission(ExecutionContext context, 
    ObjectPermission objPerm);
public void deleteObjectPermissionForObject(ExecutionContext context, 
    Object targetObject);
public void deleteObjectPermissionsForRecipient(ExecutionContext context, 
    Object recipient);
```

### 10.1.7 OLAP Connection API

This section describes functionality that is available only in Jaspersoft OLAP. Contact Jaspersoft to obtain the software.

OLAP interactions through the Jaspersoft OLAP user interface and web services based on XML/A are supported by the repository and dedicated APIs. The repository can contain the following OLAP-related objects:

- **OlapUnit.** This is the data needed for an analysis view. It contains an MDX query and an OLAPClientConnection.
- **MondrianConnection.** Implementor of OLAPClientConnection. It contains a Mondrian schema and a JDBC or JNDI connection.
- **XMLAConnection.** Implementor of OLAPClientConnection. It contains a URL to an XML/A service and an optional data security definition.
- **MondrianXMLADefinition.** Jaspersoft OLAP can operate as an XML/A server on top of Mondrian connections. These objects catalog what can be accessed through XML/A.

The OLAP Connection API provided by Jaspersoft OLAP is simple, as most of the underlying functionality is within Mondrian (OLAP query engine, XML/A server) or Jpivot (OLAP user interface). This call creates a JPivot-compatible OlapModel, based on the relevant OlapUnit:

```java
public OlapModel createOlapModel(ExecutionContext context, OlapUnit olapUnit );
```

The call should be made and the model object put into the user session before redirecting to the JPivot JSP viewOlap.jsp. An example is in ViewOlapModelAction:

```java
OlapUnit olapUnit = (OlapUnit)
    IRepository().getResource(executionContext, viewURI);
OlapModel model =
    getOlapConnectionService().createOlapModel(executionContext, olapUnit);
```
The following call retrieves the server-wide Mondrian properties:

```java
// create a local Mondrian OLAP connection corresponding to a Mondrian connection resource defined in the repository MondrianConnection connectionResource = ..get the resource from the repository.
mondrian.olap.Util.PropertyList connectProperties =
    olapConnectionService.getMondrianConnectProperties(context, connectionResource);
mondrian.olap.Connection olapConnection =
    mondrian.olap.DriverManager.getConnection(connectProperties, null, false);
mondrian.olap.Query query = olapConnection.parseQuery("..MDX query..");  
mondrian.olap.Result olapResult = olapConnection.execute(query); ...
```

See the *Mondrian Technical Guide* for details

These calls can be used to validate an OlapUnit object while you are editing in the UI:

```java
public ValidationResult validate(ExecutionContext context, OlapUnit unit);
public ValidationResult validate(ExecutionContext context,
    OlapUnit unit,
    FileResource schema,
    OlapClientConnection conn,
    ReportDataSource dataSource);
```

### 10.1.8 Flushing the Analysis Cache Using the API

Jaspersoft OLAP maintains a cache of previously-retrieved results in order to achieve high performance. There is a simple interface for flushing the entire cache programmatically, an action that is required when new data is inserted into databases while Jaspersoft OLAP is running.

```java
public interface OlapManagementService {
    public void flushOlapCache();
}
```

The interface can be configured as a Spring bean or instantiated with a standard Java constructor call. `com.jaspersoft.jasperserver.api.metadata.olap.service.impl.OlapManagementServiceImpl` is the default implementation of the interface.

### 10.2 Ad Hoc Launcher Java API

This section describes functionality that can be restricted by the software license for JasperReports Server. If you don’t see some of the options described in this section, your license may prohibit you from using them. To find out what you’re licensed to use, or to upgrade your license, contact Jaspersoft.

When users want to create a new report in the Ad Hoc Editor, they are normally presented with a page that allows them to choose the Topic, Domain, or OLAP client connections that will form the basis of the report. The selected Topic or Domain provides a query and a list of fields that can be used in the new report.

The Ad Hoc Launcher API allows a developer to replace the Source dialog with an alternative user interface for selecting data. This replacement for the Source dialog is referred to as an *Ad Hoc launcher* because the user can launch the Ad Hoc Editor interface after selecting and setting up the data. Whereas the Source dialog can only give users a choice of the queries previously stored in the repository, an Ad Hoc launcher implementation enables them to have full control over the data source, query, and fields to be used. This gives developers a wide latitude for creating data selection interfaces tailored to the needs of their users.
Creating an Ad Hoc launcher involves modifying the JasperReports Server web application by adding one or more servlet pages that gather user input for the setup required by the Ad Hoc Editor. When the user is ready to start laying out the report, the Ad Hoc launcher starts up the Ad Hoc Editor by redirecting to its URL.

Examples of possible Ad Hoc launcher applications include the following:

- A generic SQL query builder which allows the user to choose a JDBC data source, view the tables available from it, and build a query against one or more of the tables. This approach is demonstrated in browseDB, the sample Ad Hoc launcher that is described below.
- A query builder based on application-specific metadata, such as a list of pre-defined queries maintained in its own table.
- An interface for constructing queries against a custom data source that has metadata facilities, such as Hibernate.
- A “dumb” query builder in which the user chooses a data source from the repository and enters query text and field definitions manually. This would be analogous to creating a JasperReport by editing the JRXML.

The details of the setup required in an Ad Hoc launcher are described in the next section.

10.2.1 Communicating with the Ad Hoc Editor using AdhocData

This section describes how the Ad Hoc Editor initializes its reporting data, both when a Topic is selected and when an Ad Hoc launcher is active.

The Ad Hoc Editor uses the AdhocData Java class to describe reporting data. If the user starts the Ad Hoc Editor from the Topics page, AdhocData is set up as follows:

1. A new instance of AdhocData is created.
2. The report unit representing the selected Topic is read.
3. AdhocData.initDatasource() is called with the URI of the data source used by the Topic.
4. The JRXML resource for the Topic’s report is read.
5. AdhocData.setQueryText() is called with the query from the JRXML.
6. AdhocData.setQueryLanguage() is called with the query language from the JRXML.
7. The list of fields defined in the JRXML is read.
8. Each field is turned into an AdhocField instance and passed to AdhocData.addField().
9. If the user is creating a new report, the desired report type (table, crosstab, chart) is passed to AdhocData.setReportType().

If an Ad Hoc launcher is being used, it needs to place an instance of AdhocBaseData in the servlet session, where the Ad Hoc Editor can use it. The following code examples are from the BrowseDBController.java source file from the browseDB example in section 10.2.3, “A Sample Ad Hoc Launcher,” on page 138.
To initialize an instance of the `AdhocData` class for use by the Ad Hoc Editor:

1. Create a new `AdhocData` instance, then initialize the data source by calling `initDatasource()` with the URI of a data source in the repository (which in this example is coming from an HTTP request parameter). Use `SessionAttributeManager` to create a new client key and store the `AdhocData` instance in the session:

   ```java
   String datasourceURI = req.getParameter(DATASOURCE_URI);
   AdhocBaseData data;
   long clientKey;
   SessionAttributeManager sessMgr = SessionAttributeManager.getInstance();

   // if you have a datasource URI, you just started...
   // so we need to init AdhocData and put it in session
   if (datasourceURI != null) {
       data = new AdhocBaseData();
       data.initDatasource(engine, datasourceURI);
       // get new client key for Ad Hoc
       clientKey = sessMgr.createClientKey();
       sessMgr.setSessionAttribute(AdhocConstants.ADHOC_DATA, data, clientKey, req);
   } else {
       // handle existing report
   } // if you have a datasource URI, you just started...
```

2. Initialize the query to be used for the Topic by calling `setQueryText()` and `setQueryLanguage()`:

   ```java
   String tableName = dbdata.getSelectedTable().getName();
   data.setQueryLanguage("sql");
   StringBuffer query = new StringBuffer("select * from " + tableName);
   Iterator sfi = dbdata.getSelectedTable().getFieldList().iterator();
   while (sfi.hasNext()) {
       DBField f = (DBField) sfi.next();
       DBField jf = f.getJoinField();
       if (jf != null) {
           query.append("on (" + fromField + " = " + toField + ")");
       }
   }
   data.setQueryText(query.toString());
   ```

3. Add field definitions by creating new `AdhocField` instances and calling `addField()`:

   ```java
   if (jf != null) {
       Iterator jtfi = jf.getTable().getFieldList().iterator();
       while (jtfi.hasNext()) {
           DBField jtf = (DBField) jtfi.next();
           AdhocField af = new AdhocField(jtf.getName(), getJRType(jtf.getDbType()));
           af.setDisplay(jtf.getLabel());
           data.addField(af);
       }
   }
   ```
AdhocField instances require the following information:

- **Name.** The name of the field (set in the AdhocField constructor, or by calling `setName()`).
- **Type.** The name of the Java class representing the type of the field's data (set in the AdhocField constructor, or by calling `setType()`).
- **Display.** The string used as a label for the field in the Ad Hoc Editor (set by calling `setDisplay()`).

4. Set the report type by calling `AdhocData.setAdhocReportType()`. Valid values are `table`, `crosstab`, and `chart`:

```java
String reportType = req.getParameter(AdhocAction.REPORT_TYPE);
if (reportType != null) {
    data.setAdhocReportType(reportType);
}
```

### 10.2.2 Integration with JasperReports Server

You can register your launcher with the Ad Hoc Editor so that, if the user starts a report with your launcher and later saves it, the launcher can be started again when the new report is edited.

**To register an Ad Hoc launcher:**

1. Choose a reference name for the Ad Hoc Launcher; this example assumes the name is `test`.
2. Define a Spring bean that registers the launcher’s name and URI. The following shows a portion of the `<js-install>/samples/customAdHoc/webapp/WEB-INF/applicationContext-adhoc-custom.xml` file:

```xml
<bean class="com.jaspersoft.jasperserver.api.common.util.spring.GenericBeanUpdater">
    <property name="definition" ref="setCustomAdhocProps"/>
    <property name="key" value="test"/>
    <property name="valueType" value="idRefMap"/>
    <property name="value">
        <idref bean="sampleAdHocLauncher"/>
    </property>
</bean>

<util:map id="sampleAdHocLauncher">
    <entry key="editorURI" value="browseDB/browseDB.html?action=displayTables"/>
</util:map>
```

In this example, note the property named `key` with the value `test`; this element defines the name described in **step 1**.

3. Before you launch the Ad Hoc Editor, call `AdhocData.setCustomType()` with the name used to register the application.

```java
AdhocBaseData data = (AdhocBaseData)
    SessionAttributeManager.getInstance().getSessionAttribute(
    AdhocConstants.ADHOC_DATA, req);

data.setCustomType("test");
```

The launcher can also be used to store custom data in a saved Ad Hoc report unit, so that when you relaunch the report, you can retrieve the data conveniently. The data is persisted as a byte stream resource in the report unit.

**To store data in a report unit:**

1. Create an implementation of `CustomAdhocDataHandler`, which tells the Ad Hoc Editor how to create, serialize, and deserialize the object you want to persist.
2. Call `AdhocData.setCustomDataHandler()` with an instance of your implementation:

```java
data.setCustomDataHandler(new BrowseDBDataHandler());
BrowseDBData dbdata = (BrowseDBData) data.getCustomData();
```

3. Call `AdhocData.getCustomData()` to access your custom data object.

### 10.2.3 A Sample Ad Hoc Launcher

`browseDB` is an example of how you might use the Ad Hoc Launcher API to create your own query editor. While it illustrates the API’s basic functionality, and is a useful example for administrators and developers who plan to implement their own launcher, it isn’t intended for end users. The application supports only the most basic SQL functions; it does not support such operations as a 2-column primary key or joins from parent to child (you must join child to parent).

In order to construct queries in `browseDB`, you must know the primary keys of the tables in the data source and any database constraints on joins.

#### 10.2.3.1 Installing the Sample Editor

To install the `browseDB` editor:

1. Determine the location of the JasperReports Server web application; for the default installation with the bundled Tomcat, it is `<js_install>/apache-tomcat/webapps/jasperserver-pro`.
2. In an appropriate editor, open the file `<js_install>/samples/customAdHoc/build.xml`.
3. In the file, set the `webAppDir` property to the web application location in step 1.

![Figure 10-1 Setting the webAppDir Property](image)

4. Save and close the file.
5. On a command line, change directory to `<js_install>/samples/customAdHoc`.
6. Run `<js_install>/ant/bin/ant deploy`.

   The ant utility compiles the Java source, deploys the `browseDB` files to the appropriate folders, and displays a BUILD SUCCESSFUL message.

   If there are permissions errors, make sure you have ownership or write permission in the `<js-install>` folder.

10.2.3.2 Using browseDB

To define a simple query in browseDB:

1. In a web browser, start JasperReports Server and login with administrator privileges.

3. Select a data source from the drop-down and click **Display Tables**. In this example, select the FoodmartDataSourceJNDI. A list of the tables in the data source appears.

4. To select the main table for the query, click that table’s name. In this example, click the **employee** table. The selected table appears at the top of the list as the **Main query table**. It is expanded to show all the columns in the table.
To join a table to the query:

1. Before you can join a table, you must designate one of its columns as a primary key. Expand the table you are joining by clicking 📚. If you click the folder icon or name instead, that table becomes the main table of the query, replacing the one you selected. In this example, expand the store table.
2. Indicate the additional table’s primary key by clicking the appropriate column name and clicking **Set the selected column as primary key** at the top of the page. In this example, select the `store_id` column of the `store` table as the primary key.

The additional table is added to the **Join** drop-down. The drop-down shows all the additional tables for which you select a primary key. In this example, the `store` table is added to the drop-down.

3. In the **Join** drop-down, select the table to add. Since you only expanded the `store` table, it should already be selected.

4. In the main query table, select the column to which the additional table should be joined. In this example, select the `employee` table’s `store_id` column.

5. Click **Join selected column on main table to**. The additional table (`store`) is added to the main query table as the join table. Expand the join table to see all of its fields.
To open the Ad Hoc Editor with this query:
1. In the Pick datasource window, click a report type (Table, Crosstab, or Chart). In this example, click **Table**. The Ad Hoc Editor opens. All the columns of the selected query tables appear in the editor’s list of available fields.
2. Define and save your report in the usual way. For more information, refer to the *JasperReports Server User Guide*.

To open a saved Ad Hoc Report created with the sample Ad Hoc launcher:
1. Locate the report saved from browseDB in the repository.
2. Right-click the report and select **Open in Designer** from the menu that appears.
3. The browseDB interface appears with the main query table and join table in the same state as when the report was last saved.
4. Click **Start Ad Hoc** to open the Ad Hoc Editor. Because this report was saved, there is no longer the choice of Table, Crosstab, or Chart. The report will open to the format in which it was saved.

**Figure 10-8** BrowseDB Report Opened with Ad Hoc Launcher

10.3 Repository HTTP API

The HTTP interface provides an easy way to implement the API for accessing repository objects. However, the HTTP interface is not embeddable in the same way that the web services and Java APIs can be embedded in non-Jaspersoft applications. Rather, the HTTP interface is used primarily as shortcuts or entry points to commonly used features or content. Typically, the HTTP interface is accessed programmatically by generating the URL that returns HTML that displays either the desired object (in the case of report execution and repository URLs) or the content of repository objects (such as report output in the form of content resources).

For example, the SuperMart demonstration data that ships with JasperReports Server commercial editions makes frequent use of the HTTP interface. This simple API allows users to interact with the SuperMart dashboard, dynamically pass parameters from a report to an analysis view, and display the user’s personal folder based on the login ID.

As shown in the following examples, the major entry points are:
- `flow.html`
- `olap/**`
- `fileview/**"
10.3.1 Executing ReportUnits

The following sections provide examples and details about URLs that execute reports.

10.3.1.1 Simple Report Execution

The HTTP interface can execute and export reports within the JasperReports Server web application.

The following example calls a report without any parameters and exports to the default format (HTML):

The URL is:

```
```

This is the simplest possible report execution URL. The following section explains more advanced options.

10.3.1.2 Passing ReportUnit Execution Parameters

The following example executes the same report as shown in the previous section, but also passes 4012 as an input control parameter and exports to PDF instead of HTML:

```
```

Note the URL parameters:

- `&customerID=4012` indicates that the value 4012 should be passed to the input control called customerID. The report returns data about customer 4012.
- `&output=pdf` indicates that the output should be generated in PDF format.

If the report parameter supports multiple values, you can specify them using the ampersand (&); for example:

```
```

This section describes more such parameters.

The report execution parameters can either be reserved parameters that the server uses to determine general attributes of the report execution, or they can be arbitrary parameters that correspond to the report’s input controls/parameters. The parameters are specified as standard HTTP GET parameters (that is, they are in the form `name=value` and are separated by ampersands (&)).

The following general parameters are recognized by JasperReports Server:

- `reportUnit` specifies the URI of the report unit resource in the repository.
- `output` (optional) specifies the output format. Values for this parameter are keys for the report exporters configured in the server. By default, the server recognizes the following output types: `pdf` for PDF, `xls` for Excel, `rtf` for RTF, `csv`
for CSV and swf for the Flash report viewer. When this parameter is not specified, the default format is HTML displayed in the report viewer.

- `reportLocale` specifies the locale in which the report should be executed. A locale is passed as code consisting of a lower-case two letter ISO-639 language code, followed by optional upper-case two letter ISO-3166 country code and a locale variant, separated by underscore (the format is identical to the standard Java locale programmatic names).

- `userlocale` also specifies a locale, but in this case, it specifies the locale of the user. It can be added to the startup URI, as in this example:


- `j_username` and `j_password` pass the credentials to authenticate a user with the server. The user name should correspond to a valid user, and password should be the user password (in clear text). If you use a commercial edition and host more than one organization, the organization ID or alias must be passed along with the user ID as part of the `j_username` parameter. The format is `j_username=userID|orgID`.

If such credential parameters are not present, and no authenticated server session exists, and the server is not configured to use automatic authentication mechanisms (such as single sign-on), the server prompts the user for a user name and a password; after logging in, she is redirected to the report execution page. These authentication parameters let the user skip the login page and directly access the report execution page.

Note that these two parameters are not specific to report execution URLs; they can be used for any URLs that point to a JasperReports Server web page.

- `pageIndex` (optional) specifies the initial page that should be displayed when launching the target report. The page index is 1-based; if a negative page index or a page index greater than the number of pages in the report is used, the report opens at the first page. This parameter is only effective when HTML is used as output.

- `anchor` (optional) specifies the name of an anchor from the target report at which the report should open. If an anchor with the specified name is not found in the report, the first page of the report is shown. The parameter is only effective when `pageIndex` is not specified and when HTML is used as output.

In addition to these standard parameters, report execution URLs can contain parameters that provide values for the report input controls/parameters. The URL parameter names must match the name of the corresponding report input control. The values used for such URL parameters depend on the type of input control:

- For simple, single value input controls, the value is a URL parameter value:
  - If the type of the input control is text, the URL parameter value is directly used as input control value.
  - If the type of the input control is numeric, the URL parameter value is the numerical value formatted according to standard rules, using a period (.) as the decimal separator.
  - If the type of the input control is date or date/time, the URL parameter value is the date/time value formatted according to the `yyyyMMddHHmmss` format (for example, 20090621054500 for June 2, 2009, 05:45 AM). This value is interpreted as being in the server's default time zone.

- For boolean (check box) input controls, the URL parameter value is either true or false.

- For input controls that refer to static list of values, the URL parameter value is the key/value of the list entry. For example, to select first value in a list called ListInput, use the parameter `&ListInput=1`.

- For input controls that rely on a query, the URL parameter value corresponds to the query key/value column. For example, to set a query-based control to the value `l_meade`, use the parameter `&QueryInput=lmeade`.

- For multi-value input controls, multiple occurrences of the same URL parameter can be use. For example, `parameter=value1&parameter=value2&parameter=value3`.

  You can use a special marker URL parameter for multi-value input controls to specify that an empty list should be used as input control value. In this case, the name of the marker URL parameter is the name of the input control, prefixed by an underscore; it doesn't require any value. This is useful when a multi-value input control has a non-empty list as its default value, and the user wants to override the default value with an empty list.

  For example, consider a multi-value input control named MultiInput. By default, this parameter is list of two values. To override this list of values with another set of two value, you could use the parameter `&MultiInput=item1&MultiInput=item2`, where `item1` and `item2` are the overriding list values. To override the default with an empty list, use the parameter `&MultiInput=`.

If the report parameter values included in a report execution URL are not valid (for example, if a required parameter is missing or a parameter value is not valid), the user is prompted with the input controls so he can correct the values.
10.3.2 Linking to Content

The HTTP interface can return generated content saved to the repository in PDF, HTML, Excel, or RTF format.

The following example links to a PDF file stored in the repository:


10.3.3 Viewing Resources in the Repository

The following example displays all resources saved in a the /reports/test folder in the repository:

http://<host>:<port>/<context>/flow.html?_flowId=repositoryFlow&folder=/reports/test

The following example displays all resources of type olapview (analysis view) saved in all folders in the repository:

http://<host>:<port>/<context>/flow.html?_flowId=olapViewListFlow

10.4 Ad Hoc Editor HTTP API

This section describes functionality that can be restricted by the software license for JasperReports Server. If you don't see some of the options described in this section, your license may prohibit you from using them. To find out what you're licensed to use, or to upgrade your license, contact Jaspersoft.

The HTTP API enables you to control the Ad Hoc Editor by issuing commands programmatically instead of using the UI. You can build a complete custom user interface to the Ad Hoc Editor with the available action codes or only use a small subset of them to perform specific tasks (such as changing the style of the report). Only one action code can be issued at a time, but the effect of consecutive actions accumulate just as if they were issued in the UI. Action codes are passed to the server through URIs.

In the URIs, the .html filename should specify the type of report to which the action applies; for example, use crosstab.html for crosstab reports. For action codes that apply to all report types, such as run, the server properly interprets the action code no matter which .html filename you specify. Jaspersoft recommends using table.html in such cases.

In action codes that specify position or index, position 0 is the left-most position in the report as it appears on the screen or the topmost position in a hierarchy of fields, whichever applies. For example, in a hierarchy of Country, State, and City, Country is in position 0.

Some actions include parameters that have a limited range of permitted values, such as paper size. The accepted values for these parameters are defined in the applicationContext-adhoc.xml configuration file.

10.4.1 Examples URIs

To use the Fall style in a table report:

/jasperserver-pro/adhoc/table.html?action=setTheme&t=fall

To use the countAll summary function in the first column (that is, i = 0) of a table report:

/jasperserver-pro/adhoc/table.html?action=setColumnSummaryFunction&f=Count&i=0

To insert the ShipCountry rowGroup in the second position in a crosstab report:

/jasperserver-pro/adhoc/crosstab.html?action=insertRowGroup&f=ShipCountry&i=1
## 10.4.2 Action Codes

### Table 10-2 Action Codes for All Report Types

<table>
<thead>
<tr>
<th>Action Code</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run</td>
<td>–</td>
<td>Runs the report.</td>
</tr>
<tr>
<td>save</td>
<td>AruName – report name</td>
<td>Saves the current report in the AruFolder with the indicated report name and description.</td>
</tr>
<tr>
<td></td>
<td>AruFolder – repository folder</td>
<td></td>
</tr>
<tr>
<td></td>
<td>aruDesc – report description</td>
<td></td>
</tr>
<tr>
<td>setPageOrientation</td>
<td>O – orientation (the letter O rather than the number zero)</td>
<td>Sets the orientation of the report on the page. Valid values are Landscape or Portrait (case-sensitive).</td>
</tr>
<tr>
<td>setPaperSize</td>
<td>s – size</td>
<td>Sets the page size of the printout. By default, valid values are a4, letter, and content (case-sensitive). a4 and letter constrain the report to the indicated page size; content removes the constraint and allows the page to grow infinitely to a size large enough to contain the entire report.</td>
</tr>
<tr>
<td>setTheme</td>
<td>t – theme name</td>
<td>Applies the indicated style to the report.</td>
</tr>
<tr>
<td>setTitle</td>
<td>l – title label</td>
<td>Sets the title of the report. Set l to _null for no title.</td>
</tr>
<tr>
<td>viewReport</td>
<td>–</td>
<td>When set to true, display the report in presentation mode.</td>
</tr>
</tbody>
</table>

### Table 10-3 Action Codes for Table Reports (table.html)

<table>
<thead>
<tr>
<th>Action Code</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>insertColumn</td>
<td>i – position to insert into</td>
<td>Adds a column at the indicated position.</td>
</tr>
<tr>
<td></td>
<td>f – name of the field</td>
<td></td>
</tr>
<tr>
<td>insertGroup</td>
<td>i – position to insert into</td>
<td>Adds the indicated field as a group at the indicated position.</td>
</tr>
<tr>
<td></td>
<td>f – name of the field</td>
<td></td>
</tr>
<tr>
<td>moveColumn</td>
<td>c1 – position to move from</td>
<td>Moves the column at position c1 to position c2. The c2 position is calculated after the element has been removed from c1.</td>
</tr>
<tr>
<td></td>
<td>c2 – position to move to</td>
<td></td>
</tr>
<tr>
<td>moveGroup</td>
<td>i1 – position to move from</td>
<td>Moves the group at position i1 to position i2. The i2 position is calculated after the element has been removed from i1.</td>
</tr>
<tr>
<td></td>
<td>i2 – position to move to</td>
<td></td>
</tr>
<tr>
<td>removeColumn</td>
<td>i – position of the column to remove</td>
<td>Removes the column at the indicated position.</td>
</tr>
<tr>
<td>removeGroup</td>
<td>i – position of the group to remove</td>
<td>Removes the group at the indicated position.</td>
</tr>
<tr>
<td>resizeColumn</td>
<td>i – position of the column</td>
<td>Sets the column at position i to the indicated width, measured in pixels.</td>
</tr>
<tr>
<td></td>
<td>w – column width</td>
<td></td>
</tr>
</tbody>
</table>
### Table 10-3  Action Codes for Table Reports (table.html), continued

<table>
<thead>
<tr>
<th>Action Code</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setColumnHeader</td>
<td>i – position of the column</td>
<td>Sets the header label of the indicated column. Set l to _null for no header.</td>
</tr>
<tr>
<td></td>
<td>l – column label</td>
<td></td>
</tr>
<tr>
<td>resort</td>
<td>so – serialized list of column names and sorting orders</td>
<td>Sorts the table. TRUE indicates ascending sort (A-Z); FALSE indicates descending sort (Z-A). For example, for columns named Region and ShippingDate: so=sortFields:{Region</td>
</tr>
<tr>
<td>setColumnMask</td>
<td>m – mask</td>
<td>Sets the data format mask of the indicated column. For more information about format masks, refer to the JasperReports Server Administrator Guide.</td>
</tr>
<tr>
<td></td>
<td>i – position of the column</td>
<td></td>
</tr>
<tr>
<td>SetColumnSummary Function</td>
<td>i – position of the column</td>
<td>Sets the summary function of the indicated column. For more information about summaries, refer to the JasperReports Server User Guide.</td>
</tr>
<tr>
<td></td>
<td>f – name of the function</td>
<td></td>
</tr>
<tr>
<td>setGroupLabel</td>
<td>g – position of the group</td>
<td>Sets the label of the indicated group. Set l to _null for no label.</td>
</tr>
<tr>
<td></td>
<td>l – group label</td>
<td></td>
</tr>
</tbody>
</table>

### Table 10-4  Action Codes for Chart Reports (chart.html)

<table>
<thead>
<tr>
<th>Action Code</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>insertColumn</td>
<td>i – position to insert into</td>
<td>Adds a column (the specified field) at the indicated position.</td>
</tr>
<tr>
<td></td>
<td>f – name of the field</td>
<td></td>
</tr>
<tr>
<td>insertGroup</td>
<td>i – position to insert into</td>
<td>Adds the indicated field as a group at the indicated position.</td>
</tr>
<tr>
<td></td>
<td>f – name of the field</td>
<td></td>
</tr>
<tr>
<td>moveColumn</td>
<td>c1 – position to move from</td>
<td>Moves the column at position c1 to position c2. The c2 position is calculated after the element has been removed from c1.</td>
</tr>
<tr>
<td></td>
<td>c2 – position to move to</td>
<td></td>
</tr>
<tr>
<td>moveGroup</td>
<td>i1 – position to move from</td>
<td>Moves the group at position i1 to position i2. The i2 position is calculated after the element has been removed from i1.</td>
</tr>
<tr>
<td></td>
<td>i2 – position to move to</td>
<td></td>
</tr>
<tr>
<td>removeColumn</td>
<td>i – position of the column</td>
<td>Removes the column at the indicated position.</td>
</tr>
<tr>
<td>removeGroup</td>
<td>i – position of the group to remove</td>
<td>Removes the group at the indicated position.</td>
</tr>
<tr>
<td>resizeColumn</td>
<td>i – position of the column</td>
<td>Sets the column at position i to the indicated width, measured in pixels.</td>
</tr>
<tr>
<td></td>
<td>w – column width</td>
<td></td>
</tr>
<tr>
<td>setColumnHeader</td>
<td>i – position of the column</td>
<td>Sets the header label of the indicated column. Set l to _null for no header.</td>
</tr>
<tr>
<td></td>
<td>l – column label</td>
<td></td>
</tr>
</tbody>
</table>
### Table 10-5 Action Codes for Chart Reports (chart.html), continued

<table>
<thead>
<tr>
<th>Action Code</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>insertColumnGroup</td>
<td>f – name of the field</td>
<td>Adds the indicated field as a column group at position i.</td>
</tr>
<tr>
<td></td>
<td>i – position to insert into</td>
<td></td>
</tr>
<tr>
<td>insertMeasure</td>
<td>f – name of the field</td>
<td>Adds the indicated field as a measure at position i.</td>
</tr>
<tr>
<td></td>
<td>i – position to insert into</td>
<td></td>
</tr>
<tr>
<td>insertRowGroup</td>
<td>f – name of the field</td>
<td>Adds the indicated field as a row group at position i.</td>
</tr>
<tr>
<td></td>
<td>i – position to insert into</td>
<td></td>
</tr>
<tr>
<td>moveColumnGroup</td>
<td>f – position to move from</td>
<td>Moves the column group at position f to position t.</td>
</tr>
<tr>
<td></td>
<td>t – position to move to</td>
<td></td>
</tr>
<tr>
<td>moveMeasure</td>
<td>f – position to move from</td>
<td>Moves the measure at position f to position t.</td>
</tr>
<tr>
<td></td>
<td>t – position to move to</td>
<td></td>
</tr>
<tr>
<td>moveRowGroup</td>
<td>f – position to move from</td>
<td>Moves the row group at position f to position t.</td>
</tr>
<tr>
<td></td>
<td>t – position to move to</td>
<td></td>
</tr>
<tr>
<td>removeColumnGroup</td>
<td>i – position of the column group to remove</td>
<td>Removes the column group at position i.</td>
</tr>
<tr>
<td>removeMeasure</td>
<td>i – position of the measure to remove</td>
<td>Removes the measure at position i.</td>
</tr>
<tr>
<td>removeRowGroup</td>
<td>i – position of the row group to remove</td>
<td>Removes the row group at position i.</td>
</tr>
<tr>
<td>setColumnGroup Categorizer</td>
<td>cat – name of the categorizer (that is, the group size option)</td>
<td>Sets the data categorizer for the column group at position i. By default, categorizers only apply to date fields.</td>
</tr>
<tr>
<td></td>
<td>i – position of the column group to categorize</td>
<td></td>
</tr>
</tbody>
</table>
This section describes the HTTP API for dashboards, which can be used to programmatically update the state of a dashboard in JasperReports Server.

### 10.5.1 Examples URIs

- To move frames as follows: `frame1` to position `x=200, y=300` and `frame2` to position `x=400, y=10`:

  ```html
  /jasperserver-pro/adhoc/dashboard.html?action=moveFrames&paramString=frame1@@200@@300**frame2@@400@@10
  ```

- To set auto refresh intervals for `frame1` and `frame2` to 5 minutes:

  ```html
  /jasperserver-pro/adhoc/dashboard.html?action=setAutoRefresh&paramString=frame1@@5**frame2@@5
  ```

### 10.5.2 Action Codes for the Dashboard Designer

Dashboard action codes are similar to Ad Hoc Editor action codes, but they are used with the dashboard designer.

These actions update the dashboard’s server model. However, they do not guarantee updates to the dashboard view in the user’s browser. The application extension is responsible for ensuring that the user’s view is refreshed.
<table>
<thead>
<tr>
<th>Action Code</th>
<th>Parameters (* indicates required)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addContentFrame</td>
<td>type* – type of resource object (for example, type = &quot;com.jaspersoft.jasperserver.api.metadata.jasperreports.domain.ReportUnit&quot;) resourceName* – the report’s name as it should be displayed in the tooltip (for example, resourceName = &quot;Freight Report&quot;) src* – the URL to set as the iframe’s source (for example, src = &quot;/flow.html?_flowId=viewReportFlow&amp;viewAsDashboardFrame=false&amp;reportUnit=/reports/samples/Freight&quot;) uri* – resource object’s repository path (for example, uri = &quot;/reports/samples/Freight&quot;) left – frame left offset top – frame top offset width – frame width height – frame height</td>
<td>Adds a frame displaying a repository object.</td>
</tr>
<tr>
<td>updateCustomContentFrame</td>
<td>frameName – name of frame (for example, frameName=&quot;frame_3&quot;) src* – type of resource object (for example, &quot;com.jaspersoft.ji.adhoc.AdhocReportUnit&quot;) paramMappingsString* – double delimited list of paramMappings (for example, paramMappingsString= &quot;Country@@q**OrderId@@1234&quot;) left – frame left offset top – frame top offset width – frame width height – frame height</td>
<td>Updates the content of a custom content frame. Note: paramMappingsString represents a 2 dimensional array. ** denotes a first level element break; @@ denotes a second level element break.</td>
</tr>
<tr>
<td>replaceFrame</td>
<td>uri* – new resource object’s path name* – name of frame whose content is being replaced; for example, frame_1,</td>
<td>Replaces old content with new.</td>
</tr>
<tr>
<td>moveFrames</td>
<td>paramString – double delimited list of frame names and destination coordinates</td>
<td>Moves a frame. Note: paramString is of the form “name1@@left1@@top1**name2@@left2@@top2” and so on.</td>
</tr>
<tr>
<td>resizeFrames</td>
<td>paramString – double delimited list of frame names, new frame sizes and new font sizes.</td>
<td>Removes the column at the indicated position. Note: paramString is of the form “name1@@width1@@height1@@fontSize1*name2 width2@@height2@@fontSize2” and so on. The new font size can be set to ”-1” when not applicable.</td>
</tr>
</tbody>
</table>
### Table 10-6  Action Codes for Dashboards, continued

<table>
<thead>
<tr>
<th>Action Code</th>
<th>Parameters (* indicates required)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resizeFramesAndRender Controls</code></td>
<td>See <code>resizeFrames</code>, above</td>
<td>Same as <code>resizeFrames</code> but also restreams control HTML to browser.</td>
</tr>
<tr>
<td><code>deleteFrames</code></td>
<td><code>namesParamString</code> – delimited list of names for frames to be deleted.</td>
<td>Deletes specified frames. Note: <code>namesParamString</code> is of the form “name1<strong>name2</strong>name3” and so on.</td>
</tr>
<tr>
<td><code>reset</code></td>
<td><code>layoutLeft</code> – leftOffset of dashboard layout from window&lt;br&gt;<code>layoutTop</code> – topOffset of dashboard layout from window</td>
<td>Used to initialize dashboard settings on refresh or first entry.</td>
</tr>
<tr>
<td><code>setTitle</code></td>
<td><code>title</code> – new title</td>
<td>Updates dashboard title.</td>
</tr>
<tr>
<td><code>toggleScrollBars</code></td>
<td><code>namesParamString</code> – delimited list of names for frames to be toggled.</td>
<td>Toggles the hide/show scrollbar status of specified frames. Note: <code>namesParamString</code> is of the form “name1<strong>name2</strong>name3” and so on.</td>
</tr>
<tr>
<td><code>setScrollBars</code></td>
<td><code>namesParamString</code> – delimited list of names for frames whose scrolling is to be set.&lt;br&gt;<code>showScrollBars</code> – (boolean) indicates whether to show (true) or hide (false) the scroll bars.</td>
<td>Sets the hide/show scrollbar status of specified frames based on the value of <code>showScrollBars</code>. Note: <code>namesParamString</code> is of the form “name1<strong>name2</strong>name3” and so on.</td>
</tr>
<tr>
<td><code>setAutoRefresh</code></td>
<td><code>paramString</code> – double delimited list of frame names and refresh intervals</td>
<td>Sets the specified refresh interval on the specified frames. Note: <code>paramString</code> is of the form “name1@@interval1**name2@@interval2” and so on.</td>
</tr>
<tr>
<td><code>toggleAbsoluteSizing</code></td>
<td>–</td>
<td>Switches between the fixed and proportional sizing behavior.</td>
</tr>
<tr>
<td><code>setLayoutSize</code></td>
<td><code>size</code> – one of the available screen resolutions</td>
<td>Sets the layout size.</td>
</tr>
<tr>
<td><code>addControlFrame</code></td>
<td><code>name</code> – the name of the corresponding input control&lt;br&gt;<code>left</code> – control left offset&lt;br&gt;<code>top</code> – control top offset&lt;br&gt;<code>width</code> – control width&lt;br&gt;<code>height</code> – control height</td>
<td>Adds a new input control.</td>
</tr>
<tr>
<td><code>updateControlValues</code></td>
<td><code>paramString</code> – double delimited list of control frame names, parameter names and new values</td>
<td>Sets the value of the specified control frames to the specified values. Note: <code>paramString</code> is of the form “controlFrameName1@@paramName1 @@paramValue1**controlFrameName2 @@paramName2@@paramValue2 and so on.</td>
</tr>
<tr>
<td><code>resetParamValues</code></td>
<td>–</td>
<td>Resets the value of all input controls to the last saved parameter values.</td>
</tr>
</tbody>
</table>
### Table 10-6  Action Codes for Dashboards, continued

<table>
<thead>
<tr>
<th>Action Code</th>
<th>Parameters (* indicates required)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFreeTextFrame</td>
<td>label – the initial text value</td>
<td>Adds a new Free Text control.</td>
</tr>
<tr>
<td></td>
<td>left – control left offset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>top – control top offset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>width – control width</td>
<td></td>
</tr>
<tr>
<td></td>
<td>height – control height</td>
<td></td>
</tr>
<tr>
<td>addTextLabelFrame</td>
<td>label – the initial text value</td>
<td>Adds a new Text Label.</td>
</tr>
<tr>
<td></td>
<td>left – control left offset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>top – control top offset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>width – control width</td>
<td></td>
</tr>
<tr>
<td></td>
<td>height – control height</td>
<td></td>
</tr>
</tbody>
</table>
| addClickableFrame      | left – frame left offset                                                                                                                                                                                                                 | Adds a new frame (a button by default). The following valid values map to the three types of dashboard button (Submit, Reset, and Print):
|                        | top – frame top offset                                                                                                                                                                                                                 | `idToFunction.put("submit", "paramsUpdated");`
|                        | width – frame width                                                                                                                                                                                                                   | `idToFunction.put("reset", "resetParams");`
|                        | height – frame height                                                                                                                                                                                                                  | `idToFunction.put("print", "printDashboard");`
|                        | id – frame id (for example, id = "submit")                                                                                                                                                                                            |                                                                             |
| setTextLabel           | name – text frame name                                                                                                                                                                                                                 | Sets the label of a free text frame or a text label frame.                 |
|                        | label – text to set as the label                                                                                                                                                                                                       |                                                                             |
|                        | width – new width                                                                                                                                                                                                                     |                                                                             |
| save                   | dbFolder – folder to save in                                                                                                                                                                                                           | Saves the dashboard.                                                        |
|                        | dbnameLabel – name to save with                                                                                                                                                                                                      |                                                                             |
|                        | dbDesc – description                                                                                                                                                                                                                  |                                                                             |
|                        | aruOverwrite – (boolean) whether to overwrite                                                                                                                                                                                           |                                                                             |
CHAPTER 11 CUSTOMIZING THE USER INTERFACE

JasperReports Server is highly customizable because it is built on the Spring Framework and uses web-standards such as Cascading Style Sheets (CSS) and JavaServer Pages (JSP). When the server is embedded in a web application or portal, its user interface (UI) can be customized to extend functionality and better reflect the parent application.

As with any large web application, the logic to generate the JasperReports Server UI is complex and relies on several mechanisms. They are listed here from simplest to most complex.

- Themes – The themes mechanism exposes the CSS of the UI through the repository. This makes it easy for administrators to change the appearance of the UI, such as images, colors, font size, spacing, and even the general layout. The theme mechanism is described in detail in the JasperReports Server Administrator Guide. Refer to that document first.
- SiteMesh – The SiteMesh decorator mechanism creates the header and footer for every page of the server. Decorators provide a quick way to edit the overall appearance of the web app, such as branding and copyright.
- Java Server Pages (JSP) and JavaScript – These are the templates and logic, respectively, that generate the pages of JasperReports Server. Edit these files to change the content of individual pages or the way a page is generated. This chapter assumes you are familiar with JSP and JavaScript syntax.
- Action Model – This mechanism provides a simple way to edit any menu in JasperReports Server. The simple XML syntax lets you remove default menu items, restrict their visibility based on roles, and add new menu items if you have implemented an action.
- Spring MVC (Model, View, and Controller) and Spring Webflow are frameworks for creating states and transitions that represent a business process in a web application. By creating custom flows, you can add your own sequence of pages that integrate with the server.

Themes can be modified in a running server and changes can be seen immediately by all users. For all other types of customization, you need to edit the files that are deployed in the web app. If you are modifying files in the web application, you then need to redeploy the web app in the app server. In some cases, you need to modify source code, in which case you must re-compile the source code and redeploy the web app in the app server.

This chapter includes the following sections:
- Changing the UI With Themes
- Working With Web App Files
- Customizing the Branding with SiteMesh
- Customizing JSP and JavaScript for the Login Page
- Customizing the Report Rendering Page
- Customizing Menus
- Working With Source Code Files
- Adding a Custom JSP Page in a Spring Web Flow
- Adding Custom Export Channels
The information in this chapter applies to JasperReports Server 4.0 and later. For previous versions of the server, refer to the corresponding version of the JasperReports Server Ultimate Guide.

This chapter assumes that you are proficient with CSS and with J2EE application development and configuration. Some of the changes described here are cosmetic, but others affect the core behavior of the server. Use extreme caution when making the described edits, because inadvertent changes might cause JasperReports Server to fail. Jaspersoft recommends that you make all customizations in an isolated test environment before applying them in your production environment.

11.1 Changing the UI With Themes

The themes mechanism uses CSS to render the UI and exposes the CSS for easy modification. The CSS files are stored in the repository and can be downloaded, uploaded, and made active through the UI while the server is running. In fact, the server must be running and you must be logged in as an administrator to modify the themes.

You still need an external editor to modify the contents of the CSS files. For more information about CSS files, see http://www.w3.org/Style/CSS/. This chapter focuses on what CSS needs to change, not the procedures for making changes.

This section walks through several simple changes to the web UI. These modifications take effect when you make your new theme files active. You don’t need to recompile any JasperReports Server source code, or even to restart the server, as is the case in later sections. However, changing the theme only affects the look and feel of the interface, it does not fundamentally alter the functionality of the server.

For more extensive changes, you need to change the JSP files that control the logic of the server and determine what users see, not just how they see it. Those customizations are covered in the other sections of this chapter.

11.1.1 Working with Themes

In commercial editions of JasperReports Server, the hierarchy of organizations, and even the single default organization can inherit themes from the root, or system, theme. Depending on whether you implement organization or how you want to deploy your themes, you may want to implement your theme at the root level or at an organization level. This section focuses on the CSS and image files to modify in a theme, but you may still deploy that theme anywhere you wish.

Also, depending on how you work, you can create theme folders directly in the repository, or you can create them offline and upload them as a ZIP archive file. The procedures in this section do not specify either method, and do not give detailed steps for either method. You need to determine the extent of your customizations to see which method works best for you, and then adapt the procedures for your case.

Finally, several customization can be combined into one theme. Some CSS files may contain values that you override in another file. And through inheritance, the themes defined in one organization can be expressed in all child organizations. You must plan your theme deployment carefully and test it with end-users to ensure it works the way you intend.

For more information about how themes work, see the JasperReports Server Administrator Guide (or the JasperReports Server Community Project User Guide). Familiarize yourself with themes and the themes mechanism in order to create the theme that implements all the customizations you want on the target organization or organizations.

11.1.2 Changing the Logo

One very simple way to customize JasperReports Server is to replace the Jaspersoft logo with your own. Your logo should be approximately the same size as the Jaspersoft logo: 141 pixels wide by 28 pixels high.

There are two ways to change the logo in CSS:

- If your logo is roughly the same size or has the same dimensions as the Jaspersoft logo, replace the logo file.
- If your logo cannot be resized or has different dimensions, change the CSS to load your own logo file.

The Jaspersoft logo is white on a transparent border and is displayed on the blue background of the default theme. Make sure your logo is visible on this background. You can also change the background color to match your logo as well, as explained in section 11.1.4, “Changing Colors and Fonts,” on page 156.
In addition to the logo, there are other customizations you can do to change the branding of the server. See section 11.3.4, “Editing decorators.jsp for Rebranding,” on page 164.

### 11.1.2.1 Replacing the Jaspersoft Logo File

One way to replace the Jaspersoft logo is to create a logo file with the same file name as the Jaspersoft logo that is provided with the server. When you create a theme with this file in the same path as the logo file, it is displayed instead. This is the easiest customization.

**To replace the Jaspersoft logo file with your own:**

1. Create a new theme or modify an existing one.
2. If necessary, add a folder named images to the theme.
3. Convert your logo or image to the PNG format and save it with the filename logo.png.
4. Copy the new logo.png to the images folder.
5. Upload and activate the new theme to the chosen location, then click your browser’s **Refresh** button.

Your logo appears in the top-left corner of every page:

![MyCompany Logo on Home Page](image-url)

### 11.1.2.2 Modifying the CSS File

Another way to replace the logo is to use your own logo file and modify the CSS files in the theme so that they reference it. This creates a new filename and path.

Use this procedure if your logo cannot be resized to fit in the same space, or if the dimensions of your logo have a different ratio. The Jaspersoft logo is 28 pixels high by 141 pixels wide.

**To modify the CSS file to use your own logo file:**

1. Create a new theme or modify an existing one.
   - In this theme, you can place your image file in any folder structure you want.
   - In this example, the logo is a file called MyCompanyLogo.png in a folder called MyImages.
2. If necessary, copy the overrides_custom.css file from the default theme to the main folder of your theme.
3. Edit the overrides_custom.css file and add the following rules. Adjust all pixel values based on the size of your image:

```css
#logo { /* new logo image name and size (example is twice the original logo) */
    background: url("MyImages/MyCompanyLogo.png") 0 0;
    height: 56px;
    width: 282px;
}
#frame { /* moves the main frame a bit down to accommodate bigger logo */
    margin-top: 88px;
}
#globalSearch { /* moves the search box to the right */
    left: 350px;
    top: 32px;
}
```

The IDs and classes that are used in the CSS are dependent on the JSP source code for the server. To find out which IDs and property values are used, you need to look at the JSP files. Alternatively, you can inspect the HTML and CSS returned by the server in a browser tool such as Firebug. For more information, see the JasperReports Server Administrator Guide or the JasperReports Server Community Project User Guide.

4. Upload and activate the new theme with your image and CSS file, then click your browser’s Refresh button.

### 11.1.3 Hiding UI Elements

CSS provides a convenient rule to hide any element of the UI. If you set display:none; as an override, the corresponding element is not rendered in the browser. For example, you can use this parameter to remove the logo from the UI.

Setting the parameter display:none; is a convenient way to hide any element through the CSS. It can be used alone or in conjunction with the other CSS parameters that you might want to keep in the rule.

The hidden element still exists in the HTML transmitted to browsers, so do not use this parameter to hide sensitive information. For example, if you hide a menu item, a user with the FireBug plug-in to Firefox can remove the display:none; attribute to reveal the menu and use it. For a more secure way to remove menu items, see section 11.6.2, “Restricting a Menu Item by Role,” on page 173.

**To remove the logo:**
1. Create a new theme or modify an existing one.
2. If necessary, copy the overrides_custom.css file from the default theme to the main folder of your theme.
3. Edit the overrides_custom.css file and add the following CSS rules:

```css
#logo { display:none;
}
#globalSearch { /* move the search box to the left */
    left: 24px;
}
```

4. Upload and activate the new theme with your image and CSS file, then click your browser’s Refresh button.

### 11.1.4 Changing Colors and Fonts

Changing colors, fonts, size, and spacing throughout the UI is very simple with the themes mechanism. However, creating a complete theme that modifies all aspects of the user interface involves changing many CSS rules and re-creating many images of buttons and window decorations.
To provide an example of theme customizations, the sample data installed with JasperReports Server includes an alternate theme called pods_summer. This theme makes many changes to the default appearance, mostly a new color scheme. You can activate this theme to see its effect, and you can download its files to see how it works.

Like most small-scale customizations, the pods_summer theme uses a single overrides_custom.css file and several image files. The image files have the same names as those in the default theme, so when the theme is activated, they automatically replace the default images.

Just as an example of the CSS involved in modifying a theme, some of the customizations that appear in Figure 11-2 are:

- New logo as described in section 11.1.2.1, “Replacing the Jaspersoft Logo File,” on page 155.
- Horizontal gradient image for the background, along with a different color:

```css
body {
    background:url("images/body_bkgnd.png") repeat-y scroll 0 0 #0ABAEE;
    position:relative;
}
```

- New color for main menu text, a darker green:

```css
.button.action.primary.up,
.menu.primaryNav .up, .menu.primaryNav .wrap,
.tabSet.vertical.buttons .button,
.tabSet.horizontal.buttons .selected > .button > .wrap {
    color:#559502;
}
```

- A new image file for the green button bar used for the main menu
- A new set of images for the search field and button. These images are the same as in the default, but in a different color to match the new background.
Sprites are image files that contain multiple images. The CSS rules that reference them give a vertical and horizontal offset for reading an individual image from the file. In the case of buttons and bars, the sprite contains a long image to accommodate any width, and an end cap to be placed at the desired width. For icon sprites, the images often reflect the various icon states, such as enabled, disabled, mouse-over, and pressed.

When customizing the theme images, having multiple images in one file is helpful because it allows you to work with many related images together in an image editor, instead of dealing with numerous files. For example, you can easily change the color hue for all buttons in a sprite file at the same time.

### 11.1.5 Changing the Spacing in the UI

One advantage of customizing CSS in themes is that you can drastically change the appearance of the UI with very few overrides. CSS controls the spacing and layout of contents on the page, and by changing a few rules, you can rearrange the default layout in many ways.

In the following example, we remove the Jaspersoft logo entirely, then remove the vertical space that it occupies, as well as all borders. As a result, the main menu bar moves up, and we move the search to the right so that it doesn’t overlap the menu items.

1. Create a new theme or modify an existing one.
2. If necessary, copy the overrides_custom.css file from the default theme to the main folder of your theme.
3. Edit the overrides_custom.css file and add the following CSS rules:

```css
#frame {
  margin:10px 0px 7px 0px;
}

#logo {
  display:none;
}

#globalSearch {
  left:380px;
  top:8px;
}

#metaLinks {
  top:13px;
}
```

4. Upload and activate the new theme with your image and CSS file, then click your browser’s Refresh button.

The result is a page that has no blue border, thus giving more screen area to the content:

![Figure 11-4 Theme With Modified Spacing to Remove Borders](image)
There are many CSS attributes that affect spacing, including margins, borders, and edge and corner images. In additions, the various div elements that make up the UI interact with each other, making it hard to set the spacing correctly in some cases. Be sure to look at your overrides on all JasperReports Server pages to verify that the appearance is consistent and correct throughout the UI.

11.1.6 Replacing the Default Theme

In general, Jaspersoft recommends creating new themes with CSS and image files that override those of the default theme. However, there may be some cases where the default theme provided with the server must be replaced.

The <js-webapp>/themes folder contains a copy of the default theme and other sample themes. These files are provided as examples for copying and creating new themes. Even though these theme files appear in the source code and in the deployed web application, they are never used to render the UI.

It is possible to configure the server to load the theme from files, but this disables the dynamic theme mechanism and has other effects. Jaspersoft does not recommend such a configuration.

Like all themes, the default theme is stored in the repository, in the server’s private database. In order for the theme mechanism’s propagation and inheritance to work, the theme files must exist in the repository before the server is started. There are two ways to achieve this:

- Using the import utility to import your theme as the default root theme while the server is stopped (the repository database must be running). In commercial editions that have the organizations architecture, when the server restarts, it propagates your new default theme to all other theme folders.
- During the installation process, there are scripts or manual commands that create the repository database and populate it with the initial contents by performing an import before the server starts. Those initial contents include the default root theme. If you are proficient with buildomatic commands, and database initialization, you can locate the files used to populate the repository and insert your own default theme.

In both cases, you must create a valid repository catalog containing your custom theme in the /Themes/default folder of the catalog. Your catalog must contain the valid XML description for each of the files, for example by exporting a copy of your theme and renaming contents of the exported catalog. The details of changing the default theme through either of these methods is beyond the scope of this document.

Creating a default theme is difficult and prone to error:

- Your version of the default theme must be a complete theme that contains all files named in <js-webapp>/WEB-INF/decorators/decoratorCommonImports.jsp.
- Your default theme must provide rules for rendering all the elements used by the UI, otherwise pages may not render properly.
- Your default theme cannot use the overrides_custom.css file, because themes that override your default theme may include this file. By convention, this file is reserved for non-default themes to override the default theme.
- During an upgrade of the server, your default theme may be overwritten or the new version may not be backwardly compatible with your theme. See section 11.2.4, “Upgrading With UI Customizations,” on page 161.

11.2 Working With Web App Files

This section gives an overview of how components of the user interface are created from the source code. The entire process from designing the UI in source files to displaying the UI in the server is very complex, with several interacting mechanisms. JasperReports Server uses the Spring Framework based on compiled Java beans, but the layout of the UI is also controlled by Java Server Pages (JSP files), the SiteMesh framework that decorates pages, and the CSS files seen in the previous section. Some of these mechanisms use additional XML files for configuration.

There are essentially two kinds of files that are involved in creating the UI:

- Compiled Java files that define the underlying behavior of the Server, for example, what happens when you click a button to run a report. These are Java beans that are used in the Spring framework, where they are also called Spring beans.
In order to change the behavior of a Java file, you must recompile it, rebuild the server, and redeploy it in an application server. Therefore, to change the UI controlled by Java files, you must have the source code distribution and a testing environment to build and deploy the server. For more information, see section 11.7, “Working With Source Code Files,” on page 184.

- Interpreted files such as JSP, XML, and CSS that the server processes in order to generate the UI. The advantage of interpreted files is that you can modify them in a running instance and have them take effect immediately (as with CSS), or after restarting the server (as with JSP). Fortunately, much of the UI can be customized with interpreted files.

### 11.2.1 File Locations

Interpreted files are located in the JasperReports Server web application, known as `<js-webapp>`. Depending on your deployment and your needs, there are several ways to work with the interpreted files, which in turn determine the definition of `<js-webapp>` that you use.

<table>
<thead>
<tr>
<th>File Location</th>
<th><code>&lt;js-webapp&gt;</code> Path</th>
</tr>
</thead>
</table>
| Installed server   | Once you have installed your server, either through a platform installer or any other deployment, the files are deployed in a running application server. The location depends on the application server where you installed JasperReports Server. If you used the bundled Apache Tomcat application server, the files are located in:  
  <js-webapp> = <js-install>/apache-tomcat/webapps/jasperserver[-pro]  
  For other application servers, see Unzipping WAR Files below.  
  After modifying the UI files (except CSS), reload the web app to see the changes. This is the easiest way to customize files, because you can see your changes almost immediately. However, your changes are limited to this one instance of the server. |
| WAR file distribution | When you download the WAR (web archive) file distribution, you can customize your deployment of JasperReports Server and possibly install it on several machines. The WAR file distribution also includes the UI files in the following location:  
  <js-webapp> = <js-install>/jasperserver[-pro].war  
  To modify files with the WAR file, see Unzipping WAR Files below. After modifying the WAR file distribution, you need to deploy it to your application server, as described in the JasperReports Server Installation Guide. But every time you redeploy your modified WAR file, your UI changes are included. |
| Source code        | The JasperReports Server source code also contains the original versions of the interpreted files. If you maintain other customizations in the source code, you can modify the UI files as well. The files in the source code are located in:  
  <js-webapp> = <js-src>/jasperserver/jasperserver-war/src/main/webapp  
  When building the source, these files are copied into the WAR file that you must then deploy into a running application server. See the JasperReports Server Source Build Guide for more information. The advantage of working with the source code is that you can always generate the server with your customized UI files. |

When working with the WAR file distribution or source code, you usually modify files in an installed server for testing. But after testing, you copy the changes into your WAR file or source code.

### 11.2.2 JavaScript and JSP Folders

Inside the JasperReports Server web application, the various JavaScript and JSP files are organized as follows:

- JavaScript files are all in the `<js-webapp>/scripts` folder. File names contain module and subcomponent names, for example `<js-webapp>/scripts/adhoc.chart.js`.
- JSP files are split up under the `<js-webapp>/WEB-INF/jsp` folder:
  - The `<js-webapp>/WEB-INF/jsp/templates` folder contains UI components such as panels, lists, menu, and dialogs.
  - The `<js-webapp>/WEB-INF/jsp/modules` folder contains flow-specific pages:
• The main layout and subpanel JSP files are directly in modules.
• Each feature has subfolders for its various JSPs.

### 11.2.3 Unzipping WAR Files

When modifying UI files in application servers other than Apache Tomcat or in the WAR file distribution, the web application is kept as a single WAR (web archive) file. To modify files inside the WAR file, you must extract, modify and replace the files. The following example shows one way to do this from the Windows command line (the commands are similar in Linux).

In this example, `<path/filename>` refers to the relative path and name of the file to modify within the WAR file:

```bash
cd <js-webapp>
"%JAVA_HOME%\bin\jar" xf jasperserver[-pro].war <path/filename>
<edit> <path/filename>
"%JAVA_HOME%\bin\jar" uf jasperserver[-pro].war <path/filename>
delete <path/filename>
```

After modifying any files in the running application or reloading the web application, you may need to perform the following steps, depending on your application server:

1. Clear the application server's work folder. In the case of Apache Tomcat, you would delete all files and folders in the `<tomcat>/work` folder.
2. Click Refresh on your browser.
3. In some cases, you may need to restart the application server.

### 11.2.4 Upgrading With UI Customizations

One consideration when customizing the user interface is how those customizations can be maintained during upgrades to JasperReports Server. As explained in section 11.2.1, “File Locations,” on page 160, where and how you make changes affects how easy it is to maintain them.

- Changes to the CSS files and images of a theme are stored in the repository, which is preserved through updates. However, the CSS in the new version may refer to different IDs and classes, or to image files with different names. Also, the overrides in an old theme may not have the desired appearance when applied to a new theme.
- Because the server is distributed as a WAR file, changes to files in an installed version or an existing WAR file are likely to be overwritten during an upgrade. Sometimes, you can save a copy of the modified files and redo the changes in the new installation or WAR file. In this case, you must be careful to copy only your modifications from the saved file into the new server files. In other cases, the existing modifications no longer apply to the files or mechanisms in the new server.
- Keeping your changes in a copy of the source code lets you take advantage of source control to update and merge your changes into the code for the new server. However, in the case of major releases UI mechanisms sometimes change, and the files with your modifications may no longer exist or be relevant in the new code.

In general, new releases of the server attempt to provide backward compatibility of the mechanisms described in this chapter, especially minor (“dot”) releases. However, in order to improve the UI and provide new features, Jaspersoft cannot guarantee that all customizations still apply, or even that they will use the same mechanism, especially in major releases.

### 11.3 Customizing the Branding with SiteMesh

While themes allow you to modify CSS files and images, there are some parts of the UI that are controlled by other mechanisms. In particular, JasperReports Server uses the SiteMesh framework to lay out and decorate nearly every page. The decoration is the HTML for the headers and footers that are nearly identical on every page.
The SiteMesh framework is controlled by the following files:

- `<js-webapp>/WEB-INF/web.xml`
- `<js-webapp>/WEB-INF/sitemesh.xml`
- `<js-webapp>/WEB-INF/decorators.xml`
- `<js-webapp>/WEB-INF/decorators/main.jsp`
- `<js-webapp>/WEB-INF/decorators/decorators.jsp`

Essentially, the XML files specify how UI pages should be generated, and the JSP files generate the pages. The following sections describe these files and how to customize the JSPs to change the overall branding that appears in the UI.

### 11.3.1 web.xml

The `<js-webapp>/WEB-INF/web.xml` configuration file contains the configuration information that enables SiteMesh. Here, we see that SiteMesh's `PageFilter` class is applied to all targeted URLs (that is, `<url-pattern>/*</url-pattern>`):

```xml
<filter>
  <filter-name>sitemesh</filter-name>
  <filter-class>com.opensymphony.module.sitemesh.filter.PageFilter</filter-class>
</filter>
... 
<filter-mapping>
  <filter-name>sitemesh</filter-name>
  <url-pattern>/*</url-pattern>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

### 11.3.2 sitemesh.xml and decorators.xml

The SiteMesh page filter assumes that the `<js-webapp>/WEB-INF/sitemesh.xml` file specifies further configurations. We see that the main decorator’s definition points to `decorators.xml`. In addition, we see the SiteMesh mapping that handles the default locale (U. S. English):

```xml
<property name="decorators-file" value="/WEB-INF/decorators.xml"/>

<!-- Mapper for localization -->
<mapper class="com.opensymphony.module.sitemesh.mapper.LanguageDecoratorMapper">
  <param name="match.en" value="en"/>
  ...
</mapper>
```
Next, we look at the `<js-webapp>/WEB-INF/decorators.xml` file. First, it defines URL patterns that SiteMesh should skip. Then, it defines the main decorator JSP page that is used by JasperReports Server:

```
<excludes>
  <pattern>*adhoc/crosstab*</pattern>
  <pattern>*adhoc/table*</pattern>
  ...
</excludes>

<decorator name="main" page="main.jsp">
  <pattern>/*</pattern>
</decorator>
```

For more detailed information about SiteMesh and decorators, see [http://www.opensymphony.com/sitemesh/decorators.html](http://www.opensymphony.com/sitemesh/decorators.html).

### 11.3.3 main.jsp and decorator.jsp

In `<js-webapp>/WEB-INF/decorators/`, the `main.jsp` includes the `decorator.jsp` file, and together they set the appearance and layout of the JasperReports Server web interface. In particular, `decorator.jsp` specifies all the display elements that appear as the header and footer of every JasperReports Server page. Inside the header and footer and main frame is the `<decorator:body/>` tag that specifies where to add the HTML content generated for the target page.

In the following listing of `decorator.jsp`, you can see the structure of every JasperReports Server HTML page, with the main frame, the header, the body content, and the footer:

```
<html>
  <head>
    <title>Jaspersoft: <decorator:title /></title>
    ...
    <decorator:head />
  </head>

  <body id="<decorator:getProperty property='body.id'/>"
    class="<decorator:getProperty property='body.class'/>">
    <div id="frame" class="column decorated">
      ...
      <div class="content">
        <div class="header">
          ...
        </div><!--/#frame .header -->
        <!-- START decorated page content-->
        <decorator:body />
        <!-- END decorated page content -->
      </div><!--/#frame .content -->
    </div><!--/#frame -->
  </body>
</html>
```
For example, if the user clicks View > Reports, the WEB-INF/jsp/modules/ListReports.jsp is executed. The ListReports.jsp generates the HTML content. Before this content is emitted, the SiteMesh page filter inserts the content into the location specified by <decorator:body/>. Then, the whole HTML content is sent to the user’s browser.

11.3.4 Editing decorators.jsp for Rebranding

Now that we see how decorators.jsp defines the main page of the server UI, we can customize the file. If you use JasperReports server as part of your suite of business applications, you may not want the Jaspersoft branding on the page. Editing the decorators.jsp file in the deployed webapp lets you remove the branding. The following elements make up the Jaspersoft branding:

- The company logo: to change or remove the logo, see section 11.1.2, “Changing the Logo,” on page 154.
- The browser icon (favicon).
- The page title appearing in the browser.
- The About link and copyright footer on every page.

To replace the browser icon (favicon):
1. Create a 16 by16 pixel icon file named favicon.ico.
2. Copy the icon to the following location:
   Instance:  <js-install>/apache-tomcat/webapps/jasperserver[-pro]
   Source Code: <js-src>/jasperserver/jasperserver-war/src/main/webapp
3. Restart the server.

To edit the page title:
1. Edit the file <js-webapp>/WEB-INF/decorators/decorator.jsp.
2. Change the title text, for example:

   <html>
   <head>
   <title>My Company: <decorator:title /></title>
   ... 
   <decorator:head />
   </head>
   ...

3. After saving your changes to the JSP file, restart your application server or reload the JasperReports Server web app.

To remove the footer text:
1. Edit the file <js-webapp>/WEB-INF/decorators/decorator.jsp.
2. Change the footer text, for example to comment out the about and copyright lines:

```html
... 
<div id="footer" class="footer">
<!-- <a id="about" href="#"><spring:message code="decorator.aboutLink"/></a>
<p id="copyright"><spring:message code="decorators.main.copyright"/></p> -->
</div> ...
```

3. After saving your changes to the JSP file, restart your application server or reload the JasperReports Server web app.

### 11.4 Customizing JSP and JavaScript for the Login Page

If you want to replace the Jaspersoft branding, the login page requires extensive changes. Rebranding with a new theme and with the SiteMesh decorators are necessary but not sufficient. To completely customize the login page, you also need to edit the following files that define the content on the page:

- Favicon and page title can be changed through the SiteMesh decorators, as described in section 11.3.4, “Editing decorators.jsp for Rebranding,” on page 164.
- There are several specific CSS rules for the login page that you may customize in a theme.
- The content of the login page is located in a JSP file that we can edit to remove sections of the login page.
- However, the URL for a button we want to keep is located in a JavaScript page.
- All strings on the login page are stored in properties files. Most of these don’t need to change, but several do mention Jaspersoft.

The following example shows how the CSS, JSP, JavaScript, and properties files all need to be modified together to make a uniform change such as removing the branding from the login page.

**To change the branding on the login page:**

1. Create a new theme or modify an existing one.
2. If necessary, copy the overrides_custom.css file from the default theme to the main folder of your theme.
3. Edit the overrides_custom.css file and add the following CSS rules:

```css
#loginPage #loginForm { /* where users enter their ID and password */
  position: absolute;
  left: 0px;
  top: 0;
}
#loginPage #copy { /* information panel with the image and info buttons */
  float: right;
  clear: right;
  width:700px;
  margin-right: 10px;
}
#loginPage #buttons {
  border-bottom: none;
  margin-bottom: 0;
}
```

In this example, we swap the login form and information panel left and right, respectively. You could also change the size and vertical position of the panels.
4. If you want to change the image on the login page, create an images folder if necessary, and save your image in the JPG format with the following name. Your image should be the same size, approximately 670 pixels wide and 283 pixels high, otherwise you may need to adjust the spacing on the login page through the theme as well:

Community Project: images/login_welcome_ce_bkgd.jpg  
Commercial Editions: images/login_welcome_bkgd.jpg

In this example, we replace the image with an image of the fictional MyCompany logo.

5. Upload and activate the new theme with your image and CSS file, then click your browser’s Refresh button.

In commercial editions with the organizations architecture, you must upload and activate the theme at the root level. The theme at the root level applies to the login page for all users, regardless of the users’ organization. You must login as system admin (superuser) to set the theme at the root level.

7. Remove the following lines by commenting them out:

```html
<%-- <div id="rotating" class="row">
    <jsp:include
        page="rotating/login_rotating_${jsEditionClass}_${randomRotatingPageNumber}.jsp"/>
    </div>
--%>
```

8. Edit the file <js-webapp>/scripts/components.loginBox.js.
9. Change the URL of the CONTACT_SALES_URL in the loginBox variable definition from the jaspersoft.com address to another address:

```javascript
var loginBox = {
    LOGIN_BOX_TEMPLATE_DOM_ID: "login",
    ...  
    CONTACT_SALES_URL: "http://example.com/contact-us",
    ...
```

10. Edit the file <js-webapp>/WEB-INF/bundles/jasperserver_messages.properties.
11. Change the following strings to customize your login page:

```properties
#Welcome Login Page  
LOGIN_WELCOME_OS=Welcome to MyCompany  
...  
BUTTON_CONTACT_SALES=Contact MyCompany
```

12. If you support multiple locales, modify the same message keys in the other language bundles:

    jasperserver_messages_de.properties  
    jasperserver_messages_es.properties  
    jasperserver_messages_fr.properties  
    jasperserver_messages JA.properties  
    jasperserver_messages zh_CN.properties

13. Restart the app server or reload the JasperReports Server web app as shown in the following section.
14. When combining these customizations with those in section 11.3, “Customizing the Branding with SiteMesh,” on page 161, the login page has a new appearance that removes all Jaspersoft branding, as shown in the following figure:

![Login Page Custom Layout and Rebranding](image)

**Figure 11-5 Custom Layout and Rebranding of Login Page**

### 11.4.1 Reloading the JasperReports Server Web App

When you customize interpreted files such as the JSP, JavaScript, and properties files, you need to reload them in the JasperReports Server web application to take effect. The standard installation of the server only includes shortcut actions for starting and stopping the bundled application server and database server. If you have other web applications that you don’t want to stop, or if you are doing a lot of customization and testing, it is simpler and quicker to simply reload the web app without restarting the app server.

Each different app server has its own management console that lets you view and control the web apps that are deployed. This example shows how to manage the JasperReports Server web app in the bundled Apache Tomcat.

1. If you have not configured any users on your Apache Tomcat server, you must first add a user and give him the manager role. Edit the file `<js-install>/apache-tomcat/conf/tomcat-users.xml`.

2. Create a user if necessary and give him the manager role. You can give the user any name and password you prefer.

   ```xml
   <tomcat-users>
   
   <user username="tomcat" password="tomcat" roles="manager"/>
   
   </tomcat-users>
   ```

3. Open the Apache Tomcat Manager page in a browser: http://<host>:/<port>/manager/html/, where <host> and <port> are where you installed the server. For a default installation, use http://localhost:8080/manager/html/.

4. When prompted, log in with the user credentials you created in step 2.

5. On the management interface, scroll down to find the jasperserver[-pro], and click Reload.

6. After you confirm you want to reload the web app, wait until the page reloads, then open the JasperReports Server login page to see your changes. If the manager doesn’t reload, there was likely an error in the files, and you must then stop and restart the app server.
11.5 Customizing the Report Rendering Page

The report viewer is the module of the JasperReports Server UI that displays JasperReports. It is a central part of the UI that users interact with frequently. In addition to displaying the report contents, the report viewer offers the following functionality:

- Display input controls so that users may set report options, also called parameters.
- Let the user navigate through the pages of the report.
- Allow the user to export the report in various formats.

Each of these is customizable, and in particular, the input controls and report viewing page can be set either globally or on an individual report basis. This allows you to control many details of the report viewing experience for your users. Adding new export formats is a more advanced customization covered in section 11.9, “Adding Custom Export Channels,” on page 188.

11.5.1 Customizing the Input Controls Form

The input controls are displayed in a form that gives the label of each form and the field or widget for the user to enter a value. The input fields and widgets, such as drop-down menus, radio buttons, and check boxes are determined by the input controls themselves, but their layout can be customized with a JSP file. For general information about input controls, refer to the JasperReports Server Administrator Guide.

JasperReports Server can display the input controls form in a pop-up window (the default), in a separate page, in a separate column, or at the top of the report page. Each kind of layout is determined by a JSP file that renders the input control page.

<table>
<thead>
<tr>
<th>Layout Type</th>
<th>Files To Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pop-up window</td>
<td>&lt;js-webapp&gt;/WEB-INF/jsp/modules/inputControls/DefaultParametersForm.jsp</td>
</tr>
<tr>
<td>Separate page</td>
<td>&lt;js-webapp&gt;/WEB-INF/jsp/templates/inputControls.jsp</td>
</tr>
<tr>
<td>In page (left-hand panel)</td>
<td></td>
</tr>
</tbody>
</table>

If you modify the files listed above and save them in your web application, your changes apply to all reports. The procedure would be similar to section 11.4, “Customizing JSP and JavaScript for the Login Page,” on page 165. After making your changes and redeploying the web app, every user would see your new input controls form.

In the case of the file DefaultParametersForm.jsp, you can save a copy with a different name and you can use it in individual reports, without affecting the default appearance of input controls for other reports. For example, you could create a custom JSP that changed the alignment of the input control labels and rendered them in bold against a yellow background.
Customizing the User Interface

To use this custom JSP in a report, save it as `<js-webapp>/WEB-INF/jsp/custom/MyParametersForm.jsp` in your server instance. Then edit the JasperReport resource, and on the Controls and Resources page, set the **Optional JSP location** to the relative path of your file, in this case `custom/MyParametersForm.jsp`.

For information about editing JasperReports in the repository, refer to the *JasperReports Server User Guide*. 
11.5.2 Customizing the Report Viewer

The report viewer creates the page with the controls for paging through a report and exporting its contents in other formats. The main JSP files of the report viewer are:

<js-webapp>/WEB-INF/jsp/modules/viewReport/ViewReport.jsp
<js-webapp>/WEB-INF/jsp/modules/viewReport/DefaultJasperViewer.jsp

As with input controls, you have two options:

- Modify and save the default files without changing their names. The procedure would be similar to section 11.4, “Customizing JSP and JavaScript for the Login Page,” on page 165. After redeploying the web app, your changes to the report view appear in every report for every user.

- Make a copy of one or more files and save them with a different name. Customize your copy of each file to produce the layout you prefer. Then change the custom JSP setting in the JasperReport resource.

The following figure shows the default layout of the report viewer:

![Figure 11-8 Default Appearance of the Report Viewer](image)

And the following figure shows a possible customization. For example, you may want to restrict the export formats, but make them more accessible. This example also increases vertical space by moving the pagination buttons to the tool bar.
Customizing the User Interface

For example, if you created this layout in a copy of `<js-webapp>/WEB-INF/jsp/modules/viewReport/DefaultJasperViewer.jsp` you would save it in a file named `<js-webapp>/WEB-INF/jsp/custom/MyJasperViewer.jsp`. Then, in the JasperReports that you want to use this viewer, you would specify your file on the Edit JasperReport > Customization page as shown in Figure 11-10.

For information about editing JasperReports in the repository, refer to the *JasperReports Server User Guide*.

### 11.6 Customizing Menus

A very common customization is the removal, addition, or restriction of access to the main menu of JasperReports Server. Section 11.1.3, “Hiding UI Elements,” on page 156 showed a limited way of removing access to the main menu items by simply hiding the menu elements with CSS in a theme. Customizing the menu structure is more robust, more secure, and lets you fine tune the functionality available to users. In particular, you can restrict access to menus based on roles, so that users see different menu choices depending on the roles they belong to.

The mechanism that implements the main menu is called the actionModel, and it is defined in a set of actionModel-*.xml files in the `<js-webapp>/WEB-INF/` folder. The action model is a way to represent menus, sub-menus, and menu items in XML,
giving each item an action when selected, as well as optional role-based restriction. The actionModel is also used to define context menus on folders and resources listed on repository browse and search pages.

The actionModel represents the structure of the menus through the structure of the XML. When pages are processed, the actionModel mechanism converts the XML into the JavaScript that generates menus.

### 11.6.1 Removing a Menu Item

In this example, suppose that neither users nor administrators have been trained to work with Domains, and to prevent users from accidentally creating Domain resources, system administrators decide to remove any reference to Domains in the user interface.

**To remove the Domain menu items:**

1. Edit the file `<js-webapp>/WEB-INF/actionModel-navigation.xml`. This file defines the main menu visible by default on all pages of the server. In the XML that defines the actionModel, there are many condition tests that determine when each menu and menu item should be displayed.

   ```xml
   ...
   <context name="main_create_mutton" test="isProVersion">
     <condition test="!banUserRole">
       <selectAction labelKey="NAV_005_CREATE">
         <condition test="isAvailableProFeature" testArgs="AHD">
           <option labelKey="NAV_051_ADHOC_REPORT"
               action="primaryNavModule.navigationOption"
               actionArgs="designer"/>
         </condition>
         <condition test="isAvailableProFeature" testArgs="DB">
           <option labelKey="NAV_050_DASHBOARD"
               action="primaryNavModule.navigationOption"
               actionArgs="dashboard"/>
         </condition>
       </selectAction>
       <!--
       <condition test="isAvailableProFeature" testArgs="AHD">
         <condition test="checkAuthenticationRoles" testArgs="ROLE_ADMINISTRATOR">
           <option labelKey="NAV_056_DOMAIN"
               action="primaryNavModule.navigationOption"
               actionArgs="domain"/>
         </condition>
       </condition>
       -->
       </condition>
     </condition>
   </context>
   ...
   ``

   "Mutton" is a term that means menu-button, and designates a button that creates a drop-down menu.

2. Find the section at the end for the Create menu and insert comments to remove the last menu item, as shown in the code sample above.

3. Edit the file `<js-webapp>/WEB-INF/actionModel-search.xml`. This file defines the context menu items visible when right-clicking on folders and resources in repository listings.
4. Find the section at the top of the file that creates the Add Resources sub-menu. Then add comments to remove the lines that define the Add Resources > Domain menu item, as shown in the code sample above.

5. Save the modified files and reload the web app in the app server to see the changes (see 11.4.1, “Reloading the JasperReports Server Web App,” on page 167).

6. When the web app has reloaded, log into JasperReports Server as jasperadmin. Click on the Create menu and right-click on a folder in the repository. In both cases, the menu item for Domain is no longer available.

When you remove a menu item, it is removed for all users, even administrators. Often it is preferable to prevent only non-administrators from viewing a menu item, as shown in the next section.

11.6.2 Restricting a Menu Item by Role

In this new example scenario, users haven’t had training in creating reports with the Ad Hoc designer. We customize the UI to restrict the Ad Hoc creation menu to administrators. We must also remove the Create a Report button from the users’ home page. To do this we use a special syntax to conditionalize JSP content so only administrators can see the button.

1. Edit the file `<js-webapp>/WEB-INF/actionModel-navigation.xml`. The actionModel for the Create > Ad Hoc Report is near the end of the file.
2. Following the pattern of conditions for other administrator-only functionality, insert the \texttt{condition} tag for checking role authentication around the \texttt{option} tag to display the Ad Hoc menu item, as shown in the code sample above.

In commercial editions, you must specify the role’s organization ID when restricting access to roles defined in an organization. There are three ways to specify a role in the commercial edition:

- \texttt{ORG\_ROLE|orgID} – Explicitly specify a role belonging to an organization.
- \texttt{ORG\_ROLE|*} – Match the role name in any organization in the user’s scope (parent organizations and root).
- \texttt{SYSTEM\_ROLE} – Explicitly specify a role defined at the root or system level, such as \texttt{ROLE\_ADMINISTRATOR}.

If you want to hide an entire menu, follow the pattern of the Manage menu, which is hidden from non-administrators. In this case, add the \texttt{test} and \texttt{testArgs} attributes to the \texttt{context} tag that displays the menu, as shown in the following sample:

```
<!-- The Manage menu is displayed only to administrators -->
<context name="main_manage_mutton" test="checkAuthenticationRoles" testArgs="ROLE_ADMINISTRATOR">
  <selectAction labelKey="menu.administration">
  </selectAction>
</context>
```

Creating an Ad Hoc report is also available through the large buttons on the home page of every user. The home page buttons are generated in traditional Java Server Pages, outside of the \texttt{actionModel} for menus.
The folder `<js-webapp>/WEB-INF/jsp/modules/home` includes the file homeForNonDemo.jsp that displays the home-page buttons. Open this file for editing.

```html
<div class="row">
  <a id="viewReports" class="button action jumbo up"><span class="wrap"><spring:message code="home.view" javaScriptEscape="true"/></span><span class="icon"></span></a>
  <% if (LicenseManager.isAdHocFeatureSupported()) { %>
    <authz:authorize ifAllGranted="ROLE_ADMINISTRATOR">
      <a id="createReports" class="button action jumbo up"><span class="wrap"><spring:message code="home.create" javaScriptEscape="true"/></span><span class="icon"></span></a>
    </authz:authorize>
  <% } %>
  <% if (LicenseManager.isAnalysisFeatureSupported()) { %>
    <a id="analyzeResults" class="button action jumbo up"><span class="wrap"><spring:message code="home.analyse" javaScriptEscape="true"/></span><span class="icon"></span></a>
  <% } %>
</div>
```

3. Following the pattern of conditions for other administrator-only functionality, insert the `authz:authorize` tag for checking role authentication before displaying the button, as shown in the code sample above.

   In commercial editions, you must use the `js:authorize` tag and specify the role's organization ID when restricting access to roles defined in an organization. For example:

   ```html
   <js:authorize ifAllGranted="ORG_ROLE\|OrgID">...</js:authorize>
   
   You can use either `ORG_ROLE\|OrgID`, `ORG_ROLE\|*`, or `SYSTEM_ROLE` syntax described in step 2.
   ```

4. Save the modified files and reload the web app in the app server to see the changes (see 11.4.1, “Reloading the JasperReports Server Web App,” on page 167).

5. When the web app has reloaded, log into JasperReports Server as `joeuser`. You can see that the button for creating a report is removed, and there is no `Create > Create Ad Hoc Report` menu item. Log out and log back in as `jasperadmin`. Both the button and the menu item are visible to administrators.

Be very careful when editing the JSP or XML files that define the UI. Simple typos or bugs such as unclosed tags can cause the server to appear in an incorrect state, or make it impossible to log in. After fixing the problem, you may need to restart the app server; reloading the web app doesn’t always resolve the issue.

### 11.6.3 Adding an Item to the Main Menu

Adding menu items involves three files:

- The `actionModel` file for the menu item.
- The properties file that labels the menu item.
- An action file that handles events for the menu item.

In this example, we add a special menu item so that MyCompany employees can easily find their accounts reports.
1. Edit the file `<js-webapp>/WEB-INF/actionModel-navigation.xml`. Locate the actionModel for the View menu near the beginning of the file.

   ```xml
   <!--context for view option on primary menu-->
   <context name="main_view_mutton" test="!banUserRole">
     <selectAction labelKey="menu.repository">
       <option labelKey="menu.search" action="primaryNavModule.navigationOption"
            actionArgs="search"/>
       <option labelKey="menu.all" action="primaryNavModule.navigationOption"
            actionArgs="library"/>
       <option labelKey="menu.reports" action="primaryNavModule.navigationOption"
            actionArgs="report"/>
       ...
     <separator/>
       <option labelKey="NAV_028_ACCOUNTS" action="primaryNavModule.navigationOption"
            actionArgs="accounts"/>
   </selectAction>
   </context>
   ...
   ```

2. Add a separator tag and an option tag for the menu item. The option tag has attributes to specify the label key and the name of the action we want to perform.

3. Open one of these files. The name of the properties file depends on your edition of JasperReports Server:

   Commercial Editions: `<js-webapp>/WEB-INF/bundles/pro_nav_messages.properties`
   Community Project: `<js-webapp>/WEB-INF/bundles/jasperserver_messages.properties`

   The names of the keys are slightly different depending on the file. The following example shows the contents of the commercial edition `pro_nav_messages.properties`.

   ```properties
   NAV_001_HOME=Home
   NAV_002_VIEW=View
   NAV_003_MANAGE=Manage
   NAV_004_LOGOUT=Log Out
   NAV_005_CREATE=Create
   NAV_020_TITLE=JasperReports Server
   NAV_020_FULL_TITLE=JasperReports Server
   NAV_021_TITLE_MODIFIER=professional
   NAV_021_LIBRARY=Repository
   NAV_022_REPORTS=Reports
   NAV_023_OLAP_VIEWS=OLAP Views
   NAV_024_EVENTS=Messages
   NAV_025_ADHOC=Ad Hoc Report
   NAV_026_REPOSITORY_EXPLORER=Repository Explorer
   NAV_027_SEARCH=Search Results
   NAV_028_ACCOUNTS=MyCompany Accounts
   ...
   ```

4. Add the line for the `NAV_028_ACCOUNTS` property with an appropriate value, in this case `MyCompany Accounts`.

5. If you support multiple locales, add the same message key to the other language bundles.

```javascript
var primaryNavModule = {

    NAVIGATION_MENU_CLASS : "menu vertical dropDown",
    ACTION_MODEL_TAG : "navigationActionModel",
    CONTEXT_POSTFIX : "_mutton",
    NAVIGATION_MUTTON_DOM_ID : "navigation_mutton",
    NAVIGATION_MENU_PARENT_DOM_ID : "navigationOptions",
    JSON : null,

    /**
     * Navigation paths used in the navigation menu
     */
    navigationPaths : {
        library : {url : "flow.html", params : "_flowId=searchFlow"},
        home : {url : "home.html"},
        logOut : {url : "exituser.html"},
        search : {url : "flow.html", params : "_flowId=searchFlow&mode=search"},
        report : {url : "flow.html", params : "_flowId=searchFlow&mode=search&
            filterId=resourceTypeFilter&filterOption=resourceTypeFilter-reports&
            searchText=""},
        accounts : {url : "flow.html", params : "_flowId=searchFlow&mode=search&
            filterId=resourceTypeFilter&filterOption=resourceTypeFilter-reports&
            searchText=account"},
        ...
    }
}
```

7. Add a URL and parameters to load the desired page in response to selecting the new accounts menu item. In the case of this example, the action is to perform a search for all reports with the word `account` in their name or description.

8. Save the modified files and reload the web app in the app server to see the changes (see 11.4.1, “Reloading the JasperReports Server Web App,” on page 167).

9. When the web app has reloaded, log into JasperReports Server as `joeuser`. You can see the View > MyCompany Accounts menu item was created and selecting it performs a customized search. The following figure shows both the menu item and the search results on the same page.

![Figure 11-11 Custom Menu Items and Corresponding Action](image)
11.6.4 Adding a New Main Menu

The following example shows two useful variants when adding custom menu items:

- You can add your own menus to the main menu bar.
- Menu items, added to either existing or custom menus, can link outside of the server.

This example creates a new menu named Accounts. It contains the internal search item we created in the previous example and an external search with Google.

1. Edit the file `<js-webapp>/WEB-INF/actionModel-navigation.xml`. Add the menu definition with two menu items to the end of the file.

   ```xml
   ...<context name="main_custom_mutton" test="!banUserRole">
   <selectAction labelKey="NAV_801_ACCOUNTS">
     <option labelKey="NAV_802_SEARCH_IN"
           action="primaryNavModule.navigationOption"
           actionArgs="accounts"/>
     <option labelKey="NAV_803_SEARCH_OUT"
           action="externalSearchHandler"
           actionArgs="MyCompany+account"/>
   </selectAction>
   </context>
   ...```

   The two menu items have a label and an action defined, but no conditions. All users can access these menu items.

   The internal search item is the same as the MyCompany Accounts example above. The external search item needs to have its own action that we’ll define in another file.

2. Open the properties file for your edition of JasperReports Server:

   Commercial Editions: `<js-webapp>/WEB-INF/bundles/pro_nav_messages.properties`
   Community Project: `<js-webapp>/WEB-INF/bundles/jasperserver_messages.properties`

3. Add three simple labels, one for each key defined in the actionModel file.

   ```properties
   NAV_801_ACCOUNTS=Accounts
   NAV_802_SEARCH_IN=Accounts Reports
   NAV_803_SEARCH_OUT=Google MyCompany Accounts
   ```

4. If you support multiple locales, add the same message key to the other language bundles.

5. Edit the file `<js-webapp>/scripts/actionModel.primaryNavigation.js`. Add the accounts navigation path if you didn’t do so in the previous procedure:

   ```javascript
   navigationPaths : {
     ...accounts : {url : "flow.html", params : "_flowId=searchFlow&mode=search&
                        filterId=resourceTypeFilter&filterOption=resourceTypeFilter-reports&
                        searchText=account"},
     ...```

   The accounts navigation path used in the previous example is the same as the one used in this example. It performs a search of reports in the repository containing the word `account` in their name or description.
6. In the same `<js-webapp>/scripts/actionModel.primaryNavigation.js` file, add the external search handler function to the end of the file, outside of all other definitions:

```javascript
var externalSearchHandler = function(qstring) {
    window.location.href = "http://google.com/search?q=" + qstring;
};
```

7. Save the modified files and reload the web app in the app server to see the changes (see 11.4.1, “Reloading the JasperReports Server Web App,” on page 167).

8. When the web app has reloaded, log into JasperReports Server as `joeuser`. You can see the new Accounts menu with Accounts > Google MyCompany accounts opening a Google search. The following figure shows both the new menu and the external search results.

![Figure 11-12 Creating a Custom Menu and an External Link](image)

The menu shown on the left is the one seen by `joeuser`. The actionModel automatically creates and places the new menu and its menu items. For `jasperadmin` who also sees the Manage menu, the Accounts menu would overlap the search field. When creating custom menus whose visibility is based on roles, be sure to test different users to see the different UI layouts.

You may need to adjust other parts of the UI, such as the placement of the search field. In this case you could use the Themes mechanism as described in section 11.1.5, “Changing the Spacing in the UI,” on page 158 to avoid overlap with the search field.

### 11.6.5 Changing Other Menus

The context menus available on folder and resources when browsing or searching the repository can also be customized. They use the same action model mechanism that you can customize as shown in the preceding sections. Context menus are defined in the file `<js-webapp>/WEB-INF/actionModel-search.xml`.

Menus for the Dashboard Designer are defined in the file `<js-webapp>/WEB-INF/actionModel-dashboard.xml`.

You can customize the menus in the Ad Hoc Editor in the following files:
- `<js-webapp>/WEB-INF/actionModel-adhocChart.xml`
- `<js-webapp>/WEB-INF/actionModel-adhocCrosstab.xml`
- `<js-webapp>/WEB-INF/actionModel-adhocOlapCrosstab.xml`
- `<js-webapp>/WEB-INF/actionModel-adhocTable.xml`

### 11.6.6 Action Model Reference

The action model is a complex mechanism for generating menus dynamically. In particular, menus in Ad Hoc must be generated programmatically based on the contents of the reports, for example the context menu on a column. The following high-level steps explain how menus are generated:

When generating menus, the Ad Hoc Editor follows these steps:

1. Whenever a page with menus is to be displayed for the first time, the server looks up the corresponding action model XML definition and use JDOM (Java-based Document Object Model for XML) to build a Document that gets cached.
2. Subsequent viewings reference the cached Document.
3. After the page is modified (usually triggered by an Ajax Request) the server generates a client side action model in the form of a JSON expression. Based on the current page state, the client model is a filtered version of the full action model in which internationalized names and generated options are resolved, and so forth.
4. Every time a menu is requested on the client, the server looks up the context in the JSON model, and for each action, clones the appropriate HTML template for the menu and tweaks its attributes accordingly.

Some pages have mostly static menus that don’t change based on page contents. Other pages, such as the Ad Hoc Editor, have dynamic menus that are updated in this way in response to changes the user makes on a page.

The following sections document the XML elements used the action model definition files. In particular, you can define various conditions at several levels so that menus only appear to certain users or based on the current state of the page.

11.6.6.1 Context

Each context represents a distinct menu type and refers to the part of the design page that launches that menu. Examples are reportLevel, columnLevel, fieldLevel. They are directly equivalent to the menu levels defined in the popup-style JSP files used in previous releases.

11.6.6.2 Condition

A condition element invokes the specified server side test as a method on the view model. The enclosed actions are only included in the client action model if the test returns true.

If the test has a leading exclamation point (!), the condition tests for false:

- attributestest — The name of the java method to be invoked on the view model.
- testargs — Array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant, which is used as a delimiter.

11.6.6.3 Actions

Each defined action produces a row or rows in the generated menu. The following tables describe the several action types.

<table>
<thead>
<tr>
<th>simpleAction</th>
</tr>
</thead>
<tbody>
<tr>
<td>A standalone menu action that, when clicked, fires the specified JavaScript method.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>id</strong></td>
</tr>
<tr>
<td>The ID of the menu row DOM object.</td>
</tr>
<tr>
<td><strong>disabled</strong></td>
</tr>
<tr>
<td>If set to true, disable this row initially (shows a grey block instead).</td>
</tr>
<tr>
<td><strong>labelKey</strong></td>
</tr>
<tr>
<td>The text the menu should display. If it corresponds to an localization bundle key it is translated, otherwise it is displayed as is.</td>
</tr>
<tr>
<td><strong>labelCondition</strong></td>
</tr>
<tr>
<td>Sometimes the label value is contingent on the current state. The label condition references a Java method on the view model and should return a boolean. A <strong>labelCondition</strong> defines two <strong>labelOption</strong> sub elements (see below).</td>
</tr>
<tr>
<td><strong>clientTest</strong></td>
</tr>
<tr>
<td>Only generate a row for this action if it passes the specified JavaScript method. If the <strong>clientTest</strong> has a leading exclamation point (!), tests for false.</td>
</tr>
<tr>
<td><strong>clientTestArgs</strong></td>
</tr>
<tr>
<td>An array of parameters pass to the above test, expressed as a comma delimited string.</td>
</tr>
<tr>
<td><strong>action</strong></td>
</tr>
<tr>
<td>The JavaScript method to be fired when the action is taken.</td>
</tr>
<tr>
<td><strong>actionArgs</strong></td>
</tr>
<tr>
<td>An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant.</td>
</tr>
<tr>
<td><strong>leaveMenuOpen</strong></td>
</tr>
<tr>
<td>If true the menu stays displayed after action.</td>
</tr>
</tbody>
</table>
### separator

Not strictly an action. Outputs a separator bar. The style of the bar automatically adjusts to the current nesting level.

### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Only generate a row for this action if it passes the specified JavaScript method.</td>
</tr>
<tr>
<td>clientTestArgs</td>
<td>An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant.</td>
</tr>
<tr>
<td>disabled</td>
<td>Disable this row initially (shows a grey block instead).</td>
</tr>
</tbody>
</table>

### selectAction

A drop down (roll down) parent. Selectors can now be nested up to three levels deep. The menu automatically renders the roll-downs in a nested fashion.

### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>If set to true, disable this row initially (shows a grey block instead).</td>
</tr>
<tr>
<td>disabled</td>
<td>If set to true, selector is opened initially.</td>
</tr>
<tr>
<td>labelKey</td>
<td>The text for the menu to display. If it corresponds to an localization bundle key it is translated, otherwise it is displayed as is.</td>
</tr>
<tr>
<td>labelCondition</td>
<td>Sometimes the label value is contingent on the current state. The label condition references a java method on the view model and should return a boolean. A labelCondition defines two labelOption sub elements (see below).</td>
</tr>
<tr>
<td>clientTest</td>
<td>Only generate a row for this action if it passes the specified JavaScript method.</td>
</tr>
<tr>
<td>clientTestArgs</td>
<td>An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant.</td>
</tr>
</tbody>
</table>

#### 11.6.6.4 Options

Some actions can have child elements, as described in the following tables.

### option

Child of selectAction. Defines a static menu option. Use this when you can't define options programatically.

### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The ID of the menu row DOM object.</td>
</tr>
<tr>
<td>disabled</td>
<td>If set to true, disable this row initially (shows a grey block instead).</td>
</tr>
<tr>
<td>button</td>
<td>If set to true the option displayed as a button (as in formula builder).</td>
</tr>
<tr>
<td>labelKey</td>
<td>The text for the menu to display. If it corresponds to an localization bundle key it is translated, otherwise it is displayed as is.</td>
</tr>
<tr>
<td>labelCondition</td>
<td>Sometimes the label value is contingent on the current state. The label condition references a java method on the view model and should return a boolean. A labelCondition defines two labelOption sub elements (see below).</td>
</tr>
<tr>
<td>clientTest</td>
<td>Only generate a row for this action if it passes the specified JavaScript method.</td>
</tr>
<tr>
<td>clientTestArgs</td>
<td>An array of parameters to be passed to the above test, expressed as a comma delimited string.</td>
</tr>
<tr>
<td>allowsInputTest</td>
<td>Show an input box in this option if it passes the specified JavaScript method.</td>
</tr>
<tr>
<td>option, continued</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>allowsInputTestArgs</td>
<td>An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant.</td>
</tr>
<tr>
<td>action</td>
<td>The JavaScript method to be fired when the action is taken.</td>
</tr>
<tr>
<td>actionArgs</td>
<td>An array of parameters to be passed to the above test, expressed as a comma delimited string.</td>
</tr>
<tr>
<td>isSelectedTest</td>
<td>Indicate a check mark next to this option if it passes the specified JavaScript method.</td>
</tr>
<tr>
<td>isSelectedTestArgs</td>
<td>An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>generatedOptions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Child of selectAction. Defines a set of dynamically-defined options.</td>
<td></td>
</tr>
</tbody>
</table>

**Reserved Variables (in addition to standard reserved variables)**

<table>
<thead>
<tr>
<th>Reserved Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>${optionId}</code></td>
<td>Programmatically assigned ID. If function returns a Map this is the key part of each key-value pair, if it returns a Collection it is the toString value of each element.</td>
</tr>
<tr>
<td><code>${optionValue}</code></td>
<td>Programmatically assigned display value. If function returns a Map this is the value part of each key-value pair, if it returns a Collection it is the toString value of each element.</td>
</tr>
<tr>
<td><code>$R{&lt;String&gt;}</code></td>
<td>When used in a label expression attempts to internationalize the enclosed String and if it fails it returns the literal value.</td>
</tr>
</tbody>
</table>

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The ID of the menu row DOM object.</td>
</tr>
<tr>
<td>function</td>
<td>The name of the java method to invoke on the view model that is used to generate the options. The function can return either a Map or a Collection. The type of object returned affects how the <code>${optionValue}</code> and <code>${optionId}</code> reserved variables are interpreted (see above).</td>
</tr>
<tr>
<td>functionArgs</td>
<td>An array of parameters to pass to the above test, expressed as a string delimited by the @@ string constant.</td>
</tr>
<tr>
<td>labelKey</td>
<td>The text for the menu to display. If it corresponds to an localization bundle key it is translated, otherwise it is displayed as is.</td>
</tr>
<tr>
<td>labelCondition</td>
<td>Sometimes the label value is contingent on the current state. The label condition references a java method on the view model and should return a boolean. A labelCondition defines two labelOption sub elements (see below).</td>
</tr>
<tr>
<td>labelExpression</td>
<td>Allows a custom label to be defined and allows full use of all reserved variables. (e.g. labelExpression=“${optionValue}&quot; $R(ADH_252_DATA_ROWS)”).</td>
</tr>
<tr>
<td>clientTest</td>
<td>Only generate a row for this action if it passes the specified JavaScript method.</td>
</tr>
<tr>
<td>clientTestArgs</td>
<td>An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant. If no actionArgs are specified, the ${optionId} variable is automatically assigned as an argument.</td>
</tr>
<tr>
<td>allowsInputTest</td>
<td>Show an input box in this option if it passes the specified JavaScript method.</td>
</tr>
<tr>
<td>allowsInputTestArgs</td>
<td>An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant. If no actionArgs are specified, the ${optionId} variable is automatically assigned as an argument.</td>
</tr>
</tbody>
</table>
### Customizing the User Interface

<table>
<thead>
<tr>
<th><strong>generatedOptions, continued</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>action</strong></td>
</tr>
<tr>
<td><strong>actionArgs</strong></td>
</tr>
<tr>
<td><strong>leaveMenuOpen</strong></td>
</tr>
<tr>
<td><strong>isSelectedTest</strong></td>
</tr>
<tr>
<td><strong>isSelectedTestArgs</strong></td>
</tr>
</tbody>
</table>

**Default Settings (DOM attributes set automatically on each generated Option)**

| **id**                     | If not specified as an a attribute defaults to the ${optionId} variable. |
| **clientTestArgs**         | If not specified as an a attribute defaults to the ${optionId} variable. |
| **actionArgs**             | If not specified as an a attribute defaults to the ${optionId} variable. |
| **isSelectedArgs**         | If not specified as an a attribute defaults to the ${optionId} variable. |

**labelOption**

Child of labelFunction. Every labelFunction element should have two labelOption elements as children. One of them is used to define the action label depending on the result of the labelCondition.

**Attributes**

| **attributes**             | A boolean string. If the parent labelCondition result matches this boolean value, this labelOption is used. |
| **labelKey**               | The text for the menu to display. If it corresponds to an localization bundle key it is translated, otherwise it is displayed as TgenerateFromTemplate. |

**generateFromTemplate**

Programmatically generates multiple actions by specifying a function to iterate over the enclosed template. The template can be any valid action structure, it can include multiple actions and/or nested actions.

**Reserved Variables (in addition to standard reserved variables)**

| **${templateInjection Index}** | The zero-based index of the current iteration. |
| **${templateInjection Id}**    | Programmatically assigned ID. If function returns a Map this is the key part of each key-value pair, if it returns a Collection it is the toString value of each element. |
| **${templateInjection Value}** | Programmatically assigned value. If function returns a Map this is the value part of each key-value pair, if it returns a Collection it is the toString value of each element. |

**Attributes**

| **function**                | The name of the java method to be invoked on the view model that is used as the template iterator. The function can return either a Map or a Collection. The type of object returned affects how the reserved variables ${templateInjectionValue} and ${templateInjectionId} is interpreted (see above). |
special expressions are converted to built-in variable values when used as arguments for client functions:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>${selected}</code></td>
<td>The JavaScript array of selected objects on the client.</td>
</tr>
<tr>
<td><code>${event}</code></td>
<td>The JavaScript event object.</td>
</tr>
<tr>
<td><code>${label}</code></td>
<td>The generated label for this menu.</td>
</tr>
</tbody>
</table>

### 11.6.6.6 Menu DHTML API

This is a set of JavaScript functions defined on actionModel.js to dynamically manipulate a menus look and feel after it has already been displayed. You can use them to change the Ad Hoc Editor menus in run-time. Most of them take the `menuRow` or the `menuRow`'s identifier as arguments. Some have additional arguments as well; for more information about the additional arguments, refer to the code itself.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hideInputForOption</td>
<td>Hide the input box on the option.</td>
</tr>
<tr>
<td>showInputForOption</td>
<td>Show the input box on the option.</td>
</tr>
<tr>
<td>setRowColor</td>
<td>Set the <code>backgroundColor</code> to the specified color.</td>
</tr>
<tr>
<td>resetRowColor</td>
<td>Restore original background color.</td>
</tr>
<tr>
<td>getLabel</td>
<td>Return the current label for this row.</td>
</tr>
<tr>
<td>setLabel</td>
<td>Set the label for this row.</td>
</tr>
<tr>
<td>disableRow</td>
<td>Grey out the row.</td>
</tr>
<tr>
<td>enableRow</td>
<td>Restore the row to active state.</td>
</tr>
</tbody>
</table>

### 11.7 Working With Source Code Files

The rest of the customizations in this chapter require writing Java classes to perform some part of the new functionality. Even if you don’t modify the source code of JasperReports Server, you must compile your code with the server in order to deploy your custom features. When you redeploy the web application the existing files that are running in the application server are replaced with those you just compiled. As a result, you must make all of your changes in the source code.

Customizations that involve changes to Java source code have the following requirements:

- You must download the source code distribution and set up an environment where you can build it.
- All files being changed must be edited in the source code, even the interpreted files. This section gives the path to all files in `<js-src>`, which represents the root of the source code. Pay close attention to the path names, because many are similar.
- In order to see the changes, you must then build the source code and redeploy the web application.
To obtain the source code distribution, see the following links. To build and deploy the source code, follow the instructions in the *JasperReports Server Source Build Guide* within each distribution:

Commercial Editions: [http://support.jaspersoft.com](http://support.jaspersoft.com) then navigate to Downloads and Source Code

The following procedure gives an example of the steps for building and re-deploying the web app after making changes to the source code. This example uses the commercial source code distribution and assumes you are using Apache Tomcat in a Windows environment.

**To rebuild the source code:**

1. Make sure that all your file changes are saved in the `<js-src>` tree.
2. Stop the application server.
3. Select the **Start Menu > Accessories**, right-click **Command Prompt**, and select **Run as Administrator**.

   ![If you do not run Command Prompt as administrator, the build can fail during the deployment phase due to permissions problems when adding and deleting files.]

4. Go to the buildomatic directory in the source distribution:

   ```
   cd <js-src>/jasperserver/buildomatic
   ```

5. Enter the following commands, checking for the BUILD SUCCESSFUL message upon completion of each one:

   ```
   js-ant build-ce
   js-ant build-pro
   js-ant deploy-webapp-pro
   ```

6. Restart Tomcat.

**11.8 Adding a Custom JSP Page in a Spring Web Flow**

The flexibility of JasperReports Server lets you create your own JSP pages that integrate into the UI. In order for the SiteMesh decorator to process a custom JSP page, we must integrate it into the Spring Web Flow framework. A flow is a sequence of related pages for which you define states and transitions in relation to your own business logic. In this example, we add a single page, but you could integrate a series of pages and the navigation between them.

To further integrate with the server, your pages should use the CSS building blocks provided in themes to replicate the menu and column layout of the server. You can then apply the default theme of the UI or design your own style in a custom theme.

Spring Web Flow relies on the Spring MVC (Model, View, and Controller) module to implement the web interface, where the controller is a Java class. As a result, adding a web flow involves creating a Java method to handle the new page. In general, the server UI contains most of the functionality in the action class code that is associated with a particular JSP page. The JSP files have minimal functionality because JSP code logic can become very cluttered and hard to follow. The action classes are pure Java and thus easier to organize. In this example, the Java method simply returns the default model and view.

For more information, refer to the Spring documentation for [flows](http://docs.spring.io/spring/docs/3.2.8.RELEASE/spring-framework-reference/html/mvc.html) and [MVC](http://docs.spring.io/spring/docs/3.2.8.RELEASE/spring-framework-reference/html/mvc.html).

This example requires you to work with the JasperReports Server source code, as explained in section **11.7, “Working With Source Code Files,”** on page 184.
Example of adding a custom JSP file integrated into the server as a web flow:

1. Create the following XML file with a flow container element and an empty view-state element. If you had a custom flow with multiple pages, you would define your states and transitions here:

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <flow xmlns="http://www.springframework.org/schema/webflow"
        xmlns:ns0="http://www.w3.org/2001/XMLSchema-instance"
        ns0:schemaLocation="http://www.springframework.org/schema/webflow
        http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd"
        start-state="startHere">  
    <view-state id="startHere" view="accounts">  
      </view-state>
    </flow>
   
   Figure 11-13 Samples Gallery Showing Typography in Themes
   ```


3. Design the layout of your custom content based on the UI components in the server. To see the basic UI components:
   a. Log into the server as jasperadmin.
   b. Select View > Samples and browse through the galleries to see the UI components.
      If you have a custom theme active, you see the components with your theme in effect.
   c. In this example, we use the one-column layout and the text accent provided by the CSS warning class visible in the typography gallery.
   d. Open the file <js-src>/jasperserver-war/src/main/webapp/themes/default/pages.css. It contains instructions for using the column layout.
4. Create a JSP file to display your content using the UI elements. For example the one-column layout applies to the body, and the warning class applies to text.

```html
<!DOCTYPE html>
<html>
<head><title>MyCompany Accounts</title>
<link href="favicon.ico" rel="shortcut icon">
</head>
<body class="oneColumn primary column">
<h1>MyCompany Regional Reports</h1>
<h2 class="warning">Time on server: <%= new Date() %></h2>
...<context name="main_view_mutton" test="!banUserRole">
<selectAction labelKey="menu.repository">
  ...
  <separator/>
  <option labelKey="NAV_028_ACCOUNTS" action="primaryNavModule.navigationOption" actionArgs="accounts"/>
</selectAction>
</context>
... NAV_028_ACCOUNTS=MyCompany Accounts ...
```


6. Now we add the menu item to call the page we created, as we did in section 11.6.3, “Adding an Item to the Main Menu,” on page 175. Edit the file <js-src>/jasperserver/common/shared-config/actionModel-navigation.xml. Locate the actionModel for the View menu near the beginning of the file.

```xml
<--context for view option on primary menu-->
<context name="main_view_mutton" test="!banUserRole">
  <selectAction labelKey="menu.repository">
    ...
    <separator/>
    <option labelKey="NAV_028_ACCOUNTS" action="primaryNavModule.navigationOption" actionArgs="accounts"/>
  </selectAction>
</context>
```

If you want to restrict access to the menu item, specify a condition tag around the menu option. You should also restrict access to flows using the flowVoter bean in the file <js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/applicationContext-security.xml.

7. Add a an option tag for the menu item, as shown in the example above. The option tag has attributes to specify the label key and the name of the action we want to perform.

8. Add the label key to the file <js-src>/jasperserver-pro/jasperserver-war/src/main/webapp/WEB-INF/bundles/pro_nav_messages.properties, as shown in the following code sample:

```properties
... NAV_028_ACCOUNTS=MyCompany Accounts ...
```

9. Edit the file <js-src>/jasperserver/jasperserver-war/src/main/webapp/scripts/actionModel.primaryNavigation.js, and add a line that specifies the accounts flow to begin when the menu item is selected.

```javascript
var primaryNavModule = {
  ...
  navigationPaths : {
    ...
    accounts : {url : "flow.html", params : "_flowId=accountsFlow"},
    ...
  }
}
10. Now we edit the java class that defines the controller in the Spring MVC framework. Edit the file `<js-src>/jasperserver-war/src/main/java/com/jaspersoft/jasperserver/war/control/JSCommonController.java`:

```java
public ModelAndView accounts(HttpServletRequest req, HttpServletResponse res)
    throws ServletException {
    return new ModelAndView("accounts");
}
```

11. Edit the file `<js-src>/jasperserver-war/src/main/webapp/WEB-INF/jasperserver-servlet.xml`:
   a. In the `urlHandlerMapping` bean, add the following property. The `jsCommContr` is the ID of the bean of class `JSCommonController` that we modified above:

   ```xml
   <prop key="/accounts.jsp">jsCommContr</prop>
   ```
   
   b. In the `paramResolver` bean, add the following property. The key is the URL that is displayed for our custom page:

   ```xml
   <prop key="/accounts.jsp">accounts</prop>
   ```

12. Rebuild the source code and redeploy the web application according to the instruction in the JasperReports Server Source Build Guide within your distribution. Section 11.7, “Working With Source Code Files,” on page 184 gives an overview of this process.

13. If your changes were successful, this example displays the View > MyCompany Accounts menu item to users, and selecting it displays the custom page defined in the JSP file from step 4:

![Custom JSP in a Custom Flow](image)

The page uses the single column layout and the red text is the `warning` class in CSS.

### 11.9 Adding Custom Export Channels

When users run reports in JasperReports Server, they can export the results in several formats, such as PDF and ODT. If your users need to export to other file formats, you can create a custom export channel. You must implement a custom Java class that generates the required file format then integrate the new class into the server. As a result, this customization must be made in the source code of JasperReports Server.

In the following example, a custom export channel is added to the server’s pluggable, flexible export channel list. The example adds the export channel in three places: the report viewer, the scheduler, and web services.

This section assumes the following:
- You are familiar with the underlying technologies, such as Java, Spring, JSPs, and web services.
You have downloaded and tested the source code distribution, as described in section 11.7, “Working With Source Code Files,” on page 184. Paths and filenames in this section are based on the <js-src> location.

You have implemented an exporter class that can be integrated with JasperReports Server. The section further assumes that your exporter class creates files with a .<MyFormat> file extension and format.

That, where the instructions read <MyFormat>, you have used the correct name for your export format. For example, if you are creating an exporter that generates .DBF files, where the instructions describe the Report<MyFormat>Exporter class, your class is called ReportDBFExporter.

### 11.9.1 About Export Parameters

Export parameters define how JasperReports Server generates your output format. For example, consider the default Excel exporter. Its export parameters determine how a report is converted to the XLS file format. The parameter settings have such information as whether each page of the report should be represented as a separate sheet in the Excel spreadsheet (IS_ONE_PAGE_PER_SHEET).

Export parameters have default values that can be set at the report- or application-level. Many of the parameters are optional; others are valid only for certain export channels. When you create an export channel, you can define such parameters to control how your reports are exported in the new format. If you don’t need to specify such settings when reports are run, you might not need to define any parameters.

### 11.9.2 Adding the Exporter to the Report Viewer

The most prominent place that the export channel should be available is the page that appears when the report is run. Figure 11-8, “Default Appearance of the Report Viewer,” on page 170 shows the drop-down list of export channels in the default report viewer. Adding a new exporter to the report viewer involves:

- Implementing a new exporter action class that extends AbstractReportExporter.
- Implementing a new export parameters bean class that extends AbstractExportParameters.
- Creating a new export configuration bean of type ExporterConfigurationBean.
- Adding a resource bundle property for the label in the menu.

#### To add the exporter to the default report viewer:

1. Create a new exporter action class that extends the abstract com.jaspersoft.jasperserver.war.action.AbstractReportExporter class. Name it Report<MyFormat>Exporter. For guidance about creating such a class, look at the ReportCsvExporter class, which is similar. It is found in the com.jaspersoft.jasperserver.war.action package.

2. Create a new export parameters Java bean class, which should implement Serializable and extend the com.jaspersoft.jasperserver.api.engine.jasperreports.common.AbstractExportParameters class. It should implement the public void setPropertyValues(Object object) method, which is required to perform dynamic data binding and validation. Give the class a unique name, such as <MyFormat>ExportParametersBean, and ensure that it contains only user-customizable export parameters (that is, it should not contain the JRExporterParameter.JASPER_PRINT or JRExporterParameter.OUTPUT_FILE_NAME parameters, which are already set). For an example, see the XlsExportParametersBean class in the com.jaspersoft.jasperserver.api.engine.jasperreports.common package.

3. If your users should set some parameters for the export, associate a new intermediary JSP page in the web flow. When a user clicks your new export button, a pop-up appears before the report is displayed; it prompts for all the customizable parameters’ values.

   All the export parameters have default values, and they are bound to the newly-created JavaBean’s properties. When binding data in the JSP page, the name of the bean in the request scope is always the value of the exporter’s parameterDialogName property (see step 4, next). When a user clicks the dialog’s Submit button, the server exports the report after validating the user’s input. By default, the new JSP export pages should be stored in the folder <js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/jsp/exporters.

   If the user doesn’t need to provide any input, skip this step. In such a case, it’s better to use the exporter hint properties, as described in step 4, next.

4. Now that you have everything you need to view the new export format, and you must configure it for Spring.
There are two ways to set export parameter values:

- **Application-level settings.** These settings set the defaults for all reports in the repository, including all export parameter values. They apply in the absence of report-level settings. This is the server’s default behavior.
- **Report-level settings.** These settings allow an individual report to have different values from the default. Every report in the repository can have user-defined export parameter values.

To enable report-level settings, edit the file `<js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/ applicationContext.xml`. Locate the `configurationBean` bean, and set the `reportLevelConfigurable` property to `true`. This enables the report-level settings mode; otherwise, parameter values are inherited from the application-level settings. If a report includes exporter hint properties, they override the application-level values.

5. Edit the file `<js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/flows/viewReportBeans.xml`. The configurable list of exporters is controlled by the `configuredExporters` property in the `reportExporter` bean. The list of exporters is defined by the `exporterConfigMap` at the end of the file. Its keys are important because it is the key name that is sent to the server when a user clicks an export button. Key names are also part of some state names in the associated web flow, so you must be careful to use correctly-implemented names when adding a new object in the `exporterConfigMap` map.

As a rule, all key names should correspond with the extension name of the file generated by the given exporter (for example, the key for Excel exporter is `xls`, and the key for PDF exporter is `pdf`). While using file extensions as key names is not mandatory, Jaspersoft strongly recommends it, as in the case of `<MyFormat>`.

Add a new entry in the `exporterConfigMap` element:

```xml
<entry key="<MyFormat>" value-ref="<MyFormat>ExporterConfiguration"/>
```

6. The values in the `exporterConfigMap` object are `com.jaspersoft.jasperserver.war.action.ExporterConfigurationBean` objects, which defines a custom type that stores configuration information about any given exporter.

Create a bean named `<MyFormat>ExporterConfiguration` with the following properties. For guidance, refer to the similar objects in the `viewReportBeans.xml` file.

- **iconSrc** – A context-relative path to the icon for the tool bar button; usually icons are stored in the `/images` location. Any new exporter should have a related icon image saved in that directory. The icon should be 18 pixels by 18 pixels.

- **descriptionKey** – A key in the `jasperserver_messages.properties` resource bundle file that should be displayed as the tooltip when the mouse is held over exporter’s icon; you’ll add this key to the `jasperserver_messages.properties` file in step 8.

- **parameterDialogName** – The name of the JSP file (without its .jsp file extension) to use for a pop-up dialog that prompts your user for export-time parameters (if any); this name is used in the JSP page to identify the icon mapped on the form object. The page only appears if the report-level settings mode is enabled, otherwise the server ignores it and default values are used. If there is no related JSP page, set the `parameterDialogName` to an empty string.

- **exportParameters** – A `com.jaspersoft.jasperserver.api.engine.jasperreports.common.ExportParameters` XML bean, which wraps a specific export parameters class that contains export parameter default values. In the case of this example, use the `<MyFormat>ExportParametersBean` class you created in step 2. Name it `<MyFormat>ExportParameters`.

- **currentExporter** – An `AbstractReportExporter` type object that contains specific export business logic; for example, the `Report<MyFormat>Exporter` class you created.

7. Still in the file `viewReportBeans.xml`, write a bean with an element named `report<MyFormat>Exporter` for your new exporter class. For guidance, refer to the similar `reportCsvExporter` bean.

8. In the file `<js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/bundles/jasperserver_messages.properties`, add new key for your exporter’s name and tooltip:

```properties
report.output.<MyFormat>.label=<MyFormat>
report.output.<MyFormat>.label.tooltip=MyFormat Explanation
```

If your server instance supports multiple languages, be sure to add the correct entries in the properties file for each supported language.

9. Finally, edit the file `<js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/applicationContext.xml` to add the additional export parameters. Create a bean with the name `<MyFormat>ExportParameters` and configure it. For guidance, refer to the similar `ExportParameters` beans in the file. You must also edit the `applicationContext-report-
scheduling.xml file found in the same location; this file defines the parameters used when reports are scheduled. For more information about configuring the exporter for scheduled reports, refer to the following section.

10. The new output format is now configured. Compile the source code and redeploy the web application, as described in section 11.7, “Working With Source Code Files,” on page 184.

Your new exporter appears in the list of exporters when you run a report. When selected, the report is exported in the new file format.

### 11.9.3 Adding the Export Format to the Scheduler

When users schedule reports, they specify the file format to generate when the report runs. The report scheduling mechanism is more complicated that the mechanism that displays reports directly to users, so the process of adding an export format to the scheduler is also more complicated.

<table>
<thead>
<tr>
<th>Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDF</td>
</tr>
<tr>
<td>OTT</td>
</tr>
</tbody>
</table>

**Figure 11-15 Default Export Formats on the Scheduler Output Page**

Adding a new export format to the scheduler requires these steps:

- Implement the `Output` interface.
- Define a key for your exporter.
- Implement a new export parameters bean class.
- Register the new export format in Spring configuration files.

**To add the exporter to the report scheduler:**

1. Create a class that implements the `com.jaspersoft.jasperserver.api.engine.scheduling.quartz.Output` interface and name it `<MyFormat>ReportOutput`. You must implement the `getOutput()` method, which generates the output to the new `myformat` format and returns a `com.jaspersoft.jasperserver.api.engine.scheduling.quartz.ReportOutput` object. If your format required custom default values for the export parameters at the application level, this class should contain also a `com.jaspersoft.jasperserver.api.engine.jasperreports.common.ExportParameters` property, named `exportParams`. For guidance, refer to the `com.jaspersoft.jasperserver.api.engine.scheduling.quartz.XlsReportOutput` class.

2. Associate a new key with your `<MyFormat>ReportOutput` business class and name this key `<MyFormat>`. In order to preserve backward compatibility, the keys and integer numbers must correspond. Edit the file `<js-src>/jasperserver-war/src/main/webapp/WEB-INF/bundles/jasperserver_config.properties` to add a new row:

   ```properties
   report.scheduling.output.format.{number}=<MyFormat>
   ```

   Replace the `{number}` placeholder with the next greater integer number to continue the series defined in the properties file.

3. Edit the file `<js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/applicationContext-report-scheduling.xml` to make the following changes:
   
   a. Add a new entry in the `outputKeyMapping` bean similar to the following:
      
      ```xml
      <entry key="{number}" value="<MyFormat"/>
      ```
      
      You have to replace `{number}` and `<MyFormat>` with the values you specified in step 2. For example, if you were adding a DBF exporter, the line might be similar to:
      
      ```xml
      <entry key="{11}" value="DBF"/>
      ```
      
   b. Create a new `<MyFormat>ExportParameters` bean and specify a parent listed in the `applicationContext.xml` file. If necessary, the new bean should contain scheduling-specific values for export parameters. If no values are set, the default values are read from the parent bean. For guidance, refer to the `jobXlsExportParameters` bean.
      
      If no `<MyFormat>ExportParameters` is created, the server assumes the export parameters weren’t set, and the JasperReports engine automatically handles their values.
c. Add the new `<MyFormat>ExportParameters` bean in the `jobExportParametersMap` map object. Follow the pattern used by the existing map entries.

d. Create a new `<MyFormat>output` bean object of type `com.jaspersoft.jasperserver.api.engine.scheduling.quartz.<MyFormat>ReportOutput`, which wraps the `<MyFormat>ReportOutput` object you created in step 1. If the `exportParams` property is present, the `<MyFormat>output` bean should contain a property named `exportParams`, as well. For guidance, refer to the `htmlOutput` and `xlsOutput` objects.

e. Add the new `<MyFormat>output` bean in the `outputFormatMap` map object. Follow the pattern set by the existing map entries.

4. Edit the file `<js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/flows/reportJobBeans.xml` to add the new output format in the `allOutputFormats` list:

```xml
<bean class="com.jaspersoft.jasperserver.war.dto.ByteEnum">
  <property name="code">
    <value type="java.lang.Byte">{number}</value>
  </property>
  <property name="labelMessage">
    <value>report.output.<MyFormat>.label</value>
  </property>
</bean>
```

The `{number}` and `<MyFormat>` values are the same as you used previously in step 2.

5. If you didn’t add the label in step 8 in section 11.9.2, "Adding the Exporter to the Report Viewer," on page 190, add it now. In the `jasperserver_messages.properties` file (found in `WEB-INF/bundles`), add the new `report.output.<MyFormat>.label` key introduced in the `reportJobBeans.xml`. For example:

```ini
report.output.<MyFormat>.label=<MyFormat>
```

If your server instance supports multiple languages, be sure to add the correct entries in the properties file for each supported language.

6. The new scheduler output format is now configured. Compile the source code and redeploy the web application, as described in section 11.7, “Working With Source Code Files,” on page 184. Your new export channel is added to the scheduler’s Output page, and when selected a scheduled report generates the output in your new format.

### 11.9.4 Adding the Export Channel to Web Services

If your JasperReports Server instance is accessed through web services, you can add your export channel to the dedicated web services export channel, as follows:

- Implement the `WSExporter` interface.
- Implement a new export parameters bean class.
- Add your exporter bean to the map of available channels.

To add the export channel to the dedicated web services export channel:

1. Create a class that implements the `com.jaspersoft.jasperserver.ws2.WSExporter` interface, and name it `<MyFormat>WSExporter`.

   You must implement the `exportReport()` method, which generates the output to the new myformat format and returns a map of export parameters object. If custom export parameters are involved (that is, if you want to specify default values for export parameters at the application level), this class should also contain a `com.jaspersoft.jasperserver.api.engine.jasperreports.common.ExportParameters` property, which should be named `exportParams`. For guidance, refer to the `com.jaspersoft.jasperserver.ws2.XlsWSExporter` class.

2. Edit the file `<js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/applicationContext-web-services.xml` to create a new `<MyFormat>Exporter` bean object of type `com.jaspersoft.jasperserver.ws2.<MyFormat>WSExporter` (which wraps the `<MyFormat>WSExporter` object you created in the previous step). If the `exportParams` property is present, the `<MyFormat>Exporter` bean should contain a property named `exportParams`, as well. For guidance, refer to the `htmlExporter` and `xlsExporter` objects.
You must also add a new entry in the exportersMap map object; it should be similar to the existing entries. It must contain:

- `<MyFormat>` as the key.
- The `<MyFormat>Exporter` object as referenced value.

3. Add a new entry in the exportParametersMap map object, similar to the existing entries. It must contain:

- `<MyFormat>` as key
- The `<MyFormat>ExportParameters` object defined in step 6 in section 11.9.2, "Adding the Exporter to the Report Viewer," on page 190 as the referenced value.

If no `<MyFormat>ExportParameters` object is defined in the applicationContext.xml file, then create one.

4. The new output format is now configured with web services. Compile the source code and redeploy the web application, as described in section 11.7, “Working With Source Code Files,” on page 184.

Web service clients that request the list of exporters will receive your new exporter’s name, and when selected to run a report, the client can receive the report in the new format.

### 11.9.4.1 Removing an Export Channel from the Exporters List

To hide a format exporter in the exporters list:

- For the report viewer – Edit the file `<js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/flows/viewReportBeans.xml`, locate the exporterConfigMap element, and comment out this line:

  `<entry key="<MyFormat>" value-ref="<MyFormat>ExporterConfiguration"/>

- For the report scheduler – Edit the file `<js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/flows/reportJobBeans.xml`, locate the allOutputFormats list, and comment out this code:

```xml
<bean class="com.jaspersoft.jasperserver.war.dto.ByteEnum">
  <property name="code">
    <value type="java.lang.Byte">{number}</value>
  </property>
  <property name="labelMessage">
    <value>report.output.<MyFormat>.label</value>
  </property>
</bean>
```

- For web services – Edit the file `<js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/applicationContext-web-services.xml` file, locate the exportParametersMap element, and comment out this line

  `<entry key="<MyFormat>" value-ref="<MyFormat>ExportParameters"/>
```
CHAPTER 12 DESIGNING A CLUSTER

To provide scalability and high availability of your business intelligence infrastructure, you can deploy a cluster of JasperReports Server instances behind a load balancer. You can implement a cluster with either JasperReports Server Community Project or commercial editions, as long as all instances are the same edition, the same version, and all are configured identically.

JasperReports Server supports cluster deployments by using thread-safe access to its private repository database. As a result, any number of JasperReports Server instances can share the same repository and thus present the same environment to users. Of course, your repository database must be properly sized to handle the number of server instances, and it must have its own mechanism for providing scalability and high availability.

With a properly designed cluster, you can support many more users and organizations, avoid unintended downtime, and plan for future growth. The load balancer makes sure that user load is spread evenly, and when needed, you can add new instances of JasperReports Server to the cluster.

One important detail of any cluster is how sessions are managed in case of a failure. JasperReports Server supports only “sticky” sessions, also called “pinned” sessions, that cannot be transferred between nodes. User sessions in progress on the node that fails are lost, but all work saved in the repository can be immediately resumed on any other node.

This chapter introduces a sample architecture for JasperReports Server cluster environments and explains its components. It also discusses design considerations and deployment constraints, such as session management. This document is not intended as a tutorial, nor as a detailed deployment plan, but should only be used as a high level overview.

This chapter contains the following sections:

- Sample Cluster Architecture
- Jaspersoft OLAP in a Cluster
- Session Management and Failover
- Cluster Design Process
- Performance Requirements
- Availability Requirements
- Scalability Requirements
- Sizing a Cluster

12.1 Sample Cluster Architecture

A cluster refers to a group of servers, along with any associated computers, dedicated hardware, and other server software that perform the same task as a single server. Each server instance runs on its own node, a real or virtual computer with the
necessary software. When properly configured, the cluster architecture is transparent to users. All users access the same URL and see the same data, but each session is handled by a different node.

A cluster typically provides load balancing among nodes and some form of failover, both of which lead to higher availability and scalability. A cluster design must also take into account all of the resources that the JasperReports Server instances must access and scale them appropriately.

In general, a cluster may incorporate computers of different hardware and software configurations. For simplicity, Jaspersoft recommends deploying JasperReports Server as a cluster of identical nodes, with one instance on each node and each instance configured the same.

The following diagram shows the architecture of a sample JasperReports Server cluster:

![Architecture of a Sample JasperReports Server Cluster](image)

The major components of the sample JasperReports Server cluster architecture are:

- **JasperReports Server clients:**
  - Browser users – Administrators and end users who log into the JasperReports Server web interface.
  - Web services clients – Applications that access JasperReports Server through the web services API.
  - Load balancer – Specialized hardware or software that redirects client requests to instances in the cluster.
  - JasperReports Server instances – Identically configured instances running on separate computers, real or virtual.

- **Shared resources:**
  - Repository database – Defines users, roles, organization, folders, and resources for the cluster. The repository is a single logical database that should also be configured for high availability and failover.
  - Data sources – Contain the data that JasperReports Server queries when creating or running a report. Data sources should be able to handle the load of simultaneous queries expected from the cluster.
  - Email services – Send email notifications and output of scheduled reports. Email services are optional, but almost always implemented.
  - External authentication – Provides alternative login policies such as corporate directories or single sign-on. External authentication is optional and more complicated to configure, but it is often desired in an enterprise deployment.

The requirements and considerations for each of these components are given in the following sections. For simplicity, the sample architecture assumes that all components of the cluster are in the same geographic location. Distributed clusters to serve distributed users are possible, but require extra network and performance considerations that are beyond the scope of this chapter.
12.1.1 JasperReports Server Clients

JasperReports Server supports two types of clients that are fundamentally different:

- **Browser users** – Through the web interface, users interact with all the features of JasperReports Server, such as viewing and scheduling reports. Working with Ad Hoc reports and Dashboards, in product editions that provide these features, is an interactive process for the user, with multiple requests to access data, display it, and modify its appearance. Users with the proper permissions can upload JasperReports and use the wizards to define the required resources.

- **Web services clients** – Web services are a web-based API that applications call to access the features of JasperReports Server. For example, the JasperReports Server plug-in for iReport relies on web services to access the repository for reading and writing reports and their associated resources.

A web services client provides credentials in every request, and the server performs an action or provides information in a reply to the client. Web services calls are stateless operations, meaning that the server does not store any information about the operation after it is completed. Complex structures such as a report resource are fully defined in the client before being sent in the web services call, as opposed to the interactive nature of the browser user. For more information, see the JasperReports Server Web Services Guide.

User sessions in a cluster are further explained in section 12.3, “Session Management and Failover,” on page 200.

A significant task in the design of your cluster is to characterize client usage, such as total number of potential clients, fraction of browser users and web services clients, peak client load, average request size, and typical client bandwidth. This information will help you optimize the size and number of servers in your cluster to meet your service availability goals. For more information, see section 12.8, “Sizing a Cluster,” on page 203.

12.1.2 Load Balancer

A load balancer is a hardware device or software application that uses any number of techniques to spread traffic between the nodes of the cluster, usually so that all servers have an equal load. The load balancer provides a single address that all clients can access, and it behaves like an internet router, maintaining each client’s connection with the chosen server in the cluster.

The load balancer is the gateway to the cluster because it directs client traffic and thus optimizes the performance of your servers. At a minimum, the load balancer can determine that a server is not operating and direct traffic to the remaining functioning servers. Some load balancers offer capabilities such as analyzing client requests and server load to optimize server response times for clients. The JasperReports Server itself does not communicate with load balancers, but some load balancers may communicate with the application server to determine availability and load.

Because of the requirement for sticky sessions in a JasperReports Server cluster, the load balancer must be configured so that browser users are always connected to the same server during a continuous session. Beyond that requirement, JasperReports Server can work with any HTTP load balancer, both hardware or software-based. The load balancer module in the Apache HTTP server is a common software solution.

If you have high concurrent user loads and rigorous availability requirements, you may need a load balancer with more advanced features for balancing client traffic. Also, the load balancer capabilities and configuration sometimes limit the maximum number of servers in your cluster, an important consideration for scalability. Thus, your cluster requirements ultimately determine your choice of load balancer.

12.1.3 JasperReports Server Instances

The JasperReports Server instances deployed in a cluster are normal servers with no special configuration for the cluster. As with all server instances, they run in a Java servlet container that is usually implemented as an application server. While the load balancer may interact with the application server, JasperReports Server is not aware of the cluster environment in which it is deployed.

For performance and stability reasons, server instances are usually deployed on dedicated computers. The hardware can be a physical computer or a cloud-based virtual computer, as long as the operating system is supported by JasperReports Server. Jaspersoft recommends that the only other software installed be the required Java Virtual Machine (JVM) and the application server. This design allows you to allocate as much memory as possible to the JVM and to JasperReports Server.

This document considers only the case of deploying a single JasperReports Server as the only application in a single application server on each computer. Therefore, in this document, a node refers to the real or virtual computer hosting a JVM, an application server, and a single instance of JasperReports Server.
Jaspersoft recommends that you use a lightweight application server such as Apache Tomcat or Glassfish. For example, Tomcat is a well-tested application server with JasperReports Server. A typical configuration for handling most reporting uses 64-bit Java with 2 gigabytes of memory allocated to the Java heap. See the JasperReports Server Installation Guide for guidelines and instructions to install JasperReports Server in your choice of application server and to configure your JVM.

Some application servers can be configured to communicate within a cluster, for example to deploy applications across all instances automatically. This advanced configuration is beyond the scope of this document, and can be undesirable in some cases. For example Apache Tomcat can perform session replication when instances are configured to communicate with each other in the cluster. But JasperReports Server does not support session replication, as explained in section 12.3, “Session Management and Failover,” on page 200.

In this document, each application server in the cluster is running independently and unaware of the other instances.

Finally, every instance must have an identical configuration, at least for certain key settings. While it is possible for each instance to be configured differently, this is not recommended. For example, when instances are deployed on differing hardware with different configurations, the load balancer’s algorithm may not be effective. This is why we recommend that every server in the cluster be deployed on the same hardware and software, and have exactly the same configuration.

To deploy identical JasperReports Server instances, install a single instance from the WAR file distribution and configure it completely. Then copy the deployed WAR file to each of the other computers in the cluster. When using JNDI data sources provided by the application server, you must make sure they are pre-configured and identical on every application server. Often, the entire application server with JasperReports Server deployed within it can be copied as well. This is sometimes called “cloning” the server. Upgrades can be handled in the same way. Having identical JasperReports Server instances in your cluster greatly simplifies its installation and maintenance.

Another way to deploy identical instances to each node is to implement the configured WAR file on a shared file system and to access it from each application server with a symbolic link. You must make sure that working directories such as javaiotemp are not shared, and the shared file system should have its own redundancy and failover plan. Shared WAR files are an advanced configuration, and further details are beyond the scope of this document.

12.1.4 Shared Repository Database

The keystone of the JasperReports Server cluster is the shared repository database. All JasperReports Server instances must be configured to access the same repository, thus ensuring that they display the same folders, the same reports, and the same resources. Because the repository also stores users, roles, organizations, and security definitions, every server in the cluster behaves identically regardless of which instance actually processes the client connection.

For example, when a user logs in, the user account, organization, and roles are retrieved from the shared repository. When the user browses folders, all resources are the same as seen from any instance in the cluster, and all access permissions apply. If the user runs a report based on a Domain Topic, the report, the Domain Topic, the Domain, and any security files are all retrieved from the repository.

All repository operations are thread-safe, meaning that all JasperReports Server instances can perform operations simultaneously, while internal locks in the software prevent operations that would conflict. If a user edits and saves a report, another user running the report sees either the old report or the new one based on exactly when the request was made. As with many large systems serving numerous clients, users must coordinate their work to avoid overwriting each others’ changes. For the same reason, administrators should make changes to shared resources during off-hours to prevent conflicts with active user sessions.

Because the repository database is such a critical element of the JasperReports Server cluster, you should implement it on an equally scalable and available system, based on the predicted peak and average loads of your JasperReports Servers. For example, the repository could be a cluster itself, with its own load balancer, or any other architecture that is compatible with a database product supported by JasperReports Server. In addition, the repository database must implement data protection measures you require, such as on-site and off-site backups.

Regardless of the architecture of the repository database, it is critical that it act as a single logical repository which is identified with a single IP address that all JasperReports Server instances can access.
12.1.5 Job Schedulers

The JasperReports Server scheduler is a module that exists in every server instance to run reports scheduled to run at a later time, either once or repeatedly. Schedules, also known as jobs, are stored in the repository to be triggered by the scheduler whenever necessary. For simplicity, schedulers are not shown in Figure 12-1 on page 196.

To prevent every instance from triggering the same job at the same time, the scheduler uses a software locking mechanism when accessing jobs in the repository. This allows the scheduler to be deployed on every instance in a cluster environment but ensures that any job is triggered only once. Jobs can be created by client sessions on any instance, then be run by the scheduler on any instance, and the client sees no difference. Job output will then be emailed as necessary, and if saved in the repository, accessible from any instance.

The scheduler is based on the Quartz scheduler, an open source library. The JasperReports Server includes settings for the scheduler in the file <WAR-file>/WEB-INF/js.quartz.properties. The default configuration of the scheduler includes the following settings that allow it to work in both stand-alone servers and clusters:

```properties
org.quartz.scheduler.instanceId = AUTO
org.quartz.jobStore.isClustered = true
```

Be sure to synchronize the clocks on every node, so that the scheduler doesn’t always run jobs on the node with the earliest time. For further details, see the online documentation for the Quartz scheduler.

There are advantages to running jobs on all the server instances. If several long jobs are scheduled at the same time, a single server must process them sequentially, and some won’t start at exactly the designated time. Multiple servers in a cluster can process those jobs in parallel, and they will start on time, at least for a number of jobs up to the number of instances.

12.1.6 Other Shared Resources

As with the repository database, the server instances in the cluster usually share any other resource that they need to access, including:

- **Data sources** – These are defined in the shared repository, therefore all servers in the cluster access the same data sources. Queries, reports, and Domains are all stored in the repository as well, so that a report has access to exactly the same data, regardless of which instance the user session or job is running on.

  Data sources can be defined as either JDBC connections directly to a database or JDBC connections from the application server to the database that are exposed through JNDI. Jaspersoft recommends using JNDI data sources because the application server often has better connection pooling and management than JasperReports Server.

  However, JNDI data sources require two definitions, one in the repository and one in the application server. You must make sure that the JNDI definitions are identical on every application server in the cluster. Otherwise, the JNDI data source defined in the shared repository won't work for every instance of JasperReports Server. For more information, see section 12.1.3, “JasperReports Server Instances,” on page 197.

- **Email services** – These are defined in each server instance’s configuration files for sending the output of finished jobs. When servers are configured identically, every instance uses the same email services.

- **External authentication** – An optional server configuration that allows JasperReports Server to access an external user database to verify login credentials. As described in the JasperReports Server External Authentication Cookbook, external authentication requires extensive configuration and sometimes customization. When implementing external authentication in a cluster, all instances must be configured identically to avoid security holes.

Resources that are defined in the repository are shared by all nodes and should be able to handle connections from multiple servers. For resources that are configured in server files, each instance could define a dedicated resource in its own configuration file. For example, each server instance could have its own email server to handle the notifications it sends. However, the dedicated resource then becomes a point of failure that disables the corresponding server instance if the resource fails. In either case, shared resources are often clusters themselves, to provide the same reliability and scalability as the JasperReports Server cluster.

Having a single address and set of credentials for resources, either of a single server or a load-balanced cluster, also means that all JasperReports Server instances have identical configurations and are thus easier to deploy and maintain.
12.2 Jaspersoft OLAP in a Cluster

Jaspersoft OLAP (On Line Analytical Processing) is a module of JasperReports Server that uses different data schemas, queries, and views to perform interactive data analysis such as slicing, drill-down, and drill-through. As with JRXML and JasperReports, all of the analysis schemas, MDX queries, and analysis views are stored in the repository and accessible from any node of the cluster. Therefore, the basic configuration of Jaspersoft OLAP will run without modification in a cluster.

However, Jaspersoft OLAP is composed of two parts: an XML/A client that displays data and an XML/A provider or server that retrieves and processes the data. As described in the Jaspersoft OLAP User Guide, these two parts can run on separate instances of JasperReports server. In a cluster, how you define the XML/A connection will result in different behavior:

- In the default case, the definition of the XML/A provider is points to the localhost, so the same node that receives the user request will perform both the analysis and display it.
- You can change the definition of the provider so that it points to the URL of the load balancer for the cluster. In this case, the node that receives the user request will ask another node to retrieve the data for it, and the results will be sent back to the first node for display.

Factors such as the size of your data and the ratio of JasperReports load to OLAP load can help you determine how to configure your XML/A connections in a cluster. If you don’t perform much analysis, use the default configuration with localhost for your connections. If you have many analysis requests to the same data, a dedicated instance of Jaspersoft OLAP could provide a central cache and increase performance. Remember that connection behavior is determined by the XML/A connection defined in the repository, not by the nodes.

For simplicity, Jaspersoft recommends that every XML/A connection be configured the same way, so that XML/A connections are uniform across the cluster. However, if you have advanced analysis needs, you might benefit from having different behaviors for different XML/A connections. How to determine and configure optimal Jaspersoft OLAP performance in a cluster is beyond the scope of this document.

12.3 Session Management and Failover

Failover is the ability of the cluster to minimize the impact of a failure and continue serving clients with the remaining servers. A failure is assumed to be any unplanned incident that causes a server instance, either the software or hardware, to become and remain unavailable. In order to implement failover, you need to understand how JasperReports Server manages client sessions. This allows you to configure the cluster for optimal performance and set user expectations.

A client session is an in-memory object that represents the user to the server at run-time. After a user logs in or a web services client sends a request with credentials, the session contains the user profile such as organization and role membership for use in enforcing permissions. For browser users, the session also stores information about the state of the web interface, for example the last folder viewed in the repository, the last search term, or the data entered in a wizard. In commercial editions with the Ad Hoc Editor and Dashboard Designer, the user session also stores the on-screen state of the report or dashboard that the browser user creates interactively.

There are two types of session management in cluster design, each of which determines a different failover scenario:

- Replicated or persistent sessions – The instantaneous state of every client session is continually stored on every node or in a shared location. Upon failure, the load balancer automatically redirects client connections to a remaining node. Because
every node has access to a copy of the client session, the user can continue work from the previous state, often without even being aware of the failure or the change.

JasperReports Server does not support replicated or persistent sessions.

Unlike small, fast e-commerce sessions, JasperReports Server has larger and more complex sessions that would degrade performance when replicating or persisting. The reason for this is that the JasperPrint object for an exported report is stored in the user session. The JasperPrint object can be large (many megabytes) and would take a long time to serialize and store persistently, thereby slowing down every user action.

- Sticky or pinned sessions – Client sessions are created and managed privately by each server instance and cannot be transferred to another node. When a failure shuts down an instance, all of its client sessions are lost. When users reconnect, the load balancer directs them to a remaining node that creates its own, new session for them. As mentioned throughout this chapter, JasperReports Server implements only sticky sessions.

In the case of sticky sessions, the chance of a user experiencing a failure is the same as if working on a non-clustered server. However, the impact on users is mitigated by the fact that they can log in again on a remaining node of the cluster. Even though the risk of failure occurring is low, cluster designers should understand the impact on users.

12.3.1 Impact on Browser Users

A server failure would mainly affect long sessions where the user has unsaved work, such as in the Ad Hoc Editor and Dashboard Designer. The consequence of a failure is that users may need to re-create a report, but no critical functionality or data would be lost. If the user was only browsing the repository at the time of failure, there is no work in progress to be lost and the only consequence is the need to log in again. If the user was viewing a report, he will need to run it again after logging in. The advantage of having a cluster is that users can resume work immediately on the remaining instances.

The standard precaution against lost work is for users to save their work at regular intervals or significant milestones. Both the Ad Hoc Editor and the Dashboard Designer let users save their current view as a report or dashboard that can be opened to the same state. Work that is explicitly saved by the user is stored in the shared repository and available upon login to a new node. This includes any resources that are created, moved, or saved, as well as any Save or Save As operation that completes before a failure.

12.3.2 Impact on Web Services Clients

Web services calls can also be impacted by failure, but because they are much quicker operations that involve a request and immediate response, this is much less likely. In addition, there is no other information stored in the client session because web services calls are stateless. In the web services model, the calling application has the responsibility to remember the state of the interaction between it and the server.

Still, applications that call JasperReports Server web services should implement a reasonable timeout and verify the return value to determine if the server instance has a failure. In the case of a timeout or an error, the application should call the same operation again, which the load balancer should automatically route to a remaining server instance.

12.4 Cluster Design Process

The rest of this chapter looks how you would design a JasperReports Server cluster to fit your needs. It assumes that your cluster will follow the traditional pattern with a load balancer and some number of identical nodes, as shown in the sample architecture. Other architectures, such as dedicated OLAP nodes or a geographically distributed cluster, are possible but beyond the scope of this chapter.

As with any software project, careful design and planning will help you meet your goals. A simplified process for designing a cluster might include the following steps:

1. Gather cluster requirements in the following areas:
   - Performance – Usually defined as average response time for a given load.
   - High availability – Usually measured as percentage up-time.
   - Scalability – The ease of adding nodes to improve performance and availability over time.
2. Estimate the size of your cluster to meet your requirements within your limitations such as time and budget. Sizing determines the architecture of your cluster:
   - Load-balancing hardware.
   - Size and number of cluster nodes.
   - Shared resources, especially databases.
   - JasperReports Server configuration.

3. Deploy your cluster:
   - Hardware purchases and installation.
   - Network configuration.
   - Software configuration, including JasperReports Server configuration.
   - Testing of all components individually and in the cluster architecture.
   - Rollout to end-users.
   - Administration, maintenance, and scaling procedures.

Deployment and implementation are beyond the scope of this chapter. The following sections give more details about gathering cluster requirements and sizing.

12.5 Performance Requirements

There are two sides to performance: performance requirements and load estimates. The goal is to anticipate the number of users and their activities, so that you can design a cluster that responds to their needs with minimal delay.

Performance requirements ensure that the cluster is responsive to user requests. Such requirements are usually defined in terms of the system response relative to an amount of traffic in a given period, like “5 second response for all pages regardless of system load” or “maximum 10 second response time for up to 20 simultaneous users”. There are often different requirements for expected average load versus maximum load.

The requirements should be based on realistic estimates of the number of users and how they will interact with the features of JasperReports Server. First, you should determine what volume of users will be browser clients (real people) and web services clients (applications making API calls) and the ratio between them. See section 12.3, “Session Management and Failover,” on page 200 for information about these two types of clients and how their client sessions differ.

Then, you must estimate what kinds of operations your users perform, for example:
   - How many users will just run reports and save the output?
   - How many users will create reports or explore data interactively?
   - How large are your typical reports, in terms of data retrieved and processing required, and how often do they run?
   - What times of day will have the highest user load?
   - Will users or web services clients access the repository extensively?

In addition to user sessions, the server instances must also process scheduled jobs, so you should estimate their volume and nature. For example, what volume of jobs are critical to run at exact times, what volume of jobs must run during business hours when user load will be high? Can you educate users to run jobs outside of business hours?

Long jobs contribute to server load and may slow down user sessions. If the scheduled job load is very high at the same time as the user load, it may be desirable to configure a dedicated server instance to run jobs. This advanced cluster architecture is beyond the scope of this document. For more information about running scheduled jobs, see section 12.1.5, “Job Schedulers,” on page 199.

Client connectivity is also an issue, because the slowest link in the network creates a bottleneck. There is a difference in system performance and scalability between users who work remotely over DSL and those who work in the office on your corporate T1 network.
12.6 Availability Requirements

High availability is the ability of the cluster to effectively avoid downtime. In general, failover prevents total system unresponsiveness that would happen if a single server failed, but a properly designed cluster must also address the failure of other cluster components.

High availability is usually defined in terms of uptime, like 99.999%, 24/7/365 (always), or business hours during business days. To design for high availability, all system components must be made redundant or recoverable enough so that no single component can fail and bring the entire environment to a stop in a way that violates the high availability requirements.

Identifying the failure modes of a cluster is good practice for any deployment. For every component in the cluster, analyze what happens to users and overall availability when that component fails. Section 12.3, “Session Management and Failover,” on page 200 explains what happens to users on a server instance that fails, but you should also consider the overall performance degradation of the cluster and the impact of failures in other cluster components.

A further aspect of high availability is sizing the cluster so that should one server (hardware or software) become unavailable, the rest can still respond to anticipated user demand. If you have strict availability and performance requirements, you may need to plan for additional nodes. As you design your JasperReports Server cluster, you should take all these issues into account in order to properly express your availability requirements.

Another aspect of high availability is the ability of certain clusters to keep operating during planned maintenance and upgrades, called rolling upgrades. JasperReports Server does not support rolling upgrades, because different versions can require different schemas in the shared repository. So your availability planning must include the time to stop the entire cluster and perform upgrades as necessary.

12.7 Scalability Requirements

Scalability refers to the ability of the environment to meet the needs of an increasing number of users and external services in a way that is predictable in terms of performance.

The main scalability consideration is whether the cluster architecture is dynamic and additional servers can be added to increase the number of users (simultaneous or not). Additionally, you should consider whether other components of the cluster can be added to the design later. It may be simple to expand the size of the shared repository because database servers are usually made to scale, but it may be more complex to change load balancers or implement redundant load balancers.

There is often a relationship between scalability and high availability. If the system works under normal circumstances but stops functioning under load, this may violate your availability requirements. You should also consider whether availability requirements will change along with increased user load. For example, as the cluster-based BI solution is rolled out to more and more users, high-availability of the cluster becomes more critical. In this case, you may need to add redundant load-balancers or other component upgrades to increase availability in the future. Your initial cluster design should recognize this need allow for this expansion.

Once you have defined your scalability requirements in detail, use this information when sizing your cluster.

12.8 Sizing a Cluster

Sizing a cluster is the process of determining the number and architecture of components to meet your performance, availability, and scalability requirements. During this phase, you also need to perform load tests to determine the best configuration for your needs.

For a traditional cluster, your design should specify the following:

- Load-balancing hardware, software, and policies.
- Size and number of cluster nodes, with characteristics such as processors and memory.
- JasperReports Server configuration optimizing for the user load and cluster environment.
- Shared resources, especially databases for the repository and data sources.
- Network service levels or upgrades.
12.8.1 Load Balancer

Determine whether you need a dedicated hardware load-balancer or whether a basic software load-balancer will meet your requirements. You should also specify the load-balancing techniques you will use, such as round-robin, load-based, or some other configuration. If the load balancer communicates with the nodes to optimize traffic, specify how this happens.

Much of the load balancer requirement is based on the size and complexity of your cluster. A small cluster of two to three nodes can use a software load-balancer with a simple algorithm. But if you have many nodes or a few powerful nodes with high traffic, you need a dedicated load-balancer for the cluster.

If you have very strict performance requirements and need to closely monitor the load across the cluster, you may need a load-based balancer that communicates with the nodes. The trade-offs for these advanced load-balancing techniques are more maintenance and a more complex configuration on each node.

And finally, if the cluster is intended to scale over time, the load balancer must be able to handle more incoming traffic and more cluster nodes.

12.8.2 Cluster Nodes

The size and number of nodes is the determinant of your cluster design. Based on your expected load, you need to specify enough processing power to meet your performance and availability requirements. This includes hardware specifications such as processors and memory, or their equivalent for virtual servers.

One of the design decisions you must make for hardware is whether to have few instances on powerful hardware or many instances on cheaper hardware. If high availability is a key requirement, having more instances decreases the risk and impact of any one failure. Other issues such as maintenance and cost must also be factored into this decision.

Hardware availability can be another issue. Does your budget include the new servers, either real or virtual, to handle the loads you expect? Or can you reuse existing hardware. Remember that having mis-matched hardware is possible in the cluster, but it complicates the server configuration and may lead to sub-optimal load-balancing.

Because the node architecture is the key to cluster, running tests with various options can help you choose the right hardware. By performing load tests, you can determine how many users can run on a single node and scale the number of nodes accordingly. You can also run tests with various numbers of processors and memory to determine what configuration is optimal for your expected user load. If you plan on using virtual servers, performing load tests can help uncover any issues with connectivity and stability.

12.8.3 Software Configuration

Once you have the server hardware to handle your user load, you need to determine the optimal software configuration for your nodes. This includes:

- JVM settings to optimize processor and memory usage.
- Application server settings, mostly to provide connection pooling.
- JasperReports Server settings.

Server settings can help optimize performance based on the types and number of reports that you expect users to run:

- Data policies in Ad Hoc can help speed up Domain-based reports
- Cache settings in Ad Hoc can boost performance when there are many users accessing the same data.
- Query limits (for Ad Hoc reports and for JasperReports) can help prevent slow responses when users request huge data sets during peak usage times.
- Custom virtualization settings can help servers deal with large reports (greater than 300 pages).

For JVM setting to optimize performance, see the JasperReports Server Installation Guide. For details about query limits, data policies, and cache settings, see the chapter on configuration in the JasperReports Server Administrator Guide.

Also turning off features such as auditing and logging can improve performance in highly-loaded machines.
12.8.4 Databases

JasperReports Server relies heavily on database access, and cluster deployments add extra load to these shared resources:

- The shared repository is a critical part of the cluster with very high loads compared to a traditional single-server deployment. With multiple simultaneous connections from multiple nodes, you must ensure that the repository database doesn’t become a bottleneck. Therefore, the architecture, size, and speed of the database that hosts the shared repository must be carefully evaluated and specified in the cluster design.

- A JasperReports Server cluster enables and encourages a large population of users to process more and more data—this is a good thing. But the cluster may stress your reporting databases to their limits if they aren’t considered and given adequate hardware and software. This is another case where knowing what data your users access the most can help you optimize database configuration such as indexing.

- If you implement external authentication or single sign-on, make sure those resources can handle the load that the cluster is expected to generate.

12.8.5 Network

Finally, your cluster architecture may need to take into consideration the availability and quality of your network. How remote are your users and what kind of connection do they have to the cluster?

There are two main concerns about your network:

- Network availability and capacity affects the availability and scalability of the cluster to the users. If you have strict availability requirements, you may need redundant network connections from different providers. For scalability, your network needs to handle the load of the planned maximum number of users, or have plans to scale the network along with the number of users. Be aware that this creates external dependencies on meeting your performance requirements.

- Network bandwidth can affect server load by slowing down individual connections. JasperReports Server can generate multiple large documents simultaneously that are sent to the web browser or web services clients. Network capacity (megabytes per second) is critical to being able to deliver these generated documents in a reasonable time frame. The slower the network, the more load on the servers generating the report documents, as they will take longer to deliver the same content, and potentially lead to more simultaneous report requests.

12.8.6 Policies and Procedures

Given the complexity of a cluster design, it’s a good idea to document your design process and your final architecture. Ongoing maintenance is simpler if you have a record of decisions and document procedures for configuration.

And finally, to meet scalability requirements, you may need monitor your cluster performance and define some metric for adding nodes. For example, you might specify one node per 100 concurrent users, or add a node when peak load reaches 90% on every node.

Sizing a cluster is usually the last phase of cluster design, before you start implementing your chosen components, creating a cluster prototype, and going into production. When all components are sized appropriately for the many requirements and usage conditions, testing and rollout of your cluster will proceed more smoothly.

In conclusion, a cluster of JasperReports Server instances can help you meet the high availability and scalability requirements of your BI solution. This chapter is only meant as a guideline to help you in your design and planning phases. If you want technical assistance, Jaspersoft Professional Services can help in all phases of designing and rolling out a successful cluster.
GLOSSARY

Ad Hoc Editor
The interactive report designer in JasperReports Server Professional and Enterprise editions. Starting from a collection of fields predefined in a Topic or selected from a Domain, the Ad Hoc Editor lets you drag and drop report elements to draft, preview, and finalize reports. Like JRXML reports, Ad Hoc reports can be run, printed, and scheduled within JasperReports Server. In addition, Ad Hoc reports may be reopened in the Ad Hoc Editor, further modified, and saved.

Audit Archiving
To prevent audit logs from growing too large to be easily accessed, the system installer configures JasperReports Server to move current audit logs to an archive after a certain number of days, and to delete logs in the archive after a certain age. The archive is another table in the JasperReports Server’s private database.

Audit Domains
A Domain that accesses audit data in the repository and lets administrators create Ad Hoc reports of server activity. There is one Domain for current audit logs and one for archived logs.

Audit Logging
When auditing is enabled, audit logging is the active recording of who used JasperReports Server to do what when. The system installer can configure what activities to log, the amount of detail gathered, and when to archive the data. Audit logs are stored in the same private database that JasperReports Server uses to store the repository, but the data is only accessible through the audit Domains.

Auditing
A feature of JasperReports Server Enterprise edition that records all server activity and allows administrators to view the data.

Calculated Field
In a Domain, a field whose value is calculated from a user-written formula that may include any number of fields, operators, and constants. A calculated field is defined in the Domain Designer, and it becomes one of the items to which the Domain’s security file and locale bundles can apply.

CRM
Customer Relationship Management. The practice of managing every facet of a company’s interactions with its clientele. CRM applications help businesses track and support their customers.

CrossJoin
An MDX function that combines two or more dimensions into a single axis (column or row).
**Cube**

The basis of most OLAP applications, a cube is a data structure that contains three or more dimensions that categorize the cube’s quantitative data. When you navigate the data displayed in an OLAP view, you are exploring a cube.

**Custom Field**

In the Ad Hoc Editor, a field that is created through menu items as a simple function of one or two available fields, including other custom fields. When a custom field becomes too complex or needs to be used in many reports, it is best to define it as a calculated field in a Domain.

**Dashboard**

A collection of reports, input controls, graphics, labels, and web content displayed in a single, integrated view. Dashboards often present a high level view of your data, but input controls can parameterize the data to display. For example, you can narrow down the data to a specific date range. Embedded web content, such as other web-based applications or maps, make dashboards more interactive and functional.

**Derived Table**

In a Domain, a derived table is defined by an additional query whose result becomes another set of items available in the Domain. For example, with a JDBC data source, you can write an SQL query that includes complex functions for selecting data. You can use the items in a derived table for other operations on the Domain, such as joining tables, defining a calculated field, or filtering. The items in a derived table can also be referenced in the Domain’s security file and locale bundles.

**Data Policy**

In JasperReports Server, a setting that determines how the server processes and caches data used by Ad Hoc reports. Select your data policies by clicking Manage > Ad Hoc Settings.

**Data Source**

Defines the connection properties that JasperReports Server needs to access data. The server transmits queries to data sources and obtains datasets in return for use in filling reports and previewing Ad Hoc reports. JasperReports Server supports JDBC, JNDI, and Bean data sources; custom data sources can be defined as well.

**Dataset**

A collection of data arranged in columns and rows. Datasets are equivalent to relational results sets and the JRDataSource type in the JasperReports Library.

**Datatype**

In JasperReports Server, a datatype is used to characterize a value entered through an input control. A datatype must be of type text, number, date, or date-time. It can include constraints on the value of the input, for example maximum and minimum values. As such, a datatype in JasperReports Server is more structured than a datatype in most programming languages.

**Denormalize**

A process for creating table joins that speeds up data retrieval at the cost of having duplicate row values between some columns.

**Dice**

An OLAP operation to select columns.

**Dimension**

A categorization of the data in a cube. For example, a cube that stores data about sales figures might include dimensions such as time, product, region, and customer’s industry.

**Domain**

A virtual view of a data source that presents the data in business terms, allows for localization, and provides data-level security. A Domain is not a view of the database in relational terms, but it implements the same functionality within JasperReports Server. The design of a Domain specifies tables in the database, join clauses, calculated fields, display names, and default properties, all of which define items and sets of items for creating Ad Hoc reports.
Domain Topic
A Topic that is created from a Domain by the Data Chooser. A Domain Topic is based on the data source and items in a Domain, but it allows further filtering, user input, and selection of items. Unlike a JRXML-based Topic, a Domain Topic can be edited in JasperReports Server by users with the appropriate permissions.

Drill
To click on an element of an OLAP view to change the data that is displayed:

- **Drill down.** An OLAP operation that exposes more detailed information down the hierarchy levels by delving deeper into the hierarchy and updating the contents of the navigation table.
- **Drill through.** An OLAP operation that displays detailed transactional data for a given aggregate measure. Click a fact to open a new table beneath the main navigation table; the new table displays the low-level data that constitutes the data that was clicked.
- **Drill up.** An OLAP operation for returning the parent hierarchy level to view to summary information.

Eclipse
An open source Integrated Development Environment (IDE) for Java and other programming languages, such as C/C++.

ETL
Extract, Transform, Load. A process that retrieves data from transactional systems, and filters and aggregates the data to create a multidimensional database. Generally, ETL prepares the database that your reports will access. The Jaspersoft ETL product lets you define and schedule ETL processes.

Fact
The specific value or aggregate value of a measure for a particular member of a dimension. Facts are typically numeric.

Field
A field is equivalent to a column in the relational database model. Fields originate in the structure of the data source, but you may define calculated fields in a Domain or custom fields in the Ad Hoc Editor. Any type of field, along with its display name and default formatting properties, is called an item and may be used in the Ad Hoc Editor.

Frame
A dashboard element that displays reports or custom URLs. Frames can be mapped to input controls if their content can accept parameters.

Group
In a report, a group is a set of data rows that have an identical value in a designated field.

- In a table, the value appears in a header and footer around the rows of the group, while the other fields appear as columns.
- In a chart, the field chosen to define the group becomes the independent variable on the X axis, while the other fields of each group are used to compute the dependent value on the Y axis.

Hierarchy Level
In an OLAP cube, a member of a dimension containing a group of members.

Input Control
A button, check box, drop-down list, text field, or calendar icon that allows users to enter a value when running a report or viewing a dashboard that accepts input parameters. For JRXML reports, input controls and their associated datatypes must be defined as repository objects and explicitly associated with the report. For Domain-based reports that prompt for filter values, the input controls are defined internally. When either type of report is used in a dashboard, its input controls are available to be added as special content.

iReport Designer
An open source tool for graphically designing reports that leverage all features of the JasperReports Library. The Jaspersoft iReport Designer lets you drag and drop fields, charts, and sub-reports into a canvas, and also define parameters or expressions for each object to create pixel-perfect reports. iReport Designer outputs the JRXML of the report or uploads it directly to JasperReports Server.
Item
When designing a Domain or creating a Topic based on a Domain, an item is the representation of a database field or a calculated field along with its display name and formatting properties defined in the Domain. Items can be grouped in sets and are available for use in the creation of Ad Hoc reports.

JasperReports Library
An embeddable, open source, Java API for generating a report, filling it with current data, drawing charts and tables, and exporting to any standard format (HTML, PDF, Excel, CSV, and others). JasperReports processes reports defined in JRXML, an open XML format that allows the report to contain expressions and logic to control report output based on run-time data.

JasperReports Server
A commercial open source, server-based application that calls the JasperReports library to generate and share reports securely. JasperReports Server authenticates users and lets them upload, run, view, schedule, and send reports from a web browser. Commercial versions provide metadata layers, interactive report and dashboard creation, and enterprise features such as organizations and auditing.

Jaspersoft ETL
A graphical tool for designing and implementing your data extraction, transforming, and loading (ETL) tasks. It provides hundreds of data source connectors to extract data from many relational and non-relational systems. Then, it schedules and performs data aggregation and integration into data marts or data warehouses that you use for reporting.

Jaspersoft OLAP
A relational OLAP server integrated into JasperReports Server that performs data analysis with MDX queries. The product includes query builders and visualization clients that help users explore and make sense of multidimensional data. Jaspersoft OLAP also supports XML/A connections to remote servers.

JavaBean
A reusable Java component that can be dropped into an application container to provide standard functionality.

JDBC
Java Database Connectivity. A standard interface that Java applications use to access databases.

JNDI
Java Naming and Directory Interface. A standard interface that Java applications use to access naming and directory services.

Join Tree
In Domains, a collection of joined tables from the actual data source. A join is the relational operation that associates the rows of one table with the rows of another table based on a common value in given field of each table. Only the fields in a same join tree or calculated from the fields in a same join tree may appear together in a report.

JPivot
An open source graphical user interface for OLAP operations. For more information, visit http://jpivot.sourceforge.net/.

JRXML
An XML file format for saving and sharing reports created for the JasperReports Library and the applications that use it, such as iReport Designer and JasperReports Server. JRXML is an open format that uses the XML standard to define precisely all the structure and configuration of a report.

MDX
Multidimensional Expression Language. A language for querying multidimensional objects, such as OLAP (On Line Analytical Processing) cubes, and returning cube data for analytical processing. An MDX query is the query that determines the data displayed in an OLAP view.
Measure
Depending on the context:
- In a report, a formula that calculates the values displayed in a table’s columns, a crosstab’s data values, or a chart’s dependent variable (such as the slices in a pie).
- In an OLAP view, a formula that calculates the facts that constitute the quantitative data in a cube.

Mondrian
A Java-based, open source multidimensional database application.

Mondrian Connection
An OLAP client connection that consists of an OLAP schema and a data source used to populate an OLAP view.

Mondrian Schema Editor
An open source Eclipse plug-in for creating Mondrian OLAP schemas.

Mondrian XMLA Source
A server-side XMLA source definition of a remote client-side XML/A connection used to populate an OLAP view using the XMLA standard.

MySQL
An open source relational database management system. For information, visit http://www.mysql.com/.

Navigation Table
The main table in an OLAP view that displays measures and dimensions as columns and rows.

ODBO Connect
Jaspersoft ODBO Connect enables Microsoft Excel 2003 and 2007 Pivot Tables to work with Jaspersoft OLAP and other OLAP servers that support the XML/A protocol. After setting up the Jaspersoft ODBO data source, business analysts can use Excel Pivot Tables as a front-end for OLAP analysis.

OLAP
On Line Analytical Processing. Provides multidimensional views of data that help users analyze current and past performance and model future scenarios.

OLAP Client Connection
A definition for retrieving an OLAP view. An OLAP client connection is either a direct Java connection (Mondrian connection) or an XML-based API connection (XML/A connection).

OLAP Schema
A metadata definition of a multidimensional database. In Jaspersoft OLAP, schemas are stored in the repository as XML file resources.

OLAP View
Also called an analysis view. A view of multidimensional data that is based on an OLAP client connection and an MDX query. It is the entry point to analysis operations, such as slice and dice, drill down, and drill through.

Organization
A set of users that share folders and resources in the repository. An organization has its own user accounts, roles, and root folder in the repository to securely isolate it from other organizations that may be hosted on the same instance of JasperReports Server.

Organization Admin
Also called the organization administrator. A user in an organization with the privileges to manage the organization’s user accounts and roles, repository permissions, and repository content. An organization admin can also create sub-organizations and manage all of their accounts, roles, and repository objects. The default organization admin in each organization is the jasperadmin account.
Outlier
A fact that seems incongruous when compared to other member’s facts. For example, a very low sales figure or a very high number of helpdesk tickets. Such outliers may indicate a problem (or an important achievement) in your business. The analysis features of Jaspersoft OLAP excel at revealing outliers.

Parameter
Named values that are passed to the engine at report-filling time to control the data returned or the appearance and formatting of the report. A report parameter is defined by its name and type. In JasperReports Server, parameters can be mapped to input controls that users can interact with.

Pivot
To rotate a crosstab such that its row groups become column groups and its column groups become rows. In the Ad Hoc Editor, pivot a crosstab by clicking ⬇️.

Pivot Table
A table with two physical dimensions (for example, X and Y axis) for organizing information containing more than two logical dimensions (for example, PRODUCT, CUSTOMER, TIME, and LOCATION), such that each physical dimension is capable of representing one or more logical dimensions, where the values described by the dimensions are aggregated using a function such as SUM. Pivot tables are used in Jaspersoft OLAP.

Properties
Settings associated with an object. The settings determine certain features of the object, such as its color and label. Properties are normally editable. In Java, properties can be set in files listing objects and their settings.

Repository
The tree structure of folders that contain all saved reports, dashboards, OLAP views, and resources. Users access the repository through the JasperReports Server web interface or through iReport. Applications can access the repository through the web service API. Administrators use the import and export utilities to back up the repository contents.

Resource
In JasperReports Server, anything residing in the repository, such as an image, file, font, data source, Topic, Domain, report element, saved report, report output, dashboard, or OLAP view. Resources also include the folders in the repository. Administrators set user and role-based access permissions on repository resources to establish a security policy.

Role
A security feature of JasperReports Server. Administrators create named roles, assign them to user accounts, and then set access permissions to repository objects based on those roles. Certain roles also determine what functionality and menu options are displayed to users in the JasperReports Server interface.

Schema
A logical model that determines how data is stored. For example, the schema in a relational database is a description of the relationships between tables, views, and indexes. In Jaspersoft OLAP, an OLAP schema is the logical model of the data that appears in an OLAP view; they are uploaded to the repository as resources. For Domains, schemas are represented in XML design files.

Schema Workbench
A graphical tool for easily designing OLAP schemas, data security schemas, and MDX queries. The resulting cube and query definitions can then be used in Jaspersoft OLAP to perform simple but powerful analysis of large quantities of multi-dimensional data stored in standard RDBMS systems.

Set
In Domains and Domain Topics, a named collection of items grouped together for ease of use in the Ad Hoc Editor. A set can be based on the fields in a table or entirely defined by the Domain creator, but all items in a set must originate in the same join tree. The order of items in a set is preserved.
Slice
An OLAP operation for filtering data rows.

SQL
Structured Query Language. A standard language used to access and manipulate data and schemas in a relational database.

System Admin
Also called the system administrator. A user who has unlimited access to manage all organizations, users, roles, repository permissions, and repository objects across the entire JasperReports Server instance. The system admin can create root-level organizations and manage all server settings. The default system admin is the superuser account.

Topic
A JRXML file created externally and uploaded to JasperReports Server as a basis for Ad Hoc reports. Topics are created by business analysts to specify a data source and a list of fields with which business users can create reports in the Ad Hoc Editor. Topics are stored in the Ad Hoc Components folder of the repository and displayed when a user launches the Ad Hoc Editor.

Transactional Data
Data that describe measurable aspects of an event, such as a retail transaction, relevant to your business. Transactional data are often stored in relational databases, with one row for each event and a table column or field for each measure.

User
Depending on the context:
• A person who interacts with JasperReports Server through the web interface. There are generally three categories of users: administrators who install and configure JasperReports Server, database experts or business analysts who create data sources and Domains, and business users who create and view reports and dashboards.
• A user account that has an ID and password to enforce authentication. Both people and API calls accessing the server must provide the ID and password of a valid user account. Roles are assigned to user accounts to determine access to objects in the repository.

WCF

Web Services
A SOAP (Simple Object Access Protocol) API that enables applications to access certain features of JasperReports Server. The features include repository, scheduling and user administration tasks.

XML
eXtensible Markup language. A standard for defining, transferring, and interpreting data for use across any number of XML-enabled applications.

XML/A
XML for Analysis. An XML standard that uses Simple Object Access protocol (SOAP) to access remote data sources. For more information, see http://www.xmla.org/

XML/A Connection
A type of OLAP client connection that consists of Simple Object Access Protocol (SOAP) definitions used to populate an OLAP view.
A
access control
   administering 68
   and Domains 77
authentication 62
examples 70
overview 61
passwords 62
permissions 66, 73
recommendations for Domains 91
repository 64
roles 64, 80
Spring Security 62
action codes
   Ad Hoc Editor HTTP API 146–149
dashboard HTTP API 150–152
   valid values for action code parameters 145
Action Model 153
Ad Hoc Editor
   administering 38
   and Ad Hoc launcher 134
   and CSS 38
   and iReport 122
   and types of report 27
   as a data explorer 19
   as a report designer 19
cascading style sheets 38
charts 22, 30
creating reports 25
crosstabs 32
custom fields 35
formatting with styles 28
grouping data 29, 31, 32
HTTP API 145
JasperReports Server plug-in for iReport 122
measures 31, 33
running reports 37
saving reports 37
summarizing data 29, 33, 34
tables 20, 28
testing Domain security 87
tool bar 20
user interface 20
using 26
Ad Hoc Launcher API 134, 138
adding
   custom data sources 129
   reports, using iReport 115
   roles 64
administering
   authorization 68
   configuration 68
customizing the user interface 168
default configuration of system authorization 68, 73
Domain security 77
JasperReports Server configuration 41
memory usage 42
reports 38
roles 64
Spring Security 61
styles 38
users 65
administering JasperReports Server
custom data sources 43
data sources 43
repository 43
AND 89
anomalies 19
Ant 45
APIs
   Ad Hoc Editor HTTP API 145
   Ad Hoc Launcher 134, 138
   content files 128
Dashboard Designer HTTP API 149
flushing the analysis cache 134
folders 126
Java 123
metadata resources 127
OLAP connection 133
public API 123
report scheduling 129, 132
Repository HTTP API 142
Repository Java API 125
singletons 124
types 123
application security 101
attributes, profile. See profile attributes.
B
browseDB 138
business case, CZS 78
C
cache, flushing analysis data 134
calculated fields 35
cascading input controls 51
cascading style sheets
customizing 153, 156
in Ad Hoc Editor 38
chart reports
customizing default report styles 41
interface elements 23, 24
custom fields 35
custom queries in Ad Hoc launchers 134
customizing
cascading style sheets 156
colors 156
fonts 156
logo 154
reports 168, 188
user interface 153, 159, 168
CZS 78
D
dashboards
action codes 150
Dashboard Designer HTTP API 149
refresh interval 15
working with 13–17
data formats 35
data security 81
data sources
See also custom data sources.
custom 129
Domains 113
in iReport 112
JasperReports Library and JasperReports Server 43
query executers 44
database administrators
and Domain security 82
and profile attributes 82
datatypes, in iReport 120
dates in custom fields 36
decorator.jsp 163
decorators.xml 162
delimiters 99
documentation
guide to using Ultimate Guide 10
premium documentation 11
standard documentation 11
Domains
and access control 77
best practices 91
complex 91
creating 38
data sources 113
example 80
example design file 92
example security file 87, 96
in Ad Hoc Editor 38
performance 91
plug-in for iReport Professional 110, 113
recommendations 91
testing 90
Topics based on 92
using Groovy for security 83, 88
drill down functionality in iReport 121

E
editing
custom fields 35
Java methods 185
JRXML files 39, 117
styles 39
examples
CZS business case 78
Domain design 80, 92
Domain security file 87, 96
fields 79
joins 79
report 90
roles 80
tables 79
users 80
exclude 33
exploring data 19

F
files, in iReport 114
filter expressions 87, 98
filters
and improving scalability 42
creating in charts 30
creating in crosstabs 33
creating in tables 29, 30
in the Ad Hoc Editor 21
folders
accessing with API 126
in iReport 111
fonts, customizing 156
formulas 35

G
Groovy
expressions 83, 88
language 87
resources about 92
guide to using Ultimate Guide 10

H
HTTP APIs
See also action codes.
Ad Hoc Editor 145
Dashboard Designer 149
Repository 142
HTTPS 103

I
images 117
importing JDBC connection 115
input controls
cascading 51
in iReport 119
parameters for cascading input controls 57, 58
parameters for query-based input controls 57
query-based 51
installing JasperReports Server 11
interfaces
public API 123
singletons 124
iReport
and Ad Hoc Editor 122
compatibility issues 110
configuration 110
data sources 112, 113
datatypes 120
Domain plug-in for iReport Professional 110, 113
downloading JRXML files 39
drill down functionality 121
files 114
folders 111
input controls 119
JasperReports Server plug-in 109
JRXML files 117
lists 120
queries 120
report units 114
resources 111
running reports 117
server connection 110
item groups 91
itemAccessGrant 89
itemGroupAccessGrantList 89

J
JasperAnalysis. Now called Jaspersoft OLAP.
JasperReports 9
JasperReports Library, data sources 43
JasperReports Server
business intelligence platform 9
configuration 41
installation 11
memory usage 42
plug-in for iReport 58
public API 123
Jaspersoft 9
Jaspersoft OLAP
flushing the analysis cache 134
OLAP connection API 133
public API 123
SuperMart demo 65
Java APIs 123
Java interfaces 123
Java Virtual machine (JVM) 91
JavaScript 153
JavaServer Pages 153, 163
JDBC
connection in iReport 115
JasperReports Server plug-in for iReport 115
JDK and JVM 45
joins 79, 91
JRXML files
and Topics 38
creating 38, 115
define report styles 39
downloading 39
in iReport 117
in report units 114
JasperReports Server plug-in for iReport 117
K
keep only 33
keystore 102
L
launcher 134, 138
lists, in iReport 120
locale 144
log4j 86
logging 86
logo, customizing 154
M
main.jsp 163
memory usage 42
metadata resources 125, 127
MySQL 78
O
OLAP connection API 133
OR 89
orMultipleExpressions 89
outliers 19
P
passwords
and filter beans 62
and Spring Security 63
changing your password 12
default users and passwords 65
PKC12 keystore 102
plug-ins
Domain plug-in for iReport Professional 110, 113
JasperReports Server plug-in for iReport 58, 109
PostgreSQL 82
principal expressions 87
profile attributes
creating 84
CZS example 78, 87
domain security 81, 87
filter expressions 98
in user account 81
query-based input controls 55
testProfileAttribute 98
variable substitution 82
protection domains 104
public API
and Spring Framework 124
overview 123
Q
queries
and custom data sources 44
in Ad Hoc launchers 134, 139
in iReport 120
parameters for cascading input controls 57
parameters for query-based input controls 57
query-based input controls 51
R
redo, in Ad Hoc Editor 20
reference syntax. See repository.
referencing resources
in iReport 117
in reports 117
report developer 10
report units
and repository organization 64
creating 115
in iReport 114
uploading referenced resources 118
reports
administering 38
and Ad Hoc launchers 134, 142
and Domains 113
and iReport 117
and the Ad Hoc Editor 25
and Topics 26
chart reports 22, 30
creating 25
crosstab reports 23, 24, 32
customizing 41, 168, 188
designing 19
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
drill down functionality 121
example 90
images 117
overview 25
referencing resources 117
report scheduling API 129, 132
report type 27
running 117, 168
saving 37
Index

style templates 117
subreports 117
table reports 20, 28
testing 117
repository
access control 64
administering 43
data sources 43
JasperReports Server plug-in for iReport 122
Java API 125
syntax for referencing resources 117
resourceAccessGrant 87
resources
ccontent files 128
files 114
folders 111, 126
in iReport 111
internally-referenced resources 119
metadata resources 127
referred by reports 117
types 64
roles
access control 64
administering 64
and Domain security 80
conflicting 64
creating 64
default roles 65, 69
example 80
overview 64
row-level security 81, 87, 88, 90
running reports
in iReport 117
in the Ad Hoc Editor 37
in the Ad Hoc launcher 142
testing 117
with cascading input controls 60
S
saving
JRXML files 39
reports 37
user profiles 74
scheduling 129, 132
Secure Sockets Layer. See SSL.
security
access control 65
application 101
cookies 104
Domains 77
HTTP 103
keystore 102
protection domains 104
Spring Security 61
SSL 101
Tomcat 101
web application 101
Security Manager 104
singletons 124
SiteMesh 153, 159, 161
sitemesh.xml 162
slice 33
Spring Framework 153
Spring MVC 153
Spring Security
administering 61
authentication 62
flow files 63, 143
overview 61
Spring Framework 124
user names and passwords 63
WebFlowUpgrader 63, 143
Spring Webflow 153
SSL 101
startup URI 144
styles
administering 38
and cascading style sheets 38
and JRXML files 39
changing the thumbnail 39
editing a cascading style sheet 39
editing a JRXML file 39
in the Ad Hoc Editor 28
renaming a style 39
subreports 117
SuperMart demo 65, 66
syntax for referencing resources 117
system administrator 10
system developer 10
T
table reports
action codes for Ad Hoc Editor HTTP API 146
creating 28
interface elements 20
tables 79
technical business analyst 10
testing Domain security 87, 90
testProfileAttribute 98
themes 153
TLS. See SSL.
Tomcat 101, 138
tool bar 20
Topics
and Ad Hoc launcher 134
and Domains 92
and reports 26
based on JRXML files 38
editing 26
in Ad Hoc Editor 38
trends 19
troubleshooting 86
undo, in Ad Hoc Editor 20
user interface
   Ad Hoc Editor 20
   Ad Hoc launchers 134
   and SiteMesh 159
   customizing 153, 168
   default configuration 159
user types
   database administrator 10
   report developer 10
   system administrator 10
   system developer 10
   technical business analyst 10
users
   access control 65
   administering 65
   and filter beans 62
   and profile attributes 81
   and Spring Security 63
   and the Java API 85
   default users and passwords 65
   example 70, 80
   saving 74
using the Ad Hoc Editor
   charts 22, 30
creating reports 25, 87
   crosstabs 23, 24, 32
   drill to detail 34
   filtering 21
   for exploration 19
   formatting with styles 28
   grouping data 29, 31, 32
   measures 31, 33
   running reports 37
   saving reports 37
   slice 33
   sorting 29
   summarizing data 29, 33, 34
   tables 20, 28
   to analyze data 19
   to design reports 19
   with Ad Hoc launcher 134, 142
variable substitution 82
web.xml 162
WebFlowUpgrader 63
XML 77, 92, 96
XMLBuddy 92