



JasperReports® Server

Ultimate Guide

Version 9.0.0 | January 2024

Contents

Contents	2
Introduction	7
Community and Commercial Editions	8
User Descriptions and Document Maps	9
Technical Business Analyst	9
Report Developer	9
System Developer	10
System Administrator and Database Administrator	10
Getting Started	10
Customizing Reports	12
The Report Viewer and Conditional Text	12
Creating a New Report Template	15
Custom Data Sources	20
Pre-installed Data Source Types	21
Enabling the Pre-installed Data Source Types	22
Understanding the Pre-installed Data Sources	24
Custom Data Source Examples	25
Prerequisites	25
Installing the Custom Data Source Examples	26
Custom Bean Data Source	27
Webscraper Custom Data Source	27
Hibernate Custom Data Source	29
Custom Data Source Pro	30
Custom Data Source Architecture	31
Server Startup	31

Creation of a Custom Data Source	31
Instantiation of a Custom Data Source	32
Query Executors	33
Creating a Custom Data Source	34
Writing Java Classes	35
Defining the Custom Data Source in Spring	40
Defining the Message Catalog	43
Adding the Custom Query Language to the UI	44
Installing a Custom Data Source Type	45
Using a Custom Data Source Type	46
JasperReports Server APIs	47
REST API	47
Visualize.js API	48
Repository HTTP API	50
General Parameters	51
Executing ReportUnits	52
Executing Dashboards	54
Input Control Parameters for Reports and Dashboards	56
Linking to Content	58
Viewing Resources in the Repository	58
The Public JasperReports Server API	58
Accessing API Implementations Using the Spring Framework	59
Repository API	62
Engine Service	67
Report Data Source Service API	68
Report Scheduling API	69
Users and Roles API	77
Object Permissions API	78
OLAP Connection API	79
Flushing the OLAP Cache Using the API	80

Customizing the User Interface	82
Changing the UI With Themes	83
Changing the Logo and Favicon	84
Changing Colors and Fonts	86
Hiding UI Elements	87
Using Themes to Remove Branding	88
Replacing the Default Theme	90
Customizing the UI With Web App Files	91
Location of Interpreted Files	92
Location of JSP Files	93
Customizing JavaScript Source Code	94
Customizing WAR Files	95
Reloading the JasperReports Server Web App	95
Customizing the Branding with SiteMesh	96
web.xml	97
sitemesh.xml and decorators.xml	97
main.jsp and decorator.jsp	98
Editing decorator.jsp for Rebranding	99
Setting the Home Page	101
Adding View List Buttons to the Getting Started Page	103
Restricting Access to a Location in the Repository	104
Customizing the JasperReports Wizard Repository Trees	106
Customizing Menus	107
Removing a Menu Item	108
Restricting Access by Role	110
Adding an Item to the Main Menu	115
Adding a New Main Menu	118
Changing Other Menus	120
Action Model Reference	121
Customizing the Report Rendering Page	130
Customizing Input Controls	131

Customizing the Report Viewer	134
Working With Custom Java Classes	137
Adding a Custom JSP Page in a Spring Web Flow	138
Customizing the Scheduler Pages	147
Customizing Display of Available Outputs	147
Disabling output formats	149
Upgrading With UI Customizations	149
Designing a Cluster	151
Sample Cluster Architecture	152
JasperReports Server Clients	154
Load Balancer	155
JasperReports Server Instances	156
Shared Repository Database	157
Job Schedulers	158
Other Shared Resources	159
Jaspersoft OLAP in a Cluster	160
Session Management and Failover	161
Impact on Browser Users	163
Impact on API Clients	165
Sample Configurations	166
Load Balancer Configuration	172
Cluster Design Process	175
Performance Requirements	176
Availability Requirements	177
Scalability Requirements	177
Sizing a Cluster	178
Load Balancer	179
Cluster Nodes	179
Software Configuration	180
Databases	180
Network	181

Policies and Procedures	182
Glossary	183
Jaspersoft Documentation and Support Services	198
Legal and Third-Party Notices	200

Introduction

JasperReports® Server builds on JasperReports® Library as a comprehensive family of Business Intelligence (BI) products, providing robust static and interactive reporting, report server, and data analysis capabilities. These capabilities are available as either stand-alone products, or as part of an integrated end-to-end BI suite utilizing common metadata and provide shared services, such as security, a repository, and scheduling. The server exposes comprehensive public interfaces enabling seamless integration with other applications and the capability to easily add custom functionality.



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 heart of the Jaspersoft® BI Suite is the server, which provides the ability to:

- Easily create new reports based on views designed in an intuitive, web-based, drag and drop Ad Hoc Editor.
- Efficiently and securely manage many reports.
- Interact with reports, including sorting, changing formatting, entering parameters, and drilling on data.
- Schedule reports for distribution through email and storage in the repository.
- Arrange reports and web content to create appealing, data-rich Jaspersoft Dashboards that quickly convey business trends.

For users interested in multi-dimensional modeling, we offer Jaspersoft® OLAP, which runs as part of the server.

While the Ad Hoc Editor lets users create simple reports, more complex reports can be created outside of the server. You can either use Jaspersoft® Studio or manually write JRXML code to create a report that can be run in the server. We recommend that you use Jaspersoft Studio unless you have a thorough understanding of the JasperReports file structure.

You can use the following sources of information to learn about JasperReports Server:

- Our core documentation describes how to install, administer, and use JasperReports Server and Jaspersoft Studio. Core documentation is available as PDFs in the doc subdirectory of your JasperReports Server installation. You can also access PDF and HTML versions of these guides online from the [Documentation section](#) of the Jaspersoft Community website.
- Our Ultimate Guides document advanced features and configuration. They also include best practice recommendations and numerous examples. You can access PDF and HTML versions of these guides online from the [Documentation section](#) of the Jaspersoft Community website.
- Our [Online Learning Portal](#) lets you learn at your own pace, and covers topics for developers, system administrators, business users, and data integration users. The Portal is available online from the Professional Services section of our [website](#).
- Our free samples, which are installed with JasperReports Library, Jaspersoft Studio, and JasperReports Server, are available and documented online. Please visit our [GitHub repository](#).
- If you have a subscription to our professional support offerings, please contact our Technical Support team when you have questions or run into difficulties. They are available on the web at <https://www.jaspersoft.com/support>.

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 views and reports, advanced charts, dashboards, Domains, auditing, and a multi-organization architecture for hosting large BI deployments.

This chapter contains the following sections:

- [Community and Commercial Editions](#)
- [User Descriptions and Document Maps](#)
- [Getting Started](#)

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.

User Descriptions and Document Maps

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 in this guide.

Technical Business Analyst

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

If you're a technical business analyst, refer to the following:

- [Changing the UI With Themes](#)
- [Customizing Menus](#)
- [Working With Custom Java Classes](#)

Report Developer

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

If you're a report developer, refer to the following:

- [Custom Data Sources](#)

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're a system developer, refer to the following:

- [JasperReports Server APIs](#)
- [Customizing the User Interface](#)

System Administrator and Database Administrator

System administrators install, deploy, maintain, and troubleshoot JasperReports Server, along with other systems in their environment. They also manage the server, including creation and maintenance of users, roles, permissions, and organizations, and authorization.

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're a system or database administrator, refer to the following:

- [Custom Data Sources](#)
- [Restricting Access by Role](#)
- [Working With Custom Java Classes](#)
- [Designing a Cluster](#)

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-9.0.0`

Linux: `<USER_HOME>/jasperreports-server-9.0.0`

Mac: /Applications/jasperreports-server-9.0.0

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

Commercial Editions: `http://<hostname>:<port>/jasperserver-pro/login.html`

Community Project: `http://<hostname>:<port>/jasperserver/login.html`

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:

User ID	Password	Description
superuser	superuser	System-wide administrator (commercial editions only)
jasperadmin	jasperadmin	Administrator for the default organization (commercial editions) System-wide administrator (community project)
joeuser	joeuser	Sample end-user
demo	demo	Sample end-user for the SuperMart Dashboard demonstration



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.

Customizing Reports

JasperReports® Server makes it easy to run reports. When you run a report, it opens in the interactive Report Viewer. With the Viewer, you can personalize and refine the displayed report data. If the report has input controls, you run the report with one set of data and then another.



This chapter 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@.

This chapter contains the following sections:


- [The Report Viewer and Conditional Text](#)
- [Creating a New Report Template](#)

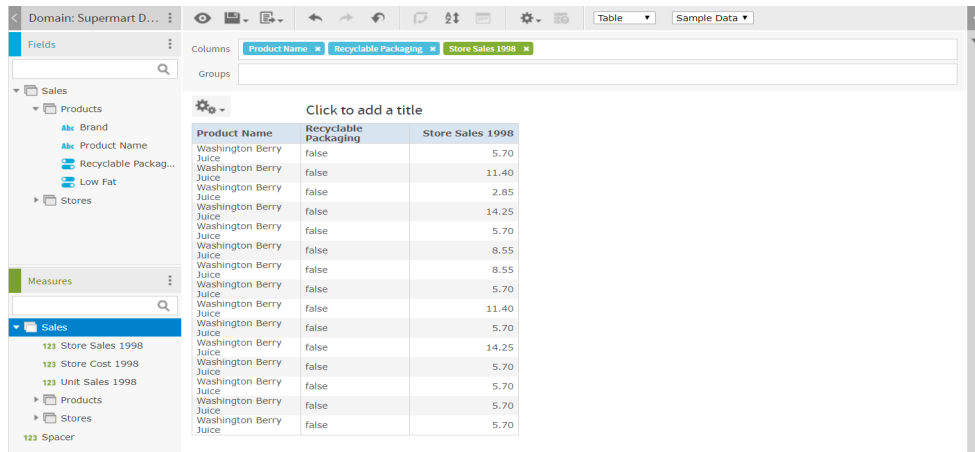
The Report Viewer and Conditional Text

The interactive Report Viewer lets you highlight table values using conditional formatting. This section shows how to use multiple conditions in a table to create a stoplight format based on ranges. To set up this format, you need to use the inheritance feature of conditional formatting. For colored backgrounds, this specifies that when a table cell satisfies multiple conditions, the condition that appears highest in the list of conditions is applied.

To create the Ad Hoc table for use in the example


1. Select Create > Ad Hoc View from the menu. The Data Chooser wizard opens.
2. Click Domains, select SuperMart Domain, and click Choose Data. The Data Chooser opens to the Select Fields page.
3. In the Source panel, double-click Sales to move it to the Selected Fields panel and click OK. The Ad Hoc Editor is displayed with the selected fields.
4. Click . The Select Visualization Type window appears.

5. Select  and click X in the upper corner of the window to create a table.
6. Double-click the following fields and measures to add them to the Columns area: Product Name, Recyclable Packaging, Store Sales. The Ad Hoc view appears as shown in the following figure.



Product Name	Recyclable Packaging	Store Sales 1998
Washington Berry Juice	false	5.70
Washington Berry Juice	false	11.40
Washington Berry Juice	false	2.85
Washington Berry Juice	false	14.25
Washington Berry Juice	false	5.70
Washington Berry Juice	false	8.55
Washington Berry Juice	false	8.55
Washington Berry Juice	false	5.70
Washington Berry Juice	false	11.40
Washington Berry Juice	false	5.70
Washington Berry Juice	false	14.25
Washington Berry Juice	false	5.70
Washington Berry Juice	false	5.70
Washington Berry Juice	false	5.70
Washington Berry Juice	false	5.70
Washington Berry Juice	false	5.70
Washington Berry Juice	false	5.70





Figure 1: Ad Hoc View for Conditional Text

7. Hover over  and select Save Ad Hoc View and Create Report. The Save Ad Hoc View dialog opens.
8. Fill in the required fields as follows:
 - a. Data View Name: Conditional Text Example View
 - b. Data View Description: Created in Ultimate Guide
 - c. Report Name: Conditional Text Example Report
 - d. Report Description: Created in Ultimate Guide
9. For Save Location, click Browse, select Public > Samples > Reports, and click OK.
10. Click Save. A message confirms that the view was saved.

To open the report in the viewer

1. Select View > Repository.
2. Navigate to Public > Samples > Reports and click Conditional Text Example Report. The report opens in the interactive report viewer.

To create “stop light” conditional formatting on a numeric column

1. Click the Store Sales column. The column is highlighted and the column formatting icons appear at the top of the column.
2. Move your mouse over  and select Formatting... The Format Columns dialog box appears.
3. Click the Conditional Formatting tab. The Conditional Formatting options appear.
4. Click Add to create a new condition, and fill in the fields as follows:
 - a. Select Greater than from the Operator menu.
 - b. Enter 8 in the Condition box.
 - c. Click  and pick a green background.
5. Click Add to create a second condition, and fill in the fields as follows:
 - a. Select Greater than from the Operator menu.
 - b. Enter 3.5 in the Condition box.
 - c. Click  and pick a yellow background.
6. Click Add to create a new condition, and fill in the fields as follows:
 - a. Select Less than or equal to from the Operator menu.
 - b. Enter 3.5 in the Condition box.
 - c. Click  and pick a red background.

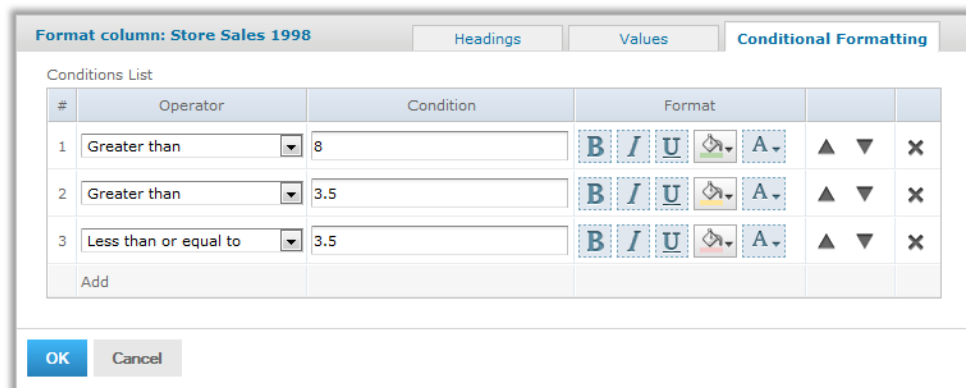


Figure 2: Conditional Formatting for Numeric Values

7. Click OK. The dialog box closes and your choices are applied. The report appears as shown in the following figure:

Product Name	Store Sales 1998
Landslide Sesame Oil	2.90
Booker Mild Cheddar Cheese	2.72
Tri-State Honey Dew	4.26
Blue Medal Large Brown Eggs	4.38
Denny Soft Napkins	9.96
Cormorant Toilet Bowl Cleaner	11.43
Club Strawberry Yogurt	2.22
Ship Shape Seasoned Hamburger	3.50
Imagine Frozen Cheese Pizza	8.10
High Top Walnuts	5.44

Figure 3: Report with Conditional Formatting

Notice that numbers greater than 8 satisfy the first two conditions; the first condition they satisfy is the one that is applied.

Creating a New Report Template

Report templates customize the look of the reports you create from Ad Hoc views. JasperReports Server has several report templates in the Public/Templates folder of the repository. If you want to create a new report template, Jaspersoft recommends editing the default report template in Jaspersoft Studio and importing it into JasperReports Server.

The following example shows how to create a template using the following:

- Export the default report template from JasperReports Server. This is usually the Actual Size report template, which is used in this example.
- Edit the report template in Jaspersoft Studio to add a page header with a company logo and the current date and a page footer with the page number. This example uses the logo.png sample image from the JasperReports Server's repository as part of the updated template.
- Import the edited report template as a new template in JasperReports Server and use it to create a report from an Ad Hoc view.



Do not use the original filename for the new template file. JasperReports Server will overwrite the original template if you upload a new file with the same filename. In addition, your changes may be overwritten if you upgrade the server. See the JasperReports Server Administrator Guide for details on how you can change the default template for reports.

To export a template in JasperReports Server

1. Log in as an admin to JasperReports Server.
2. Click View > Repository and browse to Public > Templates.
3. Right-click on the Actual Size report template and select Export. The Export Resources dialog appears.

Export Resources

Export Data File Name (required):

Exported file will contain an encryption key. Refer to the JasperReports Server Security Guide for more information

Import-Export Key:

Server Key
 For your JasperReports Server only.

Legacy Key
 For all JasperReports Servers.

Export Options:

Include report jobs

Include alerts

Include repository permissions

Include dependencies

Export Cancel

Figure 4: Export Dialog Box

4. If desired, change the default name of the zip file for the exported catalog. This dialog allows only the zip archive format.
5. Select the Legacy key to protect any passwords in the export catalog.
6. Click Export. The server generates the catalog Zip file and your browser prompts you to save the file.
7. Locate the Zip file in your download folder and extract the files.
8. In the extracted files folder, find resources/public/templates/actual_size.<ver>.jrxml.data.

9. Rename the file to remove the .data extension. The filename should now be actual_size.<ver>.jrxml.
10. Move the template file to your MyReports folder in your Jaspersoft Studio workspace.
11. In JasperReports Server browse to Public > Samples > Resources > Images.
12. Right-click the logo.png file and select Properties from the context menu.
13. Copy the path for the file and click Cancel.



Figure 5: Image Properties Box

To edit a report template in Jaspersoft Studio

1. Open Jaspersoft Studio.
2. Go to File > Open File.
3. Browse to your My Reports folder, select the template file, and click Open. Jaspersoft Studio opens the template in the Design tab.
4. In the Outline tab, right-click Page Header and select Add Band. The Page Header band appears in the template.
5. Right-click Page Footer and select Add Band. The Page Footer appears in the template.

6. From the Palette tab, drag the Image element to the Title band of the template. The Create new image element window appears.
7. Select Select a resource from Jaspersoft Server.
8. Enter the repository path for the logo.png image file in the field under Options and click OK.

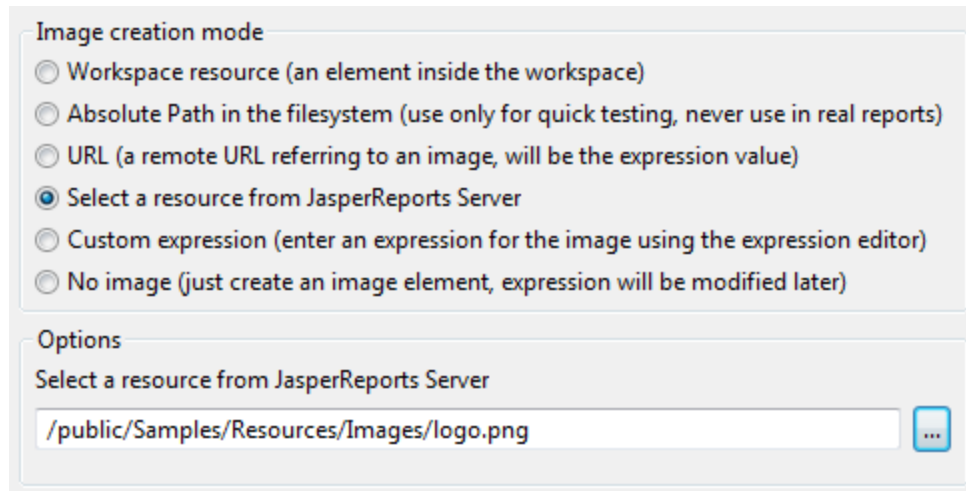


Figure 6: Create New Image Element Window

9. Click and drag the image's margins to adjust its size.
10. From the Palette tab, drag the Current Date element to the title bar of the template.
11. Drag the Page Number element to the footer of the template.

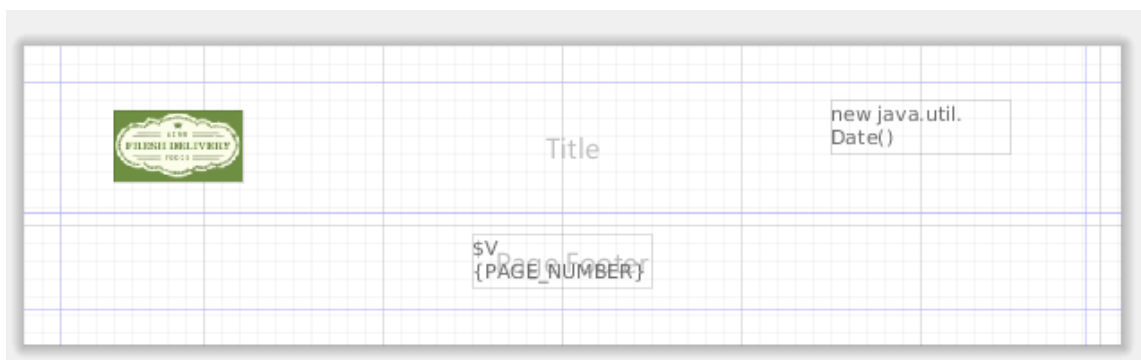


Figure 7: Customized Report Template

12. Save the template with a new filename.



Do not export the template through Jaspersoft Studio. It must be exported manually from JasperReports Server so that it can be uploaded as a template, not a report.

To import a template and create a report in JasperReports Server

1. In JasperReports Server, click View > Repository and browse to Public > Templates.
2. Right-click the Templates folder and select Add Resource > File > JRXML from the context menu. The Add File page appears.
3. Enter the required information for the template. Click Choose File to locate the report template on your file system.
4. When done, click Submit. The new template appears in the Templates folder in the repository.
5. On the JasperReports Server Home page, click Create in the Reports block. The Create Report wizard opens.
6. Select the Ad Hoc view you want to use as the basis for your report.
7. Select your new report template.
8. Click OK. If asked, enter the input controls needed.

JasperReports Server displays the report.

 March 13, 2017

Country	Measures	Non-Recyclable				Recyclable				Totals
		Ancient	New	Old	Totals	Ancient	New	Old	Totals	Totals
Mexico	Years Since Remodel	49.00	23.00	35.00	27.00	49.00	26.00	35.00	27.00	27.00
	Store Sales 2013	437.00	884.75	208.15	1,529.90	379.65	1,100.55	342.60	1,822.80	3,352.70
	\$/SQ FT	58.75	54.23	52.09	55.23	58.75	52.09	53.26	53.70	54.40
	Tax Rate	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Sales % Group	79.62	61.57	17.06	47.73	68.88	55.55	23.15	45.43	46.45
Sales % of Total	6.05	12.26	2.88	21.20	5.26	15.25	4.75	25.26	46.45	
USA	Years Since Remodel	49.00	30.00	40.00	35.00	49.00	26.00	40.00	35.00	35.00
	Store Sales 2013	111.85	552.23	1,011.65	1,675.73	171.50	880.49	1,137.16	2,189.15	3,864.88
	\$/SQ FT	46.96	35.47	78.63	62.25	48.00	48.85	76.20	63.05	62.70
	Tax Rate	6.50	6.50	6.50	6.50	6.50	6.50	6.50	6.50	6.50
	Sales % Group	20.38	38.43	82.94	52.27	31.12	44.45	76.85	54.57	53.55
Sales % of Total	1.55	7.65	14.02	23.22	2.38	12.20	15.76	30.33	53.55	
Totals	Years Since Remodel	49.00	26.00	40.00	35.00	49.00	26.00	37.50	35.00	35.00
	Store Sales 2013	548.85	1,436.98	1,219.80	3,205.63	551.15	1,981.04	1,479.76	4,011.95	7,217.58
	\$/SQ FT	56.31	46.97	74.07	58.91	55.41	50.66	70.90	58.80	58.85
	Tax Rate	1.32	2.50	5.39	3.40	2.02	2.89	5.00	3.55	3.48
	Sales % Group	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Sales % of Total	7.60	19.91	16.90	44.41	7.64	27.45	20.50	55.59	100.00	

Figure 8: Report with New Template

Custom Data Sources

JasperReports Server provides out-of-the-box support for many commonly used data sources, such as JDBC, JNDI, cloud-based databases, and several types of big data, as described in the *JasperReports Server Administrator Guide*. In some cases, however, you may want to connect to a data source that is not supported by default or to a custom data source that you have created. To do this, you need to extend JasperReports Server by adding certain executable libraries and editing files in your server's configuration.

This chapter explains how to add new data source types from several different origins:

- Pre-installed data sources – The libraries for these are installed by default, but you need to change the configuration to make them visible in the user interface.
- Custom data source examples – The source code for these is installed by default, but you must run the included scripts to compile the libraries, copy them to the correct location, and configure the server.
- Your own custom data sources – You must write Java classes and compile them into JAR files, or download them from the community website, then copy them to the correct location and configure the server.

All three use the same mechanisms and files to be configured in the server. After following the appropriate steps, your custom data source types show up in the New Data Source dialog, and users can create instances of your custom data source and use them to access data. For example, the JSON and XML file data source types added in JasperReports Server 6.1 are implemented using the same patterns as for custom data source types.



This chapter explains how to configure JasperReports Server to support an existing JasperReports Library data source (`JRDataSource`). For this, you must write additional code to implement further interfaces (`ReportDataSourceService` and optionally others).

To develop your own custom data source (`JRDataSource`), see the source code for the examples mentioned in this chapter. For further information, see the *JasperReports Library Ultimate Guide*.

This chapter contains the following sections:

- [Pre-installed Data Source Types](#)
- [Custom Data Source Examples](#)
- [Custom Data Source Architecture](#)

- [Creating a Custom Data Source](#)

Pre-installed Data Source Types

The following data source types are pre-installed with JasperReports Server, but not visible by default in the UI:

Name	Description
jdbcQueryDataSource mongoDBQueryDataSource xmlaQueryDataSource	<p>The query data sources for different types of databases let you specify a query and return a single relational table. Thus the instance of the data source in the server contains a query that limits what data can be further queried by the report or view based on it. The jdbcQueryDataSource type is a generic data source that lets you select any JDBC driver in the UI; the other data source types are instances of the jdbcQueryDataSource type where the JDBC driver is hard-coded for the specific database. In addition, the other data source types have database-specific fields that appear in the UI.</p> <p>The jdbcQueryDataSource data source type appears in the UI with "Before 6.4" appended to its name.</p>
jsonDataSource JsonSeriesDataSource remoteXmlDataSource	<p>The file data sources let you specify a system file as well as a query and return a single relational table. The jsonDataSource and remoteXmlDataSource data source types appear in the UI with "Before 6.4" appended to their name.</p> <p>For information about using the file-based data sources, see the <i>JasperReports Server Administrator Guide</i>.</p>
jdbcQueryDataSource2 jsonDataSource2 remoteXmlDataSource2 mongoDBQueryDataSource2	<p>These are data sources with full domain that support domain-related features like returning JDBC metadata, creating derived tables, supporting calculated fields, domain pre-filtering, joining other data sources in a virtual data source, and full support in the Domain Designer. The xlsDataSource and xlsxDataSource types are Excel file-based</p>

Name	Description
xlsDataSource	custom data sources and the textDataSource is a text/CSV custom data source. For information about using these data sources, see <i>JasperReports Server Administrator Guide</i> .
xlsxDataSource	
textDataSource	
cassandraQueryDataSource	These data source types are deprecated. Instead, use a JDBC data source type like Cassandra, Hive, or Impala.
HiveDataSource	

All of these data source types support the following:

- You can use SQL queries in reports to access the data as a relational table.
- You can create a Domain based on the data source, allowing you to alter the visibility and names of the fields extracted from the database or file. On the Display tab of the Domain Designer you can also specify which fields are measures.
- You can create Ad Hoc views using the Domain based on the data source, allowing you to explore and interact with data from the database or file.
- You can create virtual data sources that combine several data sources. You can then create a Domain based on the virtual data source to join the tables and access the joined data in Ad Hoc views and reports. You can even combine different formats, such as an XML file and MongoDB, as long as their data structures are compatible so the tables can be joined.

Enabling the Pre-installed Data Source Types

By default, the pre-installed data source types do not appear in the New Data Source dialog and must be enabled first.

To make pre-installed data source types available in the UI

1. Open the file `.../WEB-INF/applicationContext-remote-services.xml` for editing.
2. Locate the element `<util:set id="customDataSourcesToHide">`.
3. Comment out the data source types you want to display.
4. Restart the server.

In the following example, JSON, JSON (Before 6.4), and Remote XML data source types are commented out so they appear in the drop-down menu in the New Data Source dialog:

```
<util:set id="customDataSourcesToHide">
  <value>remoteXmlDataSource2</value> <!-- Full domain support remote XML custom data source
-->
  <value>remoteXmlDataSource</value> <!-- Simple single table remote XML custom data source
-->
  <value>mongoDBQueryDataSource2</value> <!-- Full domain support mongodb query custom data
source -->
  <value>mongoDBQueryDataSource</value> <!-- Simple single table remote XML custom data
source -->
  <value>jsonDataSource2</value> <!-- Full domain support JSON custom data source -->
  <value>jsonDataSource</value> <!-- Simple single table remote XML custom data source -->
  <value>jsonQLDataSource</value> <!-- Full domain support JSON QL custom data
source -->
  <value>jdbcQueryDataSource2</value> <!-- Full domain support jdbc query custom data source
-->
  <value>jdbcQueryDataSource</value> <!-- Simple single table remote XML custom data source
-->
  <!--<value>xlsDataSource</value> Full domain support XLS custom data source -->
  <!--<value>xlsxDataSource</value> Full domain support XLSX custom data source -->
  <value>textDataSource</value> <!-- Full domain support TEXT/ CSV custom data source -->
  <value>JsonSeriesDataSource</value> <!-- Simple single table remote XML custom data source
-->
  <value>xmlaQueryDataSource</value> <!-- Simple single table XMLA Query custom data source
-->
  <value>cassandraQueryDataSource</value> <!-- Deprecated. Please use cassandra SIMBA JDBC
driver to run native CQL. Append "QueryMode=1" in URL in order to run CQL -->
  <value>HiveDataSource</value> <!-- Deprecated. Please use Hive/ Impala JDBC drivers
instead -->
</util:set>
```

After a data source type has been enabled in the UI, you can create an instance of that type.

To create a data source using a query example

1. Log on as an administrator.
2. Click View > Repository, expand the folder tree, and right-click a folder to select Add Resource > Data Source from the context menu. Alternatively, you can select Create > Data Source from the main menu on any page and specify a folder location later. If you installed the sample data, the suggested folder is Data Sources. The New Data Source page appears.
3. In the Type field, select a data source that you enabled, for example JSON Data Source (Before 6.4) or Remote XML Data Source. The fields on the page change to prompt for the connection information required for your data source.

You have the option to use attributes in the values of data source parameters. See the *JasperReports Server Administrator Guide* for more information.

4. Fill in the required connection information and enter the query you want to use.

5. Enter a name for the data source and an optional description. The Resource ID is generated from the name you enter. If you haven't already specified a location, expand the folder tree and select the location for your data source.
6. Click Save in the dialog. The data source appears in the repository.

Understanding the Pre-installed Data Sources

The Java source for the pre-installed samples can be found online:

1. Navigate to the pre-installed samples source code:

<https://github.com/TIBCOSoftware/jasperreports-server-ce/tree/master/jasperserver/jasperserver-custom-datasources/src/main/java/example/cdspro>

The JDBC query and flat file data source types each leverage an existing data adapter class in JasperReports Library. The data adapter class used depends on the data source type. For example, the JDBC query data source type uses a `JDBCQueryDataSourceDefinition` class, based on `JdbcDataAdapterImpl`, to allow the user to enter database connection information and a JDBC query in the New Data Source dialog. When a user creates or views a report or Ad Hoc view based on an implementation of this data source type, JasperReports Server creates a JasperReports data source as follows:

- Builds a custom data source using the JasperReports Library JDBC Data Adapter.
- `JDBCQueryDataSourceService` creates a JDBC connection based on the driver, URL, username, and password entered by the user.
- `JRJdbcQueryExecuterFactory` executes the user-defined query and retrieves the metadata layer necessary for Domain support.
- `JDBCQueryDataSourceService` creates the JasperReports data source that is used by JasperReports Library to fill the report.

In addition, the pre-installed data sources implement Domain support using the `CustomDomainMetaData` class.

You can find the message catalog and Spring bean definition file for each pre-installed data source type in the locations described in [Files Used by a Custom Data Source Implementation](#). The file names are based on the data source name, for example:

```
.../WEB-INF/bundles/cassandraqueryds.properties
```


Custom Data Source Examples

The WAR file installer includes several sample data source types in the <js-install>/samples directory. You can build and deploy the examples after you have deployed JasperReports Server. These samples include both the JasperReports Library data source implementation (JRDataSource) that contains the logic to access data and the interface to fill reports, and the implementation of ReportDataSourceService that allows the server to instantiate the JRDataSource at runtime.

These samples are not intended to be used in production environments. Modify and test these samples to understand how to create your own custom data source. The examples include:

- Custom bean data source
- Webscraper data source
- Hibernate data source
- Custom data source pro with metadata support for Domains

Prerequisites

The samples directories include an Ant script that compiles the Java code for the samples and places all the required files in the correct locations in JasperReports Server. To run the Ant script, you need the correct versions of the Java Development Kit and Apache Ant.

Java Development Kit

The examples are provided as Java source files. To work with the examples, you must install the Java Development Kit (JDK). Ensure that the JAVA_HOME environment variable points to a full JDK installation. The samples are supported only with JDK 1.8.

Apache Ant

Ant is installed as part of the JasperReports Server installation process. Run Ant using the following command:

Linux: <js-install>/apache-ant/bin/ant <ant-arguments>

Windows: <js-install>\apache-ant\bin\ant.bat <ant-arguments>

If you cannot find Ant on your system, you can download it from <http://ant.apache.org>. The bundled Apache Ant is version 1.10.13; this version or higher is recommended.

Installing the Custom Data Source Examples

The `<js-install>/samples/customDataSource` and `<js-install>/samples/customDataSource-pro` directories include:

- `readme.txt` – Text file describing how to build the examples.
- `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 source types.

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 your JasperReports Server web application.
3. Run the Ant command (as described in [Prerequisites](#)) with no arguments; this executes the default target, which is named `deploy`. The `deploy` target initiates these actions:
 - Compiles the Java source files under the `src` directory.
 - Deploys the compiled Java class files to the web application.
 - Deploys files under the `webapp` directory to the web application.



See [Files Used by a Custom Data Source Implementation](#) for the final locations of the files in JasperReports Server

4. Repeat this procedure for the sample in `<js-install>/samples/customDataSource-pro`.
5. For the webscraper report example, you must register its query executer factory as described in [Webscraper Custom Data Source](#).
6. Restart the application server.

The example custom data source types are now available from the New Data Source page in JasperReports Server.

To test the samples and create or view reports

1. Log on to JasperReports Server as an administrator.
2. Open the New Data Source page (for example, by selecting Create > Data Source for the main menu), enter the required values for the data source type you selected, and save the data source. See the *JasperReports Server Administrator Guide* for more information about creating data sources.
3. To test a sample, upload the associated report from the `<js-install>/samples/customDataSource/reports` directory and view the report.

Each of the examples is described in the following sections.

Custom Bean Data Source

This custom data source shows how to create a data source type for a JasperReports Library data source (JRDataSource) implemented as a simple bean. It includes the following:

- A JRDataSource implemented as an array of data declared in the source code.
- The ReportDataSourceService code required by JasperReports Server to set up and tear down the JRDataSource.
- The Spring bean definition file for the custom data source type:
`<js-install>/samples/customDataSource/webapp/WEB-INF/applicationContext-sampleCDS.xml`
- An example report that uses this data source type
`<js-install>/samples/customDataSource/reports/simpleCDS.jrxml`

Webscraper Custom Data Source

The webscraper custom data source fetches a web page, decodes its HTML, and extracts selected data that is turned into field values in the data source. The Spring bean definition file for this custom data source type is located in:

```
<js-install>/samples/customDataSource/webapp/WEB-INF/applicationContext-  
webscraperDS.xml
```

The webscraper data source type definition 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 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. The `<js-install>/samples/customDataSource/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. Set the URL and DOM Path according the website you are trying to access.

When a data source of this type is used in a report, dashboard, or Ad Hoc view, JasperReports Server creates a `JRDataSource` by:

- Using the URL to issue a GET request for an HTML page.
- Converting the HTML response into XML using JTidy (<http://jtidy.sourceforge.net>).
- 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.



Web sites are frequently redesigned and the URLs used by any website are subject to change. This can affect your reports.

In order to use the webscraper data source, you must first register the webscraper query executer factory. One way to do this is as follows:

1. Install the samples using ant as described in [Installing the Custom Data Source Examples](#).
2. Open the file `.../WEB-INF/classes/jasperreports.properties` for editing.
3. Add the following at the end of the file:

```
# registering query executor for webscraperQETest.jrxml example
net.sf.jasperreports.query.executor.factory.webscraper=example.cds.WebScraperQueryExecutorFactory
```

4. Save the file.
5. Restart your application server.

For more information about registering query executors, see the Report Query section in the *JasperReports Library Ultimate Guide*.

Hibernate Custom Data Source

The Hibernate custom data source type implementation supports Hibernate Query Language (HQL) queries in report units in JasperReports Server. The Spring bean definition file and related Java code for the Hibernate custom data source example are in the following locations in the `<js-install>/samples/customDataSource/` directory:

- `webapp/WEB-INF/applicationContext-hibernateDS.xml` – Spring bean XML to configure the Hibernate data source
- `src/example/cds/HibernateDataSourceService.java` – implementation of `ReportDataSourceService` for the data source
- `src/example/cds/HibernateSessionFactoryFinder.java` – a helper class used to locate a `SessionFactory` instance for `HibernateDataSourceService`

The implementation works with the JasperReports Library's support for HQL queries to create a data source. The library will use the `JRHibernateQueryExecutor` to run an HQL query in a report unit, and this query executor needs a `Hibernate Session` object. The Hibernate custom data source will create the session from a `Hibernate SessionFactory` configured as a Spring bean in JasperReports Server.

The Hibernate data source configuration on the data source menu in JasperReports Server includes the following element:

- **Session Factory Bean Name:** A `SessionFactory` created with a named Spring bean of class `SessionFactory`, `HibernateDaoSupport`, or one of their subclasses.

The sample Hibernate test report contains an HQL query on the JasperReports Server repository, which uses Hibernate internally.

To run the Hibernate data source example

1. Install the samples using ant as described in [Installing the Custom Data Source Examples](#).
2. Create a new Hibernate data source:
 - a. Create a data source and select Hibernate for the type.
 - b. Enter `sessionFactory` for the name of the `SessionFactory` bean – this is the id of the repository's Hibernate `SessionFactory`.
 - c. Save the data source and give it a name when prompted.
3. Upload and configure the sample report:
 - a. Navigate to the repository location where you want to save the report.
 - b. Right-click and select Add Resource > JasperReport. The Add JasperReport page is displayed.
 - c. On the Set Up page, enter a name for the report unit.
 - d. Select Upload a Local File, click Browse, select `<js-install>/samples/customDataSource/reports/hqlTest.jrxml`, and click OK.
 - e. On the Data Source page, click Select data source from repository and browse to the data source created in the previous step.
 - f. Click Submit to save the report.
4. Run the new report, and it should show a list of resources in the repository.

Custom Data Source Pro

The Custom Data Source Pro example works in commercial editions of JasperReports Server. It uses an additional class to expose metadata, so that the data source can be used in Domains and Ad Hoc views.

This example is located in the `<js-install>/samples/customDataSource-pro` directory. It includes the following:

- A `JRDataSource` implemented as an array of data declared in the source code.
- The example implements Domain support using the `CustomDomainMetaData` class.
- Its Spring bean definition file is located in:
`<js-install>/samples/customDataSource-pro/webapp/WEB-INF/applicationContext-sampleCDSWithMetData.xml`

Custom Data Source Architecture

A JasperReports Library data source (`JRDataSource`) is a Java class that contains the logic to retrieve data and feed it to the JasperReports Library process that generates a JRXML report. For example, you may write code to access a REST API and format the data. When using JasperReports Library natively in a Java environment, you must write the code to initialize your `JRDataSource`, for example any URLs or credentials needed to access the data.

JasperReports Server lets users enter the parameters for a data source and select which reports to run. When you want to use this same `JRDataSource` in the JasperReports Server environment, you must write additional classes and resources so that the server can do the following:

- At startup, register the data source so it can be selected by users.
- When a user creates an instance of this data source, get its parameters from the user and store them in the repository.
- When a user runs a report that uses this data source, instantiate and initialize the `JRDataSource` at runtime from the stored parameters. When the report is finished, clean up the `JRDataSource` object.
- Optionally, if you have input controls for reports that use this data source, instantiate and initialize the `JRDataSource` again to process the query for input controls.

Each of these actions are described in the following sections.

Server Startup

On startup, the Spring framework registers the custom data source definition with the custom report data source service factory in JasperReports Server. Users see the custom data source type on the Type menu in the New Data Source dialog, using the name defined in the message catalog.

Creation of a Custom Data Source

When a user selects a custom data source type from the Type menu in the New Data Source dialog, they must enter values for any visible properties required to create the

corresponding `JRDataSource`. Properties are specified in the Spring bean definition file, with labels defined in a message catalog (or bundle if you include translated messages for other locales). There may be additional hidden properties, set up in the Spring definition file.

When you create a custom data source, you can optionally implement validation for the property values in the New Data Source dialog. The logic for validation is defined by a Java class that implements the `CustomDataSourceValidator` interface. This class receives the user input, verifies it, and if necessary raises an error with strings defined in the message catalog.

Saving the data source instance creates a persistent object in the JasperReports Server repository.

Instantiation of a Custom Data Source

When a user runs a report that uses the custom data source, the server instantiates the corresponding `JRDataSource` to access the data. This also happens when defining a Domain that uses the custom data source, when creating or running an Ad Hoc view based on a Domain that uses the custom data source, or when viewing a dashboard that includes any of these.

The `JRDataSource` is instantiated as follows:

- The custom data source instance is read from the repository and its data source definition is looked up by name.
- An implementation of `ReportDataSourceService` is created and the properties from the repository are used to set the corresponding properties on the data source service.
- A `JRDataSource` is created in one of two ways, depending on whether the data source uses a query:
 - If there is no query, the data source service has to provide the `JRDataSource`.
 - If there is a query, JasperReports Library creates a `JRQueryExecutor` of the correct class for the query language. The data source service passes any objects that the query executor needs by setting parameters. The `JRQueryExecutor` reads the parameters and produces the `JRDataSource`.
- The `JRDataSource` is used to fill the report and produce a `JasperPrint` object, from which the server generates HTML or other supported output formats.



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. Each field declared in the JRXML corresponds to a column in the `JRDataSource` output.

The following figure shows the main steps in creating a `JRDataSource` from a custom data source instance.

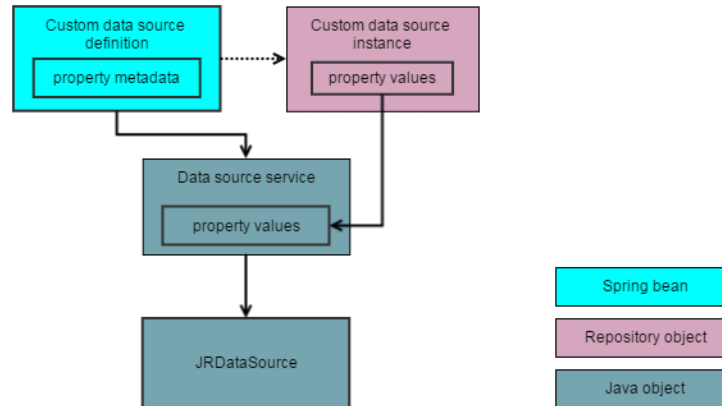


Figure 9: `JRDataSource` Creation from Custom Data Source Instance

Query Executors

A query executor is an implementation of the `JRQueryExecutor` interface in the 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 executor to use by looking at the `language` attribute of the `queryString` and looking up a query executor 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 a `JRDataSource` directly, without a `queryString` in the JRXML; or
- The server can pass implementation-specific objects to the query executor through the report parameter map. The query executor 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 executor is the JDBC data source: it passes a JDBC connection to the JDBC query executor, which it uses to pass the SQL queryString to the database.

The samples demonstrate both methods:

- The custom bean data source creates a JRDataSource directly, which returns data from a hard-coded internal array. See [Custom Bean Data Source](#).
- 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 demonstrate each of these approaches. See [Hibernate Custom Data Source](#).

Creating a Custom Data Source

If you have an existing JRDataSource implementation used with JasperReports Library that you'd like to use in JasperReports Server, you need to implement the supporting classes and configure the server. You need to create or edit the following files:

Files Used by a Custom Data Source Implementation

Type	Path (relative to web app directory)	Description
Java classes	WEB-INF/lib or WEB-INF/classes	The logic of your JasperReports data source, the supporting interfaces, and any optional classes compiled into JAR files.
Spring bean definition	WEB-INF/applicationContext- <name>.xml	The configuration of the data source within the server's web application.
Message catalog	WEB-INF/bundles/<name>.properties also <name>_<locale>.properties	Strings such as the name of the data source type in the New Data Source dialog, names for

Type	Path (relative to web app directory)	Description
		visible properties, and any validation messages. You can translate the strings for other locales in multiple files.
Query language configuration (optional)	WEB-INF/applicationContext-[pro-]remote-services.xml or WEB-INF/applicationContext-custom.xml and WEB-INF/js.spring.properties	If your JR data source is created using a query executer and you want the query language to be available in the UI, such as input control dialogs

These files work together to make your custom data source type available in the user interface and to instantiate your custom `JRDataSource`, as described in the following sections.

Writing Java Classes

As seen in [Custom Data Source Architecture](#), you must provide additional classes so that the server can instantiate your custom `JRDataSource` when needed with the requested parameters. This is the purpose of the `ReportDataSourceService` interface that you must implement. For code samples, see [Custom Data Source Examples](#).

Implementing the `ReportDataSourceService` Interface

A custom data source definition requires an implementation of the `ReportDataSourceService` interface, which sets up and tears down data source connections in JasperReports Server. It defines the following methods:

The ReportDataSourceService Interface

Interface	Method to Implement	Description
ReportDataSourceService	void setReportParameterValues (Map parameterValues)	Called before running a report; it creates resources needed by JasperReports Library to instantiate a JRDataSource, and adds them to the parameter map.
	void closeConnection()	Cleans up any resources allocated in setReportParameterValues().
	Property setters and getters	A getter and setter method corresponding to each parameter name, including hidden properties; for example, if you defined a property with the name user, you need getUser() and setUser() methods.

Defining Custom Data Source Properties

A custom data source definition can have properties so that users may 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. Ultimately, it is your JRDataSource that determines what properties are needed.

There are two kinds of properties:

- Editable properties that must be string values. When a user launches the New Data Source dialog to create an instance of your custom data source definition, the editable properties have text fields for user input. These values are persisted in the repository when you save the data source.
- Hidden properties that can be of any type. You set these property values in the Spring configuration file to be passed to your ReportDataSourceService implementation. Therefore, they do not appear in the New Data Source dialog, nor

do they need to be persisted in the repository. Use hidden properties if you want to give your `ReportDataSourceService` implementation access to a Spring bean instance.

These properties are defined in two places that must work together:

- Your `ReportDataSourceService` implementation must have getters and setters for each property, and your code can use the values for any type of processing. For source code examples, see [Hibernate Custom Data Source](#).
- The Spring beans need the list of properties to set up the New Data Source dialog, save the user values in the repository, and later instantiate your `ReportDataSourceService` when needed to fill a report. For examples of both editable and hidden properties, see the XML example in [Defining the Custom Data Source in Spring](#).

Implementing the Optional Validator Interface

A validator verifies property values entered by the user and rejects bad values with an optional message. The validators are applied to the editable properties that are returned from the New Data Source dialog, before a data source is being stored in the repository.

You can implement any level of validation that you need, such as:

- Null check (presence or absence of value)
- Type validation (string or number)
- Syntax validation (format or contents of a string)
- Range validation (value of a number)

Property values may include references to attributes, whose values are not determined until the data source is instantiated when running a report. An attribute has the following syntax: `{attribute('attrName')}` or `{attribute('attrName','[User|Tenant|Server]')}`. Your validator code should recognize these patterns in property value strings and allow or skip validation of the attribute reference.

To create a validator, implement the `CustomDataSourceValidator` interface as follows:

Optional Validator Interface

Interface	Method to Implement	Description
CustomDataSourceValidator	validatePropertyValues (CustomReportDataSource ds, Errors errors)	Your code checks parameters and calls errors.rejectValue() with the appropriate property name and error code (see Defining the Message Catalog).

For a source code example of the CustomDataSourceValidator implementation, see [Webscraper Custom Data Source](#).

Implementing Optional Query Executer 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.

Optional Query Executer Interfaces

Interface	Method to Implement	Description
JRQueryExecuterFactory	JRQueryExecuter createQueryExecuter (JRDataset dataset, Map parameters)	Returns a JRQueryExecuter for the given dataset and parameter map.
JRQueryExecuter	JRDataSource createDatasource() close()	Returns the actual data source based on the parameter map passed to the JRQueryExecuterFactory. Called when the report filling process is done with the data source.

Optional Query Executer Interfaces

Interface	Method to Implement	Description
	cancelQuery()	Called to clean up resources if the report filling process is interrupted.

Query Executors must be registered with the JasperReports Library before use. For instructions and source code examples, see [Webscraper Custom Data Source](#).

Implementing Optional Domain Support

To use your custom data source in Domains and Ad Hoc views, you must implement the CustomDomainMetaData interface.

Optional Domain MetaData Interfaces

Interface	Method to Implement	Description
CustomDomainMetaData	getFieldMapping()	Mapping between names in the data source and names to show in the Domain, as key value pairs.
	getJRFieldList()	Returns the list of JRFieldName (name, type, description), equivalent to columns, for your custom data source.
	getQueryLanguage()	Returns the query language specified in the query executor you are using.
	getQueryText()	Returns the query text used by the custom data source.

For source code examples, see the links to Java files in [Custom Data Source Pro](#) and [Pre-installed Data Source Types](#).

Defining the Custom Data Source in Spring

To configure your data source, you must add a Spring bean that references the `customDataSourceFactory` so that it will be visible to the server at run time. To do this, create a new file in the web application:

```
.../WEB-INF/applicationContext-<name>.xml
```

Within this file, there are two ways to configure your data source:

- If you implemented the `ReportDataSourceService` interface, use the `CustomDataSourceDefinition` class.
- If your data source extends `DataAdapterDefinition`, you can configure your data source as a data adapter.

Using CustomDataSourceDefinition

This class has the following properties:

Properties of CustomDataSourceDefinition Class		
Name	Required	Value
factory	Yes	A fixed value of <code>ref="customDataSourceFactory"</code> This bean manages all the custom data sources.
name	Yes	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.
serviceClassName	Yes	The class name for your <code>ReportDataSourceService</code> implementation.

Properties of CustomDataSourceDefinition Class

Name	Required	Value
validator	—	An instance of your CustomDataSourceValidator implementation.
property Definitions	—	Information describing each property used by the data source implementation, structured as a list of maps. See the table below.

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

Entry Keys for propertyDefinitions Property

Name	Required	Value
name	Yes	Name of property that matches a Java Bean property in the ReportDataSourceService implementation; also used in message catalog keys.
default	—	A default value for the property.
hidden	—	If a property has the hidden entry key set to true, then its value is fixed to that of the default entry key. Such properties are not visible in the New Data Source dialog, nor are they persisted. This is handy for making Spring beans accessible to ReportDataSourceService implementations.

The following XML defines a CustomDataSourceDefinition bean for the custom bean data source example:

```
<bean id="myCustomDataSource" class="com.jaspersoft.jasperserver.api.engine.jasperreports.util.
CustomDataSourceDefinition">
  <property name="factory" ref="customDataSourceServiceFactory"/>
  <property name="name" value="myCustomDataSource"/>
  <property name="serviceName" 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>

```

Using a Data Adapter

You can create a custom data source based on a data adapter in JasperReports Library. In this case, your query executer gets the custom data source properties from the data adapter.

To do this, you must first create an instance of your class in your custom data source definition Java code, implementing any additional properties you want:

```

public class MyCustomDataSourceDefinition extends DataAdapterDefinition {
    ...
}

```

Then you must create a bean for the data source in your XML file, using the class you defined in your Java file. In addition, you must specify the correct data adapter implementation in your `dataAdapterClassName` property. For example, the following XML defines a bean for the Mongo DB Query example:

```

<bean id="mongoDBQueryDataSource" class="example.cdspro.MongoDbDataSourceDefinition">
  <property name="factory" ref="customDataSourceServiceFactory"/>
  <property name="name" value="mongoDBQueryDataSource"/>
  <property name="dataAdapterClassName"
value="com.jaspersoft.mongodb.adapter.MongoDbDataAdapterImpl"/>
</bean>

```

Defining the Message Catalog

The message catalog contains labels and messages displayed in the New Data Source dialog when creating and editing custom data source instances. The various types of messages are shown in the following table, along with message naming conventions:

Messages about Instances of Custom Data Sources	
Message Type	Naming Convention
Name of the custom data source type	<cdsname>.name where <cdsname> is the value of the name property of the custom data source in its Spring definition.
Name of the custom data source property	<cdsname>.properties.<propname> where <propname> is the name of the property that the user must define when creating a custom data source.
Validation messages	<p><cdsname>.<propname>.<error> where <propname> is the name of the property that is invalid, and <error> is the type of error.</p> <p>The CustomDataSourceValidator implementation will call errors.rejectValue() for errors detected in property values for the custom data source. The second argument to errors.rejectValue() must match one of the messages defined in this catalog.</p>
Query language	query.language.<qlname>.label where <qlname> is the name of the query language. Define this property if you implement the optional query language and want to give your query language a different name, or want to have localized names for it in a bundle.

For example, the webscraper message catalog contains the following:

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

If you use your JasperReports Server in multiple languages, you can provide multiple copies of the message catalog, with translated strings (known as a message bundle). Each file name contains the locale to which it applies, according to the Java convention, for example `cdstest_fr.properties`. By convention, all message catalogs or bundles are stored in:

```
.../WEB-INF/bundles
```

To configure your message catalog, add a bean definition such as the following to the Spring definition file that you created in [Defining the Custom Data Source in Spring](#):

```
<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, specify the path and name of your message catalog file, omitting the `.properties` extension.

Adding the Custom Query Language to the UI

If you implemented the optional Query Executor, you must add it to the Spring configuration to make the query language for your custom data source definition appear in the UI. Query languages can be selected from a drop down list when defining query resources such as query-based input controls. The server matches the name of the query language to your data source to use its custom query executor class.

If you are using a commercial edition of JasperReports Server, edit the file `.../WEB-INF/applicationContext-pro-remote-services.xml` and locate the `queryLanguagesPro` bean. Add the name of your query language to the list, for example:

```
<bean id="queryLanguagesPro" parent="queryLanguagesCe"
  class="org.springframework.beans.factory.config.ListFactoryBean">
  <property name="sourceList">
    <list merge="true">
      <value>sl</value>
      <value>MyQueryLanguage</value>
    </list>
  </property>
</bean>
```

If you are using the community edition of JasperReports Server, edit the file `.../WEB-INF/applicationContext-remote-services.xml` and locate the `queryLanguagesCe` list. Add the name of your query language to the list, for example:

```
<util:list id="queryLanguagesCe">
  <value>sql</value>
  <value>hql</value>
  <value>domain</value>
  <value>HiveQL</value>
  <value>MongoDbQuery</value>
  <value>cql</value>
  <value>MyQueryLanguage</value>
</util:list>
```

The name must be a language supported by your query executor.

Alternatively, you can add your query executor as a separate bean without modifying the existing Spring configuration. In that case you would:

- Create a bean similar to `queryLanguagesPro` above, but that extends `queryLanguagesPro` if you are using a commercial edition. Save this bean with `id=customQueryLanguage` in a file named `.../WEB-INF/applicationContext-customQueryLang.xml`. Of course, you can use your own names for the id and file name.
- Edit the file `.../WEB-INF/js.spring.properties` and modify the following line:

```
bean.queryLanguages=queryLanguagesPro
```

so that it references the id of your new bean, for example:

```
bean.queryLanguages=customQueryLanguage
```

After editing and saving the files, restart JasperReports Server.

Installing a Custom Data Source Type

To install your custom data source type in JasperReports Server, add all the files it requires to the server web application directory. For the correct locations, refer to [Files Used by a Custom Data Source Implementation](#).

After adding the files and making the configuration changes specified in the previous sections, restart JasperReports Server.

Using a Custom Data Source Type

When you create a new data source type in JasperReports Server, it appears in the list of available types in the New Data Source dialog. If the custom data source is not listed as an available data source type, the custom data source is not properly installed.

When the new type is selected, JasperReports Server displays the fields and labels for the visible properties you configured. Users may enter values directly, or they may use attribute syntax to specify user, organization, or system attribute values. The server resolves any attribute values before passing the values to your `ReportDataSourceService` at run time.

When the form is submitted, the property values are validated with your optional `CustomDataSourceValidator` implementation and appropriate validation messages are displayed. Once the data source is validated, you can save it to the repository. The data source can now be used in a JRXML report. When you run a JRXML report, Domain, Ad Hoc view, or dashboard that uses this data source, the saved property values are used by your `ReportDataSourceService` implementation to instantiate your `JRDataSource` which then provides the data to run the report.

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.

We provide two types of APIs. The client APIs are used to create other applications that call functionality on the server. The server APIs are used to customize and extend the server itself.

The client APIs enable you to embed BI functionality into your own application. With these APIs you can access the server, browse its repository, set input controls, and run reports. The client APIs depend on a configured and running instance of the JasperReports Server. The client APIs have different uses depending on what functionality that you need and which protocol you want to use. They are described in the following sections:

- [REST API](#)
- [Visualize.js API](#)
- [Repository HTTP API](#)

The server APIs are the public interfaces and classes that can be used to extend or customize the behavior of the server itself. Using the server APIs, you can write classes that change the behavior and/or the user interface of the server. Depending on which behavior you want to extend or customize, you can either integrate your code through the Spring framework and redeploy the server or work with the source code distribution and recompile the whole server. The server APIs are described in the following sections:

- [The Public JasperReports Server API](#)

REST API

Through the REST API, your application can query a running instance of JasperReports Server to display repository contents and run reports embedded in your application's own UI. This section gives a brief summary of the API; for complete documentation, see the JasperReports Server REST API Reference.

The REST (Representational State Transfer) API depends on the standard methods provided by HTTP: GET, PUT, POST, and DELETE. Using the API, you can create client applications on any platform to access the server, interact with its resources, run reports and, with the right permissions, administer the server. Client applications send requests to named URLs that are called services.

The REST services in JasperReports Server 9.0.0 include:

- Repository services—Search the repository; create, modify, and delete resources; view and set permissions on repository objects; and import and export repository catalogs.
- Report services—Run reports and access report output; access and manipulate report options and input controls; and work with scheduled jobs.
- Administration services—Work with users and user attributes, roles and role membership, and organizations in commercial editions.

Jaspersoft also publishes a PHP client for embedding reports in PHP applications. The PHP client provides a wrapper for the REST API, so developers can use simple PHP structures and syntax to access JasperReports Server. For more information, see the [PHP client community project](#).

Visualize.js API

The Visualize.js API is a JavaScript library that allows web pages and web applications to embed JasperReports Server. The Visualize.js API was introduced in release 5.6, and Jaspersoft continues to add new features to it. This section gives a brief summary of Visualize.js; for complete documentation, see the *JasperReports Server Visualize.js Guide*.

Being a native JavaScript API, Visualize.js lets you design your own dynamic HTML interface for the reports you want to display and embed the report anywhere in your page or application. This is best demonstrated with a very simple example that consists of a web page containing a list of reports for the user to choose, and a container where you want the report to appear:

```
<!--Provide the URL to visualize.js-->
<script src="http://bi.example.com:8080/jasperserver-pro/client/visualize.js"></script>
<select id="selected_resource" disabled="true" name="report">
  <option value="/public/Samples/Reports/1_Geographic_Results_by_Segment_Report">Geographic
  Results by Segment</option>
  <option value="/public/Samples/Reports/2_Sales_Mix_by_Demographic_Report">Sales Mix by
```



```

Demographic</option>
  <option value="/public/Samples/Reports/3_Store_Segment_Performance_Report">Store Segment
Performance</option>
  <option value="/public/Samples/Reports/04._Product_Results_by_Store_Type_Report">Product
Results by Store Type</option>
</select>
<!--Provide a container to render your visualization-->
<div id="container"></div>

```

Note that the first line of the HTML above loads the Visualize.js library. You can then write the following JavaScript code to display the report the user selects from the list.

```

visualize({
  auth: { ...
  }
}, function (v) {

  //render report from provided resource
  v("#container").report({
    resource: $("#selected_resource").val(),
    error: handleError
  });

  $("#selected_resource").change(function () {
    //clean container
    $("#container").html("");
    //render report from another resource
    v("#container").report({
      resource: $("#selected_resource").val(),
      error:handleError
    });
  });

  //enable report chooser
  $(':disabled').prop('disabled', false);

  //show error
  function handleError(err){
    alert(err.message);
  }

});

```

The Visualize.js API renders interactive reports just like JasperReports Server, but within your own page. Users can sort columns, filter values, and even set conditional colors on Visualize.js reports. The Visualize.js API also lets you interact with and modify the DOM (Document Object Model) of the report if you want to make these same changes programmatically.

If your reports accept parameter values, your JavaScript can update the params object of the report properties and invoke the run function again.

```
// elsewhere in your JavaScript
```

```
// update report with new parameters
report
  .params({ "Country": ["USA"] })
  .run();
```

The API also provides functions to list the reports in the repository and get the list of report parameters. The example above is trivial, but the power of Visualize.js comes from this simple code. You can create dynamic user interfaces that allow your users to select the reports they want to see and to set the parameters they want.

Furthermore, with JavaScript you can perform database lookups or your own calculations to provide values. Your values could be based on 3rd party API calls triggered from other parts of the page or other pages in your app. When your reports can respond to dynamic events, they become truly embedded and much more relevant to the user.


For further examples and reference documentation, see the *JasperReports Server Visualize.js Guide*.

Repository HTTP API

The HTTP interface provides an easy way to implement the API for accessing repository objects. However, the HTTP interface can't be embedded the way the web services and Visualize.js APIs can be 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).

As shown in the following examples, the major entry points are:

- flow.html
- dashboard/**
- olap/**
- fileview/**

-  The examples in this section are generalized to describe both the community and commercial editions of JasperReports Server. For simplicity, this section refers to the deployment context generically. For example: `http://<host>:<port>/<context>/flow.html?_flowId=searchFlow&folderUri=/public/Samples/Reports`

```
http://<host>:<port>/<context>/flow.html?_
flowId=searchFlow&folderUri=/public/Samples/Reports
```

With the default deployments, the `<context>` refers to:

- `jasperserver-pro` in the commercial editions
- `jasperserver` in the community project

General Parameters

The following parameters can be used for any URL that points to a JasperReports Server web page. The parameters are specified as standard HTTP GET parameters (that is, they are in the form `name=value` and are separated by ampersands (&)). For dashboard URLs in JasperReports Server 6.0 and 6.0.1, general parameters must be placed before the # sign.



If the server is hosted by Tomcat 8.5.x and higher, you need to encode all pipe symbols (`|`, encoded as `%7C`) in the parameter to make the parameter URL-safe. For example, the format for a user in an organization would be `j_username=userID%7CorgID`.

- `userLocale` – It specifies the locale of the user. It can be added to the startup URI, as in this example:

```
http://localhost:8080/jasperserver-pro/login.html?userLocale=en_US
```

- `j_username` and `j_password` – Pass these credentials to authenticate a user with the server. The user name must correspond to a valid user, and `j_password` must 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 for a user in an organization is `j_username=userID|orgID`.

If the credential parameters are not present, no authenticated server session exists, and the server is not configured to use automatic authentication (such as Single Sign-on), the server prompts the user for a username and a password, then redirects the user to the requested page. These authentication parameters let the user skip the login page and directly access a specific page.

- `theme` – specifies a theme to use for the JasperReports Server interface. Once a theme has been set, it remains the default until the user log out, the session ends, or another theme is set.
- `decorate` – `decorate=no` removes the page headers, footers, and borders from the requested page. Page titles are not removed. The default is `decorate=yes`.
- `sessionDecorator` – `sessionDecorator=no` removes the page headers, footers, and borders from the JasperReports Server user interface for the entire user session. Once `sessionDecorator` has been set (yes or no), the setting remains the default until the user log outs, the session ends, or `sessionDecorator` is called again. As with `decorate`, page titles are not removed.



You can also remove decorations using a custom theme. See [Using Themes to Remove Branding](#) for more information.

Executing ReportUnits

The following sections provide examples and details about URLs that execute reports.

Simple Report Execution

The HTTP interface can execute and export reports within the JasperReports Server web application.

The following example calls a report with no parameters and exports to the default format (HTML):

The URL is:

```
http://<host>:<port>/<context>/flow.html?_flowId=viewReportFlow&reportUnit=/public/Samples/Reports/SalesByMonthReport
```

This is the simplest possible report execution URL. The following section explains more advanced options.

Passing ReportUnit Execution Parameters

Reports support the following types of parameters:

- General parameters supported by all URLs, such as `theme` or `userLocale`.
- Reserved report parameters such as `output` or `reportLocale`.
- Input control parameters, described in [Input Control Parameters for Reports and Dashboards](#).

The following example executes the same report as the previous section, but also passes 7 as an input control parameter and exports to PDF instead of HTML:

```
http://<host>:<port>/<context>/flow.html?_flowId=viewReportFlow
&reportUnit=/public/Samples/Reports/SalesByMonthReport&startMonth=7&output=pdf
```

Note the URL parameters:

- `&startMonth=7` indicates that the value 7 should be passed to the input control called `startMonth`. The report returns data starting in July.
- `&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:

```
http://<host>:<port>/<context>/flow.html?_flowId=viewReportFlow&reportUnit=
/reports/SalesbyState&country=US&country=Canada&output=pdf
```

The report execution parameters can either be reserved parameters the server uses to determine the general attributes of the report execution, or arbitrary parameters that correspond to the report's input controls/parameters. The parameters are specified as standard HTTP GET parameters (in `name=value` form and separated by ampersands (&)).

Reserved Report Parameters

The following reserved report parameters are recognized by JasperReports Server:

- `reportUnit` – specifies the URI of the report unit 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 (Paginated), `xlsNoPag` for Excel, `xlsx` for XLSX (Paginated), `xlsxNoPag` for XLSX, `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 where the report should be executed. A locale is passed as code consisting of a lower-case two letter ISO-639 language code, followed by an 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).
- `pageIndex` (optional) - specifies the initial page to 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 effective only for HTML 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 effective only when `pageIndex` is not specified and when the output is HTML.

Executing Dashboards

As of JasperReports Server 6.0, dashboards do not use the `flow.html` entry point used by Spring Web Flow. Instead, they use an updated dashboard entry point.



If you are working with legacy dashboards created in versions before 6.0, these dashboards work similar to reports.

The following example opens a dashboard for viewing:

```
http://<host>:<port>/<context>/dashboard/viewer.html#/public/Samples/Dashboards/1._Supermart_
Dashboard
```

To open a dashboard for editing, call the Dashboard Designer:

```
http://<host>:<port>/<context>/dashboard/designer.html#/public/Samples/Dashboards/1._Supermart_
Dashboard
```



The dashboard Resource URL must go after the hash sign (#). This syntax supports faster page refreshing. When you save a new or existing dashboard, only the URL segment after the hash sign is reloaded.

Passing Dashboard Execution Parameters

Dashboards support the following types of parameters:

- General parameters supported by all URLs, such as `theme` or `userLocale`.
- The reserved dashboard parameters `viewAsDashboardFrame` and `dashboardResource`
- Input control parameters, described in [Input Control Parameters for Reports and Dashboards](#).

As of JasperReports Server 6.1, you can use non-authentication parameters before or after the hash sign (#). However, if you place general parameters after the hash sign, then on Dashboard Viewer load, the parameters are moved before the hash sign and the page is redirected to the resulting URL. The authentication parameters `j_username` and `j_password` cannot be redirected; they must appear before the hash sign.

For example, suppose you enter the following URL:

```
http://<host>:<port>/<context>/dashboard/viewer.html?theme=pods_summer  
#/public/Samples/Dashboards/1._Supermart_Dashboard&userLocale=de&country=Mexico
```

When you go to the Dashboard Viewer, you are redirected as follows:

```
http://<host>:<port>/<context>/dashboard/viewer.html?theme=pods_  
summer&userLocale=de&country=Mexico  
#/public/Samples/Dashboards/1._Supermart_Dashboard
```

Reserved Dashboard Parameters

The following reserved dashboard parameters are available:

- `viewAsDashboardFrame` (optional) - displays the dashboard without any decoration. This setting is similar to `decorate=no`, but hides the dashboard viewer toolbar in addition to the page headers, footers, and borders. You can use `viewAsDashboardFrame` when you embed your dashboard in another webapp. For example:

```
http://<host>:<port>/<context>/dashboard/viewer.html?viewAsDashboardFrame=true#/public/MyDashboard
```

- `dashboardResource` (optional, deprecated) - supports earlier versions of JasperReports Server dashboard execution URLs. Use only for backwards compatibility.

```
http://<host>:<port>/<context>/dashboard/viewer.html?dashboardResource=/public/MyDashboard
```



When your application uses this format, JasperReports Server automatically redirects it to the new dashboard viewer format. For example, the URL above would be redirected to:

```
http://<host>:<port>/<context>/dashboard/viewer.html#/public/MyDashboard
```

Input Control Parameters for Reports and Dashboards

In addition to the standard parameters, report and dashboard execution URLs can contain parameters that provide values for input controls/parameters. The URL parameter names must match the name of the corresponding input control. The values used for such URL parameters depend on the type of input control:

- For filters or 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 the 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 as described in the `jasperserver_config_<locale>.properties` file for the current locale. See the *JasperReports Server Administrator Guide* for more information on formatting date and time. The following table shows the formats for the default locale.

Input Control Type	URL Parameter Format	Example
date.format	yyyy-MM-dd	2021-06-28

Input Control Type	URL Parameter Format	Example
datetime.format	yyyy-MM-dd 'T' HH:mm:ss	2021-06-28T13:45:22
time.format	HH:mm:ss	13:45:22

- For Boolean (check box) input controls, the URL parameter value is either `true` or `false`.
- For input controls that refer to a static list of values, the URL parameter value is the key/value of the list entry. For example, to select the 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 used. For example, `parameter=value1¶meter=value2¶meter=value3`.
- You can use the following special values for input controls:
 - `no value` - when a string parameter is called with no value, it finds all instances where the string is empty, for example, `&Country=`. When the database result set does not include an empty string value, this URL input is ignored.
 - `~NULL~` - finds NULL value. For example, `&Address=~NULL~` returns all records with a NULL address. When the database result set does not include any NULL values, this URL input is ignored.
 - `~NOTHING~` - depends on the form of the input control:
 - For an input control that supports multi-selection, `~NOTHING~` clears the current selection and resets the input control to all values. (The generated SQL query in this case is `0=0`, which essentially omits the filter.) For example, consider a multi-value input control named `MultiInput`. By default, this parameter is a list of two values. To override this list of values with another set of two values, 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=~NOTHING~`.

- For a single-select, non-mandatory input control, ~NOTHING~ selects --.
- For a single-select, mandatory input control, ~NOTHING~ selects the default value.

If the parameter values included in a report or dashboard 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 they can correct the values.

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:

```
http://<host>:<port>/<context>/fileview/fileview/public/Samples/Reports/05._Accounts_Report.pdf
```

Viewing Resources in the Repository

The following example displays all resources saved in the /public/Samples/Reports folder in the repository:

```
http://<host>:<port>/<context>/flow.html?_flowId=searchFlow&folderUri=/public/Samples/Reports
```

The following example displays all resources of the type `olapview` (analysis view) saved in all folders in the repository:

```
http://<host>:<port>/<context>/flow.html?_flowId=olapViewListFlow
```

The Public JasperReports Server API

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

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.

```
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 the Jaspersoft community site (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 that are not part of the API. Note, however, that the public API is a small subset of all JasperReports Server classes, and does not provide all of the functionality you may need, in which case you must create and use other 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.

Accessing API Implementations Using the Spring Framework

Many of the implementations of the API interfaces are singletons, usually services instantiated by the Spring Framework. The Spring bean configuration files control how these singletons are created and configured, so it's 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 5.3.29 and is not meant to cover all the possible ways to configure Spring. For more information on Spring, refer to its reference documentation at <https://docs.spring.io/spring-framework/docs/5.3.29/reference/html/>.

The Spring configuration files use XML to define Java singleton instances, called beans. In the JasperReports Server web application, these files are located in 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: JasperReports Server
 - 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 the conventions above. The original file is `samples/customDataSource/webapp/WEB-INF/applicationContext-hibernateDS.xml` in the JasperReports Server distribution.

```
<!-- define a custom data source -->
<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 you want to implement. This determines which Java implementations are required. A good example is creating a custom data source, which is documented in [Custom Data Sources](#).

Once you have a Java class you want to instantiate 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. Their types must 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.

JasperReports Server Public Java API

API	Bean ID	Function
RepositoryService	repositoryService	Search, retrieve, and modify persistent objects in the repository.
EngineService	engineService	Run reports and handle report metadata.
ReportDataSourceService and ReportDataSourceServiceFactory	n/a	Implement data sources used for running reports and

API	Bean ID	Function
		other purposes.
ReportSchedulingService	reportSchedulingService	Manage report schedules.
UserAuthorityService	userAuthorityService	Manage internal users and roles.
OlapConnectionService	olapConnectionService	Manage OLAP-specific repository and model runtime.
OlapManagementService	olapManagementService	Manage OLAP server run time.
ObjectPermissionService	objectPermissionService	Search, retrieve, and modify metadata repository object permissions.

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.

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, look up, and retrieve content from the repository. The repository is hierarchical and 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. 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 you instantiate folders and resources.

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, we recommend 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`:

```
import com.jaspersoft.jasperserver.api.common.domain.ExecutionContext;
import com.jaspersoft.jasperserver.api.metadata.common.service.RepositoryService;
import com.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:

```
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:

```
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:

```
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:

```
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:

```
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());
}
}
}

```

Repository Resources

Adding a other new resource in the repository differs from adding a folder. 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.

You create new resource instances by making a special request to the `RepositoryService`, not by direct instantiation. See the following example where you create a new image resource and load it to the repository from an image file on disk:

```

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:

```

// 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 shown above where you created the image resource) on the `RepositoryService` instance:

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

You can remove a resource from the repository by calling `repositoryService.deleteResource`:

```
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();
}
```

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 you've seen in the previous section, where you created an image resource:

```
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);
```

You can retrieve the binary data of a content resource from the repository using the `getContentResourceData` method of the `RepositoryService`, as follows:

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

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 the selected filter conditions.

The only required condition for a `FilterCriteria` instance is that the returned resources' parent folder must match a given folder. You will need to set the path to the parent folder. For example:

```
FilterCriteria filterCriteria = FilterCriteria.createFilter();
filterCriteria.addFilterElement(FilterCriteria.createParentFolderFilter("/examples"));
List resources = repositoryService.loadResourcesList(context, filterCriteria);
```

The `loadResourcesList` method returns a list of `ResourceLookup` objects that contain basic resource attributes like the name and label. To retrieve the full resource definition, you must use the `getResource` method. You can apply further filtering 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 JRXML files in a folder:

```
FilterCriteria filterCriteria = FilterCriteria.createFilter(FileResource.class);
filterCriteria.addFilterElement(FilterCriteria.createParentFolderFilter("/examples"));
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.

Engine Service

The engine service includes methods related to report execution. The `engineService` interface includes the `getReportExecutionStatusList` and `getSchedulerReportExecutionStatusList` methods for listing running report jobs or scheduled running report jobs.



The `engineService` is used internally for report execution. Except for the `getReportExecutionStatusList` and `getSchedulerReportExecutionStatusList` methods, methods in the `engineService` service should not be accessed directly.

To retrieve the list of all currently running instances of the All Accounts report, you would use code similar to the following. The list of report jobs retrieved includes scheduled jobs as well as jobs users are running via the UI:

```
ReportExecutionStatusSearchCriteria criteria = new
    ReportExecutionStatusSearchCriteria();
criteria.setJobLabel("All Accounts");
List<ReportExecutionStatusInformation> reportExecutionList =
    engineService.getReportExecutionStatusList(criteria);
```

To retrieve only currently running scheduled instances of the All Accounts report, you would use code similar to this:

```
SchedulerReportExecutionStatusSearchCriteria criteria = new
    SchedulerReportExecutionStatusSearchCriteria();
criteria.setJobLabel("All Accounts");
List<ReportExecutionStatusInformation> reportExecutionList =
    engineService.getSchedulerReportExecutionStatusList(criteria);
```

Once you have a list of jobs, you can cancel those jobs using the `reportExecutionStatusInformation` interface:

```
for (ReportExecutionStatusInformation reportExecution : reportExecutionList) {
    reportExecution.cancel();
}
```

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 the `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 [Custom Data Sources](#).

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.

Report Jobs

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 - You can schedule a job to repeat at fixed time intervals like every 4 hours, every 2 days, or every week. The job start date attribute is used to specify the moment of the first occurrence. You can specify the number of times the job should occur or an end date for the job.
- Calendar recurrence - You can schedule a job to repeat at specified calendar moments like 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. You specify the report output formats and the repository location where the

report output is saved. You can also specify one or more addresses to which email notifications are 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:

```
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:

```
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:

```
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. To set the output filename and format, use code similar to the following:

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

You can send your output to several different locations. The most common output destination is the repository. Alternative output locations include an FTP server or a user's local drive. To output to the jasperserver repository, use code similar to this:

```
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);
```

To upload the output to an FTP server instead of the repository, use code similar to this:

```
FTPInfo ftpInfo = new FTPInfo();
ftpInfo.setUsername("jsmith$mycompany");
ftpInfo.setPassword("_____");
ftpInfo.setFolderPath("/Shared/Users/JSmith");
ftpInfo.setServerName("ftp-mycompany.ftpserver.com");
job.getContentRepositoryDestination().setOutputFTPInfo(ftpInfo);
```

To send the output to a user's local drive, use code similar to this:

```
job.getContentRepositoryDestination().setOutputLocalFolder("C:\\Users\\JSmith\\JRS");
```

Optionally, you can instruct the reporting scheduler to send a notification once the job completes. This notice normally goes to the user who created the report:

```
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
mailNotification.setSkipNotificationWhenJobFails(true);
//prevents email for failed jobs
job.setMailNotification(mailNotification);
```



ReportJobMailNotification has a number of field types, including RESULT_SEND, which embeds a link in the email, RESULT_SEND_ATTACHMENT that sends the results as a zipped attachment, RESULT_SEND_ATTACHMENT_NOZIP, and RESULT_SEND_EMBED, which embeds the results as HTML content in the email. RESULT_SEND can be used only when the output is saved to the repository using setContentRepositoryDestination().

You can also optionally email an alert to the job creator, an administrative user, or both. For example, to send an alert when the job fails, use a code similar to the following:

```
ReportJobAlert alert = new ReportJobAlert();
alert.setRecipient(ReportJobAlert.Recipient.ADMIN);
// sets first recipient. Other options are OWNER, BOTH, NONE
alert.setMessageText("Report failed"); //the message body
alert.setJobState(ReportJobAlert.JobState.FAIL_ONLY);
ArrayList<String> to_Addresses = new ArrayList<String>();
//list of additional addresses to receive cc
to_Addresses.add("admin@smith.com");
to_Addresses.add("admin.smith@gmail.com");
alert.setToAddresses(to_Addresses);
job.setAlert(alert);
```



The ADMIN recipient is set by role. The default is ROLE_ADMINISTRATOR. The ADMIN recipient can be changed in the `administratorRole` bean in `<js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/applicationContext-report-scheduling.xml`.

For example, suppose you have used Manage > Roles in the JasperServer user interface to add the role ROLE_SCHEDULER_ADMIN. To set that user as the recipient for all ADMIN email alerts, modify the `administratorRole` bean as follows:

```
<entry key="administratorRole" value="ROLE_SCHEDULER_ADMIN">
```

Report Job Model

A report job model allows you to specify scheduling and output attributes for multiple report jobs at the same time. A report job model takes a list of report job IDs and a single set of attributes for all report jobs in the list.

A report job model definition consists of:

- Scheduling attributes. Instruct the scheduler when to execute the job. A report job model can specify a one-time job, a simple recurring job, or a calendar recurring job.
- Output attributes. Instruct the scheduling service what to do with the report output.

You need to define only those attributes you want to update. Empty attributes are left unchanged in the original report jobs.

A report job model definition is an instance of the `com.jaspersoft.jasperserver.api.engine.scheduling.domain.ReportJobModel` bean class, which extends `ReportJob`.

To instantiate a new report job model definition, you should use code similar to the following:


```
ReportJobModel jobModel= new ReportJobModel();
```

To set the destination, mail notification, and schedule for a report job model, you should use code similar to the following:

```
ReportJobRepositoryDestinationModel destinationModel = new
    ReportJobRepositoryDestinationModel();
destinationModel.setFolderURI("/test/operations");
jobModel.setContentRepositoryDestinationModel(destinationModel);
ReportJobMailNotificationModel mailNotificationModel = new
    ReportJobMailNotificationModel();
mailNotificationModel.setSkipNotificationWhenJobFails(false);
//send email even when job fails
mailNotificationModel.setIncludeStackTraceWhenJobFails
//send stack trace when job fails
jobModel.setMailNotificationModel(mailNotificationModel);
Date startDate = ...
ReportJobSimpleTriggerModel trigger = new ReportJobSimpleTriggerModel();
trigger.setStartDate(startDate);
trigger.setOccurrenceCount(20);
trigger.setRecurrenceInterval(10);
trigger.setRecurrenceIntervalUnit(ReportJobSimpleTrigger.INTERVAL_DAY);
jobModel.setTriggerModel(trigger);
```

Any report job attributes not updated by `ReportJobModel` remain unchanged. In the example above, the existing email recipient, subject, and message remain the same for the updated jobs, as well as any existing alerts.

The report scheduling service also uses `ReportJobModel` to retrieve a list of all active report jobs that match the attributes specified in the report job model. The list of report jobs retrieved from `ReportJobModel` can be sorted using the `ReportJobSortType` subclass of `ReportJobModel`. See [Report Scheduling Service](#) for more information.

Report Job Summary

Report job summaries hold information returned by the scheduler service. The `ReportJobSummary` class has a number of methods that allow you to retrieve specific information about the summary. Some useful methods are:

- `getStateCode`: Returns the execution state of the report job, such as `STATE_COMPLETE`, `STATE_EXECUTING`, etc.
- `getNextFireTime`: Returns the next time the job is scheduled to run, `null` if not scheduled in the future.
- `getPreviousFireTime`: Returns the last time that the job was run, `null` if the job has not yet been executed.

Report Job ID Holder

The `ReportSchedulingService` assigns every active report job a report job ID. This ID can be wrapped using the `ReportJobIDHolder` class. `ReportJobIDHolder` is used by some report scheduling API methods.

Report Scheduling Service

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:

```
ReportSchedulingService schedulingService = ...
ReportJob job = ...
schedulingService.scheduleJob(executionContext, job);
```



The `ReportSchedulingService` is not a Java object 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 to produce the desired behavior. For more information on defining a bean, see [Accessing API Implementations Using the Spring Framework](#).

The scheduling service also contains methods for the removing one or more existing report jobs: `removeScheduledJob` and `removeScheduledJobs`.

To load the full report job definition for a job, use the `getScheduledJob` method. You can alter the job definition then update it using the `updateScheduledJob` service method.

You can alter and update a list of report jobs using the `updateScheduledJobs` service method. `updateScheduledJob` takes a `ReportJobModel` and updates the specified jobs to match the attributes in the `ReportJobModel`. Unspecified attributes remain unchanged. The trigger is handled somewhat differently from the other attributes. A Boolean parameter, `replaceTriggerIgnoreType`, specifies one of two options:

- `true`: Replace the trigger in all listed report jobs with the trigger in the `ReportJobModel`.
- `false`: If a trigger is present in the report job model, check that all listed report jobs have the same trigger type (`SimpleTrigger` or `CalendarTrigger`) as the report job

model. If one or more report jobs have a different trigger type, the update fails for all report jobs.

For example, to update two report jobs, including the trigger, to match the parameters in a specified report job model, you should use code similar to the following:

```
List<ReportJob> reportJobs = new ArrayList<ReportJob>();
reportJobs.add(job_01);
reportJobs.add(job_02);
ReportJobModel jobModel = ...
updateScheduledJobs(executionContext, reportJobs, jobModel,
true) //replaceTriggerIgnoreType - when true, replace trigger in the report jobs
with the trigger from the model.
```



Use caution when updating the job trigger, because the original Quartz trigger is dropped and a new trigger that corresponds to the updated attributes is created.

Two methods of the scheduling service let you retrieve a list of jobs. The retrieved list consists of instances of `com.jaspersoft.jasperserver.api.engine.scheduling.domain.ReportJobSummary` that contain basic job attributes, plus runtime attributes like job status and previous/next fire times.

- `getScheduledJobSummaries` can be used as follows:
 - Retrieves job summaries of all active report jobs defined in the scheduler.
 - With a `reportUnitURI`, retrieves the list of job summaries for scheduled jobs for a report unit.
 - With a `ReportJobModel` along with other parameters, retrieves the list of scheduled job summaries that match the criteria specified by the `ReportJobModel`.
- `getJobsByNextFireTime` retrieves the list of all jobs with a next fire time in a specified time interval.

For example, to get all active jobs that match a report job model, you should use code similar to the following:

```
ReportJobModel reportJobCriteria = ...
int startIndex = 5 //return all jobs after the first 5 (starts with 0)
int numberOfRows = 30 //number of jobs returned per page
getScheduledJobSummaries(executionContext, reportJobCriteria,
startIndex,
numberOfRows,
SORTBY_REPORTNAME, //value of enum class ReportJobModel.ReportJobSortType
true); //when true, sort in ascending order
```

To get all scheduled jobs whose next fire time is between `startDate` and `endDate`, you should use a code similar to the following. The next fire time is the value retrieved by `ReportJobSummary.getNextFireTime`.

```
List<ReportJobSummary> jobList = new
Date startDate = ...
Date endDate = ...
List<ReportJobSummary> jobList =
    schedulingService.getJobsByNextFireTime(executionContext,
        null, startDate, endDate, null);
```

The `pause` and `resume` methods let you pause or resume a list of jobs. To pause all the jobs retrieved by the previous call, use code similar to the following. If an alert is set, pausing the job triggers the alert:

```
schedulingService.pause(jobList, true);
```

The `pauseById` and `resumeById` methods let you pause or resume a `ReportJobIDHolder` list.

Report Jobs Scheduler

The `ReportJobsScheduler` service includes methods for adding, getting, and deleting calendars. Calendars in `ReportJobsScheduler` service are implementations of the `QuartzCalendar` interface. They specify times during which scheduled jobs are not fired. For information about Quartz calendars, see the [Quartz API documentation](#).



The `ReportJobsScheduler` service is used internally by the central scheduling service. Except for the calendar methods, most methods in the `ReportJobsScheduler` service should not be accessed directly by the report scheduling code.

For example, to register the Quartz calendar `myHolidayCalendar()`, with the `ReportJobsScheduler`, you should use a code similar to the following:

```
ReportJobsScheduler.addCalendar("US Holiday Calendar",
    myHolidayCalendar(),
    true, //overwrite any existing calendar with the same name
    true) //update existing triggers that refer to this calendar
```

Other methods allow you to delete a calendar, retrieve a calendar by name, or get a list of the names of all registered calendars: `deleteCalendar`, `getCalendar`, and `getCalendarNames`.

Once you have added a calendar, you can use it in any number of job triggers. For example, to create a trigger that tells a job to run every Monday at 1:00 A.M., unless it is a United States holiday as specified in `myHolidayCalendar()`, you should use code similar to the following:

```
ReportJobCalendarTrigger weeklyTrigger = new ReportJobCalendarTrigger();
weeklyTrigger.setMinutes("0");
weeklyTrigger.setHours("1");
weeklyTrigger.setDaysTypeCode(weeklyTrigger.DAYS_TYPE_WEEK);
weeklyTrigger.setWeekDays(2); //sets the day of the week; Monday is 2
weeklyTrigger.setTimezone("America/Los_Angeles");
weeklyTrigger.setStartType(weeklyTrigger.START_TYPE_NOW);
weeklyTrigger.setCalendarName("US Holiday Calendar");
//tells the job not to run on holidays as specified by this calendar
```

You can use a trigger like this to set the holiday calendar for a number of `ReportJobs` using `ReportJobModel`. Set `replaceTriggerIgnoreType` to `true` to ensure that the trigger is updated for all `ReportJobs`.

Users and Roles API

Access to all JasperReports Server functionality is based on assigned user-level and role-level permissions. So 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 - represented by the `com.jaspersoft.jasperserver.api.metadata.user.domain.User` interface.
- Roles - represented by the `com.jaspersoft.jasperserver.api.metadata.user.domain.Role` interface.

You can define a new user in a few easy steps:

```
User workingUser = userAuthService.newUser(null);
workingUser.setUsername("john");
workingUser.setTenantID("organization_1");
workingUser.setPassword("changeme");
workingUser.setFullName("John Doe");
workingUser.setEnabled(true);
workingUser.setExternallyDefined(false);
userAuthService.putUser(null, workingUser);
```

The `setTenantId` method specifies the organization that the user belongs to. However, note the following:

- If you are using commercial editions of JasperReports Server, you should use this method in most cases, but if your instance hosts only a single organization, this method should set most user's organization to the default (`organization_1`).
- If you are defining a special administrative user (similar to superuser) that should not be affiliated with an organization, do not call `setTenantId`.

To get the user information from the database, you can call the `getUser` method by providing the username.

You can remove users 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:

```
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.

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 shows how to create an object permission on an image resource within the repository:

```
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);
```

You can revoke object permissions with one of the following public methods exposed by this service:

```
public void deleteObjectPermission(ExecutionContext context,
    ObjectPermission objPerm);
public void deleteObjectPermissionForObject(ExecutionContext context,
    Object targetObject);
public void deleteObjectPermissionsForRecipient(ExecutionContext context,
    Object recipient);
```

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**:

```
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:

```
OlapUnit olapUnit = (OlapUnit)
    getRepository().getResource(executionContext,viewUri);
OlapModel model =
    getOlapConnectionService().createOlapModel(executionContext, olapUnit);
```

The following call retrieves the server-wide Mondrian properties:

```
// 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:

```
public ValidationResult validate(ExecutionContext context, OlapUnit unit);
public ValidationResult validate(ExecutionContext context,
    OlapUnit unit,
    FileResource schema,
    OlapClientConnection conn,
    ReportDataSource dataSource);
```

Flushing the OLAP Cache Using the API

Jaspersoft OLAP maintains a cache of previously-retrieved results in order to achieve high performance. A simple interface enables you to flush the entire cache programmatically, an action required when new data is inserted into databases while Jaspersoft OLAP is running.

```
public interface OlapManagementService {
    public void flushOlapCache();
}
```


You can configure the interface as a Spring bean or instantiate it with a standard Java constructor call. The default implementation of the interface is `com.jaspersoft.jasperserver.api.metadata.olap.service.impl.OlapManagementServiceImpl`.

Customizing the User Interface

JasperReports Server is highly customizable because it's 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, listed here from simplest to most complex.

- Themes – The themes expose the CSS of the UI through the repository. This makes it easy for administrators change to the appearance of the UI, such as images, colors, font size, spacing, and even the general layout. Themes are described in detail in the *JasperReports Server Administrator Guide*. Refer to that document first.
- SiteMesh – The SiteMesh decorator 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're familiar with JSP and JavaScript syntax.
- Action Model – This 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 Web Flow 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.

You can modify themes in a running server and changes will be seen immediately by all users. For all other types of customization, you need to edit the files deployed in the web app. If you're 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](#)
- [Customizing the UI With Web App Files](#)
- [Customizing the Branding with SiteMesh](#)
- [Customizing JSP and JavaScript for the Login Page](#)
- [Setting the Home Page](#)
- [Adding View List Buttons to the Getting Started Page](#)
- [Restricting Access to a Location in the Repository](#)
- [Customizing the JasperReports Wizard Repository Trees](#)
- [Customizing Menus](#)
- [Customizing the Report Rendering Page](#)
- [Working With Custom Java Classes](#)
- [Adding a Custom JSP Page in a Spring Web Flow](#)
- [Customizing the Scheduler Pages](#)
- [Upgrading With UI Customizations](#)

Changing the UI With Themes

You can modify the look of the JasperReports Server user interface by creating a theme: a collection of CSS files and associated images that specify the appearance of all or part of the UI. The following sections show how to use themes to change the logo and favicon, change colors and fonts, change the amount of white space, and hide UI elements. Using themes, you can customize the branding of the server or completely remove the branding for embedding.

A theme doesn't change the UI controls or provide security; hidden elements can still be accessed using the correct URL. You need to change the JSP files that control the logic of the server to control *what* users see. Themes only control *how* they see it. Additional customizations are described elsewhere in this guide.

Theme files are stored in the repository and can be downloaded, uploaded, and made active through the UI while the server is running. You don't need to recompile any JasperReports Server source code or restart the server. For information on how to create, upload, and administer themes, see the *JasperReports Server Administrator Guide*.



You can upload themes or individual theme files in several ways. You can also upload CSS files and images into an active theme if you already have a custom theme created on the server. For implementation details, see the *JasperReports Server Administrator Guide*.

Keep the following tips in mind when working with themes:

- Plan your theme deployment carefully and test it with end-users to ensure it works the way you intend.
- The server must be running and you must be logged in as an administrator to modify themes. Modifications take effect when you make your new theme files active. You don't need to recompile any code or restart the server.
- You must create or modify your CSS files in an external editor. Once you have modified the files you want, you can create a theme folder directly in the repository, or create a theme folder offline and upload it as a Zip archive file. Use the method that works best for you.
- Your themes can be as simple or as complex as you want. You can combine multiple customizations in a single theme; you can also override one file in your theme with another file.
- In multi-organization deployments, themes defined in a parent organization are expressed in child organizations. Depending on your needs, you may want to implement your theme at the root level or at an organization level. You can also create a set of common customizations in a parent organization and override them in individual child organizations. See the *JasperReports Server Administrator Guide* for details on the organization hierarchy and how it works with themes.



Customizations for dashboards created in JasperReports Server 6.0 and higher can be placed in `overrides_custom.css`.

Changing the Logo and Favicon

One very simple way to customize JasperReports Server is to replace the logo and/or favicon with your own images.

The Jaspersoft logo is white on a transparent border and is displayed on the blue background of the default theme. If your logo isn't visible on this background, you can change the background color to suit your needs, as explained in [Changing Colors and Fonts](#). You can change the logo either of these ways:

- Replace the logo file with your own. If your logo has roughly the same dimensions as the Jaspersoft logo you can simply replace the logo file. The Jaspersoft logo is 200 pixels wide by 23 pixels high.
- Change the CSS to load your logo file. If you can't make your logo to fit the CSS, edit the CSS to load your logo.

To change the favicon, replace the favicon file. The favicon is 16 pixels wide by 16 pixels high and saved as a .ico file.

For additional ways to change the branding of the server, see [Editing decorator.jsp for Rebranding](#).

Replacing the Files

One way to replace the logo and/or favicon is to create an image file with the same file name as the one provided with the server. When you create a theme with this file in the same path as the logo or favicon file, your image is displayed instead. This is the easiest customization.

To replace the logo and/or favicon file with your own

1. Edit a theme or create a new one. Make sure it includes a folder named images.
2. Do one or both of the following:
 - Replace the logo. Convert your logo image to the SVG format, save it with the filename `logo_reverse.svg`, and copy it to the images folder.
 - Replace the favicon. Convert your favicon to the ICO format, save it with the filename `favicon.png`, and copy it to the images folder.
3. Upload and activate the new theme to the chosen location and click your browser's Refresh button. Your logo appears in the top-left corner of every page:

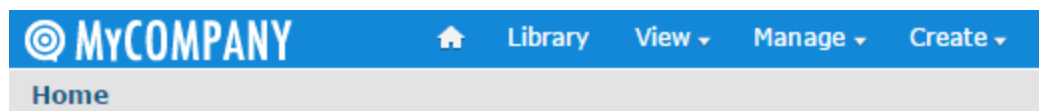


Figure 10: MyCompany Logo on Home Page

Modifying the CSS File

Another way to replace the logo is to modify the CSS files in the theme to reference your own image files. This creates a new filename and path. Use this procedure if your logo can't be modified to fit. The Jaspersoft logo is 200 pixels wide by 23 pixels high.

To modify the CSS file to use your own logo file

1. Edit a theme or create a new one. In this theme, you can place your image file in any folder structure you want.
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 rules like those below. In this example, the logo is a file called `MyCompanyLogo.png` in a folder called `MyImages` and is 40px x 230px. Adjust all pixel values based on the dimensions of your image:

```
#logo { /* new logo image name and size (example is twice the original logo) */
  background: url("images/MyCompanyLogo.png") 0 0;
  height: 40px;
  width: 230px;
}
.banner { /* increases height of banner to accommodate bigger logo */
  height:65 px
}
#frame { /* moves the main frame a bit down to accommodate bigger logo */
  top: 66px;
}
```



The IDs and classes used in the CSS are dependent on the JSP source code for the server. Look at the JSP files to see which IDs and property values are used. Alternatively, you can inspect the HTML and CSS returned by the server, for example by using the developer tools in Firefox. For more information, see the *JasperReports Server Administrator Guide*.

4. Upload and activate the new theme with your images and CSS file and click your browser's Refresh button.

Changing Colors and Fonts

Changing colors, fonts, size, and spacing throughout the UI is very simple using themes. However, creating a complete theme that modifies all aspects of the UI involves changing many CSS rules and re-creating multiple images of buttons and window decorations.

To provide an example of theme customizations, the sample data installed with JasperReports Server includes three alternate themes: `easy_access`, `Pods_summer`, and `Jasper_dark`. You can activate a theme to see its effect and download its files to see how it works.

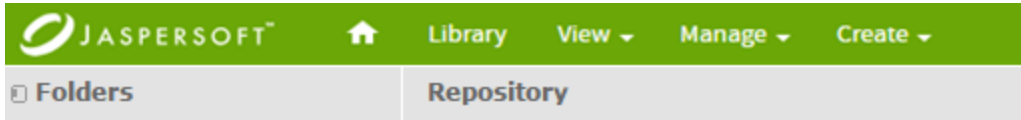


Figure 11: Menu bar in the Sample Theme `Pods_summer`

Like most small-scale customizations, the `Pods_summer` theme uses a single `overrides_custom.css` file and several files for images and sprites. 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.

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. 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.

Hiding UI Elements

Setting the parameter `display:none` is a convenient way to hide any element through the CSS. If you set `display:none` as an override, the corresponding element is not rendered in the browser. You can use `display:none` alone or in addition to other CSS parameters.



The hidden element still exists in the HTML transmitted to browsers, so don't 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 [Restricting Access by Role](#).

The following example shows how to use this parameter to remove the logo or the search box from the UI.

To remove the logo and/or the search box

1. Edit a theme or create one.

2. If necessary, copy the `overrides_custom.css` file from the default theme to the main folder of your theme.
3. Do one or both of the following:

- Remove the logo. Edit the `overrides_custom.css` file and add the following CSS rules:

```
#logo {  
  display:none;  
}
```

- Remove the search box. Edit the `overrides_custom.css` file and add the following CSS rules:

```
.searchLockup {  
  display: none  
}
```

4. Upload and activate the new theme with your image and CSS file, then click your browser's Refresh button.

You can also use this attribute to hide the footer in the UI. For instructions on changing the text in the footer, see [Editing decorator.jsp for Rebranding](#).

To remove the footer

1. In your `overrides_custom.css` file, add the following CSS statement to hide the footer:

```
#frameFooter {  
  display: none;  
}
```

2. Save and upload your `overrides_custom.css` file to your theme.

Using Themes to Remove Branding

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 page content, so by changing a few rules, you can rearrange the default layout in many ways.



Many CSS attributes affect spacing, including margins, borders, and edge and corner images. Also, the various `div` elements that make up the UI interact, 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.

If you embed JasperReports Server in your application, you may want to remove all branding and menus. The following example shows how to remove the banner and footer entirely using themes.



Removing the menus restricts your ability to navigate the JasperReports Server UI. If you need to revert to the default theme, change your theme for the session by adding `&theme=default` to the end of the current URL. You can also log in as an administrator, select **Manage Server**, then select **Repository** to locate and change themes.

1. Create a new theme in JasperReports Server. This example assumes you named the theme "embed".
2. On your computer, create the following CSS file and save it as `overrides_custom.css`.

```
/*
overrides_custom.css
Basic theme for embedding
*/
body {
  background-color: white;
  background: none;
}
#banner {
  background: none;
  display:none;
}
#frame {
  top: 0;
  bottom: 0;
}
#frameFooter {
  display:none;
}
```



If you're rebranding JasperReports Server, you must use the phrase "Powered by Jaspersoft" on any distributed reports or report portal. You may not remove or delete any of Jaspersoft's copyright or other proprietary notices.

3. Log into JasperReports Server as an administrator.
4. Select **View > Repository** and navigate to the location where you created your theme.
5. Right-click the embed theme in the repository and select **Add Resource > File > CSS**.
6. Select the `overrides_custom.css` file you created and enter `overrides_custom.css` for the Name and Resource ID.

7. Click Submit.
8. Right-click the embed theme and select Set as Active Theme. The result is a theme that has no footers or menus.

Repository

Folders	Run	Edit	Open	Copy	Cut	Paste	Delete	Sort By: <u>Name</u>	Modified Date
root									
Organizations									
Public									
Ad Hoc Components									
Audit									
Diagnostic									
Monitoring									
Samples									
Ad Hoc Views									
Dashboards									
Data Sources									
Domains									
OLAP									
Reports									
Resources									
Templates									
Temp									
Themes									
	Name	Description		Type	Created...	Modifie...			
	01. Geographic Results by Seg...	Sample Report containing hyperlinks and a ...		Report	Yesterday	10/17/2...			
	02. Sales Mix by City Report	Sample Spider Line Report created from an A...		Report	Yesterday	8/28/2019			
	03. Store Segment Performanc...	Sample Gauge Report Created from an Ad Ho...		Report	Yesterday	10/11/2...			
	04. Product Results by Store Ty...	Sample Filtered Table Report Showing Multip...		Report	Yesterday	8/28/2019			
	05. Accounts Report	Basic interactive Table Component report wit...		Report	Yesterday	8/28/2019			
	06. Profit Detail Report	Sample report containing containing interacti...		Report	Yesterday	8/28/2019			
	07. Revenue Detail Report	Sample report containing containing interacti...		Report	Yesterday	8/28/2019			
	08. Unit Sales Detail Report	Sample report containing containing interacti...		Report	Yesterday	8/28/2019			
	09. Customer Detail Report	Sample report with multiple charts, interacti...		Report	Yesterday	8/26/2019			
	10. Performance By Store Type	Sample report with Donut Pie Charts and cro...		Report	Yesterday	10/11/2...			
	11. Sales By Month Report	Sample multi-panel report		Report	Yesterday	10/16/2...			
	12. Promotion Details Report	Drill-through from the Supermart dashboard.		Report	Yesterday	10/9/2019			
	14. World Map	Report with simple Geographic Map and Hyp...		Report	Yesterday	10/11/2...			
	16. Interactive Sales Report	Interactive table and chart.		Report	Yesterday	8/27/2019			

Figure 12: Basic Theme for Embedding

Replacing the Default Theme

In general, we recommend creating new themes with CSS and image files that override those of the provided default theme. However, there may be cases where the default theme 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.

Although it's possible to configure the server to load the theme from files, this disables the dynamic theme mechanism and has other effects. We do not recommend such a configuration.

Like all themes, the default theme is stored in the repository, in the server's private database. The theme's propagation and inheritance will work only if the theme files exist in the repository before the server is started. You can achieve this two ways:

- Use 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.

- Insert your default theme during installation. During installation, 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're 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 guide.



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 [Upgrading With UI Customizations](#).

Customizing the UI With Web App Files

Many components of the UI 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 components. 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 components use additional XML files for configuration.

Several kinds of files are involved in creating the UI:

- 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). Much of the UI can be customized with interpreted files.

- 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 used in the Spring framework, where they're also called Spring beans.

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 UI features 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 [Working With Custom Java Classes](#).

Location of Interpreted Files

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.

File Location	<code><js-webapp></code> Path
Installed server	<p>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:</p> <pre><js-webapp> = <js-install>/apache-tomcat/webapps/jasperserver[-pro]</pre> <p>For other application servers, see Customizing WAR Files.</p> <p>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.</p>
WAR file distribution	<p>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:</p> <pre><js-webapp> = <js-install>/jasperserver[-pro].war</pre>

File Location	<js-webapp> Path
	<p>To modify files with the WAR file, see Customizing WAR Files. 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.</p>
Source code	<p>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:</p> <pre data-bbox="545 705 1170 774"><js-webapp> = <js-src>/jasperserver/jasperserver-war/src/main/webapp</pre> <p>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.</p>



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.

Location of JSP Files

Inside the JasperReports Server web application, the JSP files are organized as follows:

- 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.

Customizing JavaScript Source Code

The JavaScript source code is customized to improve performance.

To customize the JavaScript source code

1. Create a working directory where you can extract sources from the JasperReports Server source bundle Zip file (TIB_js-jrs_<version>_src.zip).
2. Download and install node.js from <http://nodejs.org/>.
3. Install yarn (version 1.22.x is required).
4. Run the following commands to install node modules.
 - `cd jasperserver-ui/pro`
 - `yarn install`
5. Make your changes to the JavaScript source code.
6. Run the following commands to build the final bundled static assets.
 - `cd jasperserver-ui/pro/jrs-ui-pro`
 - `yarn run clean`
 - `yarn run build`Bundled static assets will be placed in the `jrs-ui/build/overlay/scripts` folder.
7. Back up the `scripts` folder of your JasperReports Server instance deployment. For example, `<tomcat>/webapps/jasperserver-pro/scripts` folder, when Apache Tomcat is used.
8. Delete the content of the `scripts` folder of your JasperReports Server instance deployment. For example, `<tomcat>/webapps/jasperserver-pro/scripts` folder.
9. To deploy bundled static assets (build result) to the JasperReports Server instance, copy the content of the `jrs-ui-pro/build/overlay/scripts` folder to the `scripts` folder of your JasperReports Server instance deployment. For example, `<tomcat>/webapps/jasperserver-pro/scripts` folder.



Visualize.js will be built as part of the build process of JavaScript source code. Therefore, there is no separate build process for Visualize.js while starting JasperReports Server 9.0.0.

Customizing WAR Files

When you modify 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 (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:

```
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 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.

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 for your changes to take effect. The standard installation of the server includes only 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're doing a lot of customization and testing, it's simpler and quicker to simply reload the web app without restarting the application server.

Each application server has its own management console that lets you view and control deployed web apps. This example shows how to manage the JasperReports Server web app using the GUI in Apache Tomcat.

1. If you have not configured any users on your Apache Tomcat server, you must first add a user and give it the `manager-gui` role. To do this:
 - a. Edit the file `<js-install>/apache-tomcat/conf/tomcat-users.xml`.
 - b. Create a user if necessary and give it the `manager-gui` role. You can give the user any name and password you prefer.

```
<tomcat-users>
  <role rolename="manager-gui"/>
  <user username="tomcat" password="tomcat" roles="manager-gui"/>
</tomcat-users>
```

- c. Restart your Apache Tomcat server for this change to take effect.
2. Once you have created a `manager-gui` user, 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/`.
 3. When prompted, log in with your manager user credentials.
 4. On the Apache Tomcat management interface, scroll down to find the `jasperserver[-pro]` application, and click Reload.
 5. Confirm your reload of the web app and, when the page reloads, 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 stop and restart the app server.

Customizing the Branding with SiteMesh

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/decorator.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 of the UI.

web.xml

The `<js-webapp>/WEB-INF/web.xml` configuration file contains the configuration information that enables SiteMesh. You can see that SiteMesh's `PageFilter` class is applied to all targeted URLs (that is, `<url-pattern>/*</url-pattern>`):

```
<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>
```

sitemesh.xml and decorators.xml

The SiteMesh page filter assumes that the `<js-webapp>/WEB-INF/sitemesh.xml` file specifies further configurations. You can see that the main decorator's definition points to `decorators.xml`. You'll also see the SiteMesh mapping that handles the default locale (U. S. English):

```
<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, 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 <https://github.com/sitemesh/sitemesh2>.

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. [JasperReports Server Display Elements](#) shows the display elements as they appear in JasperReports Server.

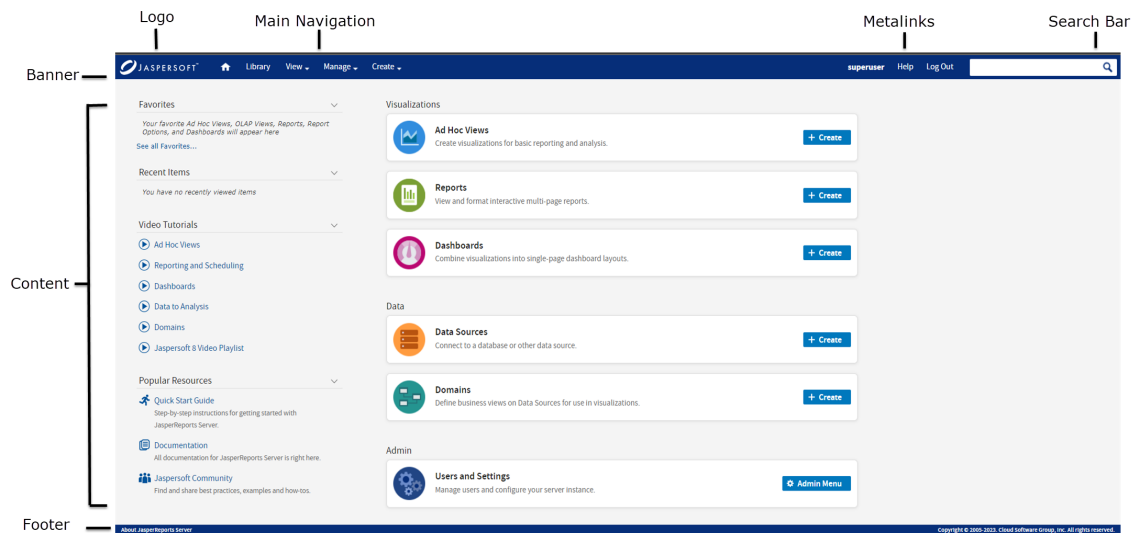


Figure 13: JasperReports Server Display Elements

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 `decorator.jsp`, you can see the structure of every JasperReports Server HTML page, with the main frame, the banner, 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="banner" class="banner">
...
<div id="logo" class="sectionLeft"></div>
...
<div class="sectionLeft">
...
<div id="mainNavigation" class="menu horizontal primaryNav">
iv class="sectionRight searchContainer">
...
<ul id="metaLinks" class="sectionRight">
...
</div>
<div id="frame" style="<c:if test="\${param['frame']} == 0">top:0;bottom:0;</c:if">"
<div class="content">
<decorator:body />
</div>
</div>
<div id="frameFooter">
...
</div>
...
</body>
</html>
```

For example, if the user clicks View > Reports, the action executes `WEB-INF/jsp/modules/ListReports.jsp`. 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.

Editing `decorator.jsp` for Rebranding

Now that you know how `decorator.jsp` defines the main page of the server UI, you 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 [Changing the Logo and Favicon](#).

- The browser icon (favicon): to change or remove the browser icon, see [Changing the Logo and Favicon](#).
- The page title appearing in the browser.
- The About link and copyright footer on every page.

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 change 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:

```
...
<div id="frameFooter">
<!-- <p id="about">
  <a href="#"><spring:message code="decorator.aboutLink"/></a>
  <c:if test="${isDevelopmentEnvironmentType}">
    <span id="license">
      (<spring:message code="LIC_023_license.envtype.development.label"/>)
    </span>
  </c:if>
</p>
  <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.



If you are rebranding JasperReports Server, you must display "Powered by Jaspersoft" on any distributed reports or report portal. You may not remove or delete any of Jaspersoft's copyright or other proprietary notices.

Setting the Home Page

The home page is the first page a user sees when logging in to JasperReports Server. You can set the home page to a specified flow based on user role.

Setting the Library as the Home Page

The following example shows how to set the library page as the home page of a non-administrative user:

1. Open the file `<js-webapp>\WEB-INF\jasperserver-servlet-pro.xml` (commercial editions) or `jasperserver-servlet.xml` (community edition) for editing.
2. Locate the home page bean:
 `proHomePageByRole` (commercial editions)
 `homePageByRole` (community edition)
3. Locate the line under this bean for `ROLE_USER` and modify it to direct to the library page:

```
<bean id="proHomePageByRole" class="java.util.ArrayList">
  <constructor-arg>
    <list>
      <value>ROLE_ADMINISTRATOR|redirect:/flow.html?_
        flowId=homeFlow</value>
      <value>ROLE_USER|redirect:/flow.html?_
        flowId=searchFlow&mode=library</value>
    </list>
  </constructor-arg>
</bean>
```

i Note:

For a user with multiple roles, the `proHomePageByRole` bean redirects the user to the page specified by the first matching role on the list. For example, for an administrator with `ROLE_ADMINISTRATOR` and `ROLE_USER` roles, the `ROLE_ADMINISTRATOR` redirect is applied because it appears first in the bean. When adding your own roles to this bean, insert them in the order that works for you.

4. Save the modified file and reload the web app in the app server to see the changes (see [Reloading the JasperReports Server Web App](#)).
5. Log into JasperReports Server as `joeuser`. The library page is displayed.

i Note:

You can use this method for dashboards. For example:

```
<value>ROLE_
ADMINISTRATOR|redirect:/dashboard/viewer.html#/public/Samples/Dashboards/1._Supermart_Dashboard</value>
```

This method does not work with `viewReportFlow`.

Setting a report as the home page

The following example shows how to set a report as the home page based on a role. This example uses the 01. Geographic Results by Segment Report.

i Note:

All roles should start with the prefix `'ROLE_'`.

For example:

```
'ROLE_MANAGER_DIME'
```

1. First, set up the role and create a sample user:
 - a. Create the role you want to use, for example, `ROLE_REPORT_HOME`.
 - b. Create a user `HomeUser` and add `ROLE_REPORT_HOME` to `HomeUser`.
2. Open the file `<js-webapp>\WEB-INF\jsp\modules\home\home.jsp` in an editor and add the following lines:

```
<authz:authorize access="hasRole('ROLE_REPORT_HOME')">
<c:redirect
    url="http://localhost:8080/jasperserver-pro-601/flow.html?_
flowId=viewReportFlow
    &standAlone=true&_
flowId=viewReportFlow&ParentFolderUri=%2Fpublic%2FSamples%2FReport
s&
    reportUnit=%2Fpublic%2FSamples%2FReports%2F01._Geographic_
Results_by_Segment_Report"/>
</authz:authorize>
```

i Note:

To find the URI for the report you want to use, open the report in the repository and copy the URI.

3. Save the modified file and restart the application server to see changes.
4. Log into JasperReports Server as HomeUser. The report is displayed.

Adding View List Buttons to the Getting Started Page

Each workflow block on the Getting Started page has a large icon you can click to open a filtered repository list. For example, you can view a list of the dashboards in your repository if you click on the icon in the Dashboards workflow block. You can add View List buttons to the workflow blocks that link to the appropriate repository list if you prefer your users didn't use the large icon.

To add View List buttons to workflow blocks

1. Edit a theme or create a new 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:

```
.workflow-actions .button.action.view
```

```
{
  display: inline-block;
}
```

4. Upload and activate the new theme with your CSS file and click your browser's Refresh button.

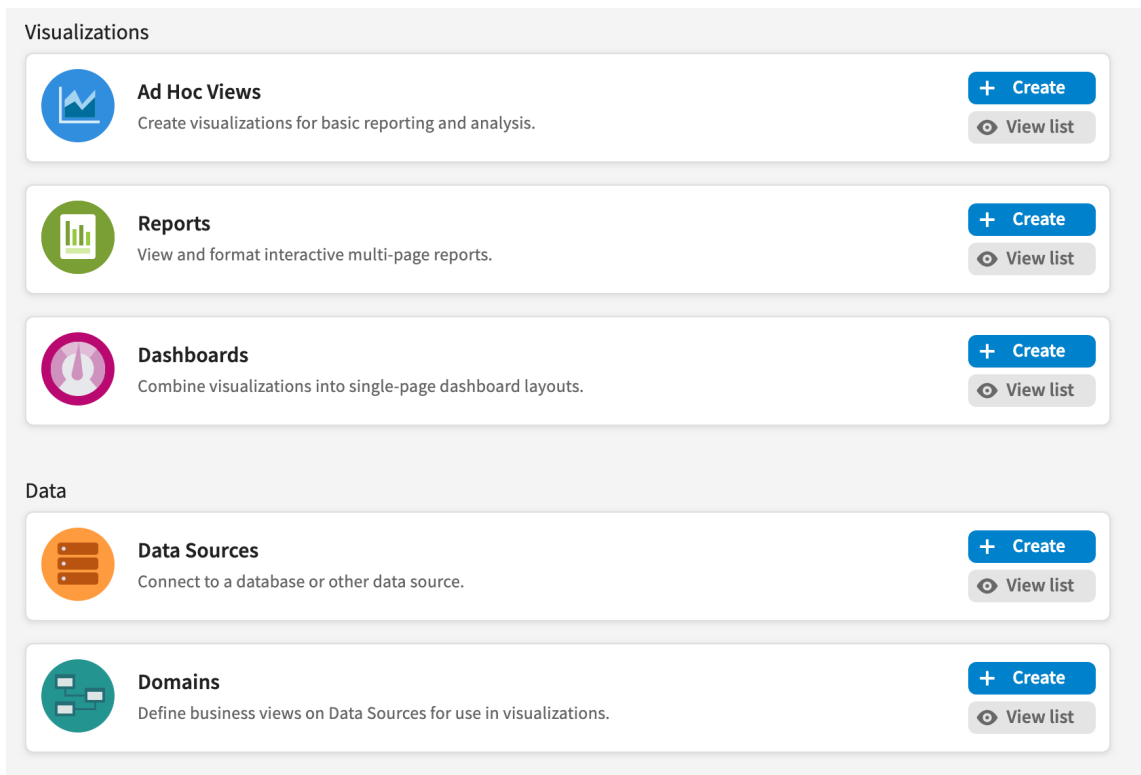


Figure 14: The Getting Started page with View List buttons in the workflow blocks

Restricting Access to a Location in the Repository

You can use role-based permissions to control access to repository locations. For example, by default, the Public > Samples > Ad Hoc Views folder is set to Read Only for ROLE_USER. This allows users to see the folder contents and to create new reports from the Ad Hoc views in this folder, but they can't edit those views.

You can hide the contents of this folder from ROLE_USER by setting repository permissions to execute only. To do this:

1. Log in as jasperadmin.
2. Select View > Repository from the menu.
3. Navigate to the Public/Samples folder.
4. Right-click the Ad Hoc Views folder and select Permissions...
5. Change the permissions for ROLE_USER to Execute Only.

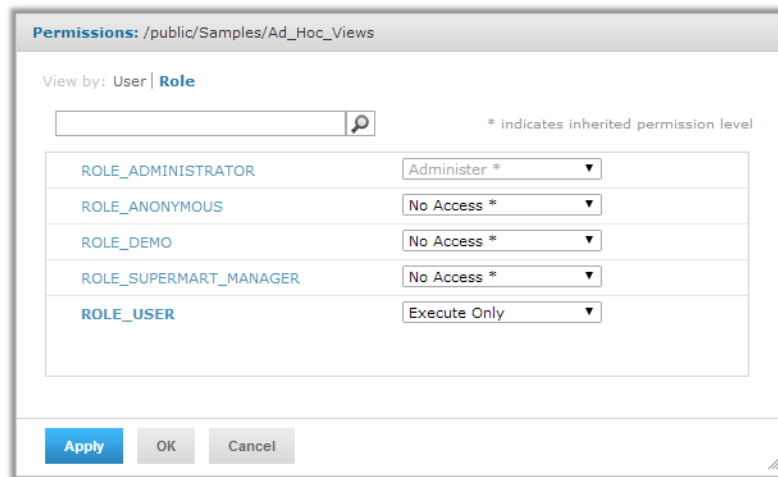


Figure 15: Setting Permissions to Execute Only

6. Click OK.

With these permissions, when users with no additional permissions select Library view, they won't see the Ad Hoc views in this folder, nor do they see the folder in the Repository. But they're still able to open reports based on the views in this folder.



When you restrict access based on file location in the repository, if a user saves an Ad Hoc view to a different location, users will still be able to access the view according to the permissions set on that location. You can also restrict access based on the Spring web flow. See [Restricting Access by Role](#) for more information.

Customizing the JasperReports Wizard

Repository Trees

Having a large number of resources in your repository may slow down the user session when using the JasperReports Wizard. When creating or editing a report using the wizard, JasperReports Server searches the repository for the appropriate resource types before opening the repository tree, which displays only the folders containing the found resource types but can result in a long load time if you have a large repository.

You can speed this up by enabling "lazy loading" for specific resource type trees in the UI.

Lazy loading speeds up the load time of the repository tree because JasperReports Server will not search for the resources until you browse to a folder in the tree. The repository tree displays all of the folders in the repository but only displays the appropriate resources when you open a folder. Any folder that does not contain resources that can be used by the report will appear empty.

To enable lazy loading for the JasperReports Wizard's repository trees, you will need to add a new property to the appropriate beans in the repository tree configuration files. The following table contains the configuration files and bean IDs for the JasperReports Wizard's repository trees.

Repository Tree Configuration Files

Configuration File	Bean ID	Repository Tree
<js-webapp>/WEB-INF/flows/repositoryExplorerBean.xml	inputControlResourceTreeDataProvider	Input Controls (Controls & Resources page)
	jrxmlTreeDataProvider	JRXML Files (Set Up the Report page)
	queryTreeDataProvider	Query

Configuration File	Bean ID	Repository Tree
		(Locate Query page)
	fileResourceTreeDataProvider	File Resources (Controls & Resources page)
<js-webapp>/WEB-INF/flows/repositoryExplorerProBean.xml	proDsTreeDataProvider	Data Source (Link a Data Source to the Report page)

To enable "lazy loading" for your repository

1. Open the repository tree configuration file you want to edit.
2. Locate the bean for the repository tree for which you want to enable lazy loading.
3. Add the following line to the bean:

```
<property name="lazy" value="true" />
```

4. Save the modified file and restart the application server to see changes.

Customizing Menus

A very common customization is the removal, addition, or restriction of access to the main menu of JasperReports Server. Although you can remove access to the main menu items by simply hiding the menu elements with CSS in a theme, as described in [Hiding UI Elements](#), users can bypass this restriction. 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 users see different menu choices depending on their roles.

The mechanism that implements the main menu is called the `actionModel`, and it's 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` converts the XML into the JavaScript that generates menus.

Removing a Menu Item

In this example, suppose 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 UI.



Removing Domains from the menu hides Domains but does not disable them. Users can still access Domains by entering the URI of the Domain webflow. See [Working With Custom Java Classes](#) for information on how to restrict a webflow.

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. The XML that defines the `actionModel`, contains many condition tests that determine when each menu and menu item should be displayed.

```
...
<context name="main_create_mutton" test="isProVersion">
  <condition test="!banUserRole">
    <condition test="!isMainFeaturesDisabled">
      <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>
<!--
    <condition test="isAvailableProFeature" testArgs="AHD">
        <condition test="checkAuthenticationRoles" testArgs="ROLE_ADMINISTRATOR">
            <option labelKey="NAV_056_DOMAIN"
                action="primaryNavModule.navigationOption"
                actionArgs="domain"/>
        </condition>
    </condition>
-->
</selectAction>
</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 users right-click folders and resources in repository listings.

```

<?xml version="1.0" encoding="UTF-8"?>
<actions>
<context name="folder_mutton">
    <simpleAction labelKey="SEARCH_CREATE_FOLDER" action="invokeFolderAction"
        actionArgs="CreateFolder" clientTest="canCreateFolder"
        className="up"/>
    <condition test="checkAuthenticationRoles"
        testArgs="ROLE_USER,ROLE_ADMINISTRATOR">
        <selectAction labelKey="SEARCH_CREATE_RESOURCE"
            clientTest="canResourceBeCreated" className="flyout">
            <condition test="checkAuthenticationRoles" testArgs="ROLE_ADMINISTRATOR">
            <option labelKey="RM_NEW_RESOURCE_DATA_SOURCE" action="invokeCreate"
                actionArgs="ReportDataSource" clientTest="canResourceBeCreated"
                clientTestArgs="ReportDataSource" className="up"/>
            <option labelKey="RM_NEW_RESOURCE_DATATYPE" action="invokeCreate"
                actionArgs="DataType" clientTest="canResourceBeCreated"
                clientTestArgs="DataType" className="up"/>
        </condition>
    </condition>
    <!--
        <condition test="isProVersion">
            <option labelKey="RM_NEW_DOMAIN" action="invokeCreate"
                actionArgs="SemanticLayerDataSource"
                clientTest="canResourceBeCreated"
                clientTestArgs="SemanticLayerDataSource" className="up"/>
        </condition>
    -->

```

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 [Reloading the JasperReports Server Web App](#)).
6. When the web app has reloaded, log into JasperReports Server as jasperadmin. Click the Create menu and right-click 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. It's often preferable to prevent only non-administrators from viewing a menu item, as shown in the next section.

Restricting Access by Role

You can use role-based customizations to control access to many user interface components, including menus, Java Server Pages, and web flows. This example shows how to control access to existing UI components; the same techniques work with your custom components.

In this example, suppose end users haven't had training in creating reports with the Ad Hoc Editor, and you want to hide it from users, but make it accessible to administrators. To hide access to the Ad Hoc Editor, you need to customize the UI in three ways:

- Customize the Create menu to restrict access to Ad Hoc creation to administrators only
- Customize the JSP content on the home page to hide the Create Ad Hoc View button from non-administrative users.
- Customize the Ad Hoc web flow to restrict access to administrators only.

The following sections show how to perform each of these actions.



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 a problem, you may need to restart the app server; reloading the web app doesn't always resolve the issue.

Restricting a Menu Item by Role

1. Edit the file `<js-webapp>/WEB-INF/actionModel-navigation.xml`. The actionModel for Create > Ad Hoc View is near the end of the file.

```

...
<context name="main_create_mutton" test="isProVersion">
  <condition test="!banUserRole">
    <condition test="!isMainFeaturesDisabled">
      <selectAction labelKey="NAV_005_CREATE">
        <condition test="isAvailableProFeature" testArgs="AHD">
          <condition test="checkAuthenticationRoles" testArgs="ROLE_ADMINISTRATOR">
            <option labelKey="NAV_051_ADHOC_REPORT"
              action="primaryNavModule.navigationOption"
              actionArgs="designer"/>
          </condition>
        </condition>
      </selectAction>
    </condition>
  </condition>
</context>

```

- Following the pattern of conditions for other administrator-only functionality, insert the `condition` tag for checking role authentication around the `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. Use one of these methods:

- `ORG_ROLE|orgID` – Explicitly specify a role belonging to an organization.
- `SYSTEM_ROLE` – Explicitly specify a role defined at the root or system level, such as `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 `test` and `testArgs` attributes to the `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>
...

```

Restricting a Section of a JSP File by Role

You can use Spring Security's authorization tags to set up access control on JSP pages. See the example below for how this tag is used. As described in the Spring documentation, the `authz:authorize` tag supports the following attributes:

- `hasRole`: All the listed roles must be granted for the tag to output its body.

- `hasAnyRole`: Any of the listed roles must be granted for the tag to output its body.
- `!hasAnyRole`: None of the listed roles must be granted for the tag to output its body.

The attributes take roles as values. To list multiple roles in an attribute, separate the roles using a comma.

To use the authorization tags in a JSP file:

1. Make sure the line to import the Spring authz tag is near the beginning of the file. This line is necessary in any JSP file that implements access control:

```
<%@ taglib uri="http://www.springframework.org/security/tags"
prefix="authz"%>
```

```
...
<%@ taglib prefix="t" uri="http://tiles.apache.org/tags-tiles" %>
<%@ taglib prefix="authz" uri="http://www.springframework.org/security/tags"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
<%@ taglib uri="/spring" prefix="spring"%>
...

<authz:authorize access="hasRole('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>

...
```

2. Insert the `authz:authorize` tag for checking role authentication before the element you want to restrict, as shown in the code sample above.



In commercial editions, you must also use the `authz:authorize` tag and specify the role's organization ID when restricting access to roles defined in an organization. For example:

```
<authz:authorize access="hasRole('ORG_ROLE|OrgID')">...</authz:authorize>
```

You can use the `ORG_ROLE|OrgID` or `SYSTEM_ROLE` syntax, as described in [Restricting a Menu Item by Role](#).

See the [Spring Security Reference Documentation](#) for more information.

Controlling Access to Web Flows

JasperReports Server uses Spring Web Flow to define and control its UI flow. A Spring flow is a sequence of related pages for which you define states and transitions in relation to your own business logic. In addition to controlling users' access to items on the menus, you can control their access to functionality by setting permissions on web flows.



Be very careful when setting access to existing web flows. UI components that depend on a web flow may not work properly if access to the web flow has been modified.

In this example, suppose you want to ensure users cannot access the Ad Hoc Editor through its URI. To do this, restrict the Ad Hoc web flow to administrative users:

1. Navigate to the Ad Hoc Editor by clicking the Create Ad Hoc View button on the home page. You see the URI for the flow in the navigation bar:

`http://localhost:8080/jasperserver-pro/flow.html?_flowId=adhocFlow`

This tells you that the URI for the Ad Hoc Editor flow is `adhocFlow`.

2. Open the file `<js-webapp>/WEB-INF/applicationContext-security.xml` for editing.
3. Locate the `flowVoter` bean.

This bean sets the permissions for flows. It contains a number of flows set to `ROLE_ADMINISTRATOR`. Note that `adhocFlow` does not appear explicitly. However, there is an entry `*=ROLE_USER,ROLE_ADMINISTRATOR`. This setting determines access for all flows that are not specifically mentioned.

```
<bean id="flowVoter"
  class="com.jaspersoft.jasperserver.api.security.FlowRoleAccessVoter">
  <property name="flowAccessAttribute" value="FLOW_ACCESS"/>
  <property name="flowDefinitionSource">
    <value>
      repoAdminFlow=ROLE_ADMINISTRATOR
      ...
      searchFlow=ROLE_USER,ROLE_ADMINISTRATOR
      *=ROLE_USER,ROLE_ADMINISTRATOR
      <!--custom flow permissions -->
      adhocFlow=ROLE_ADMINISTRATOR
    </value>
  </property>
</bean>
```

4. Set access for the Ad Hoc flow by adding an entry to restrict access to `ROLE_ADMINISTRATOR` as shown in the code sample above.

Controlling Access to the Scheduler and Dashboards UI

Restricting the Scheduler or Dashboards UI to an administrator requires editing a different file.

To restrict Scheduler or Dashboards to administrative users:

1. Open the file `<js-webapp>/WEB-INF/applicationContext-security-web.xml`.



For commercial versions of JasperReports Server, you will also need to edit a second file, `<js-webapp>/WEB-INF/applicationContext-security-pro-web.xml`, to restrict the Dashboards UI. The `applicationContext-security-web.xml` file only controls dashboards for the Community Project version.

2. Locate the `filterInvocationInterceptor` bean.

The entries for the intercept-URL patterns `/scheduler/main.html` and `/dashboard/viewer.html` determine the access for Scheduler and Dashboards.

```
<bean id="filterInvocationInterceptor"
  class="org.springframework.security.web.access.intercept.FilterSecurityInterceptor">
  <property name="securityMetadataSource">
    <security:filter-security-metadata-source lowercase-comparisons="true"
      path-type="ant" request-matcher="ant">
      ...
      <!--scheduler-->
      <security:intercept-url pattern="/scheduler/main.html"access="ROLE_USER,ROLE_ADMINISTRATOR"
    />
      ...
      <!--dashboard-->
      <security:intercept-url pattern="/dashboard/viewer.html" access="ROLE_USER,ROLE_
ADMINISTRATOR" />
    </security:filter-security-metadata-source>
  </property>
</bean>
```

3. Set access for Scheduler and Dashboards by removing `ROLE_USER`.
4. For commercial versions of JasperReports Server, open the file `<js-webapp>/WEB-INF/applicationContext-security-pro-web.xml`.
5. Locate the entry for the intercept-URL pattern for `/dashboard/viewer.html`.
6. Set access for Dashboards by removing `ROLE_USER`.

Loading Your Changes

1. Save the modified files and reload the web app in the app server to see the changes (see [Reloading the JasperReports Server Web App](#)).
2. When the web app has reloaded, log into JasperReports Server as `joeuser`. You'll see that the button for creating a report is removed, and there's 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.

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.

This example adds 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.

```

<!--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"/>
    ...
    <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 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 is based on the contents of the commercial edition `pro_nav_messages.properties`.

```

NAV_001_HOME=Home
NAV_002_VIEW=View
NAV_003_MANAGE=Manage
NAV_004_LOGOUT=Log Out
NAV_005_CREATE=Create
...
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.
6. Create a working directory where you can edit and build JavaScript source code, as described in [Customizing JavaScript Source Code](#).
7. Edit your working copy of the file jasperserver-ui/ce/jrs-ui/src/actionModel/actionModel.primaryNavigation.js.

```

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"},
  ...
```

**Warning:**

Because of space limitations, the line is shown broken across two lines. In your application, make sure it's on a single line.

8. Add a URL and parameters to load the target page of the new accounts menu item. In this example, the action is to perform a search for all reports with the word account in their name or description.
9. Build the JavaScript source code in your working directory and copy the output back to JasperReports Server as described in [Customizing JavaScript Source Code](#).
10. Save the modified files and reload the web app in the app server to see the changes (see [Reloading the JasperReports Server Web App](#)).
11. When the web app has reloaded, log into JasperReports Server as joeuser. You'll see the View > MyCompany Accounts menu item was created and, when you select it, it performs a customized search. The following figure shows both the menu item and the search results on the same page.

The screenshot shows the JasperReports Server web application interface. At the top, there is a navigation bar with the JasperSoft logo, a home icon, and menu items: Library, View, Manage, Create, superuser, Help, and Log Out. A search bar is on the right. Below the navigation bar, the main content area is titled 'Repository'. On the left, there are filters for 'Access Status' (All available) and 'File Type' (All types, Reports, Content resources, Ad Hoc views, Dashboards, OLAP views, Domains). A search dropdown menu is open, showing 'Search Results', 'Repository', 'Schedules', and 'Messages'. The 'MyCompany Accounts' menu item is highlighted. The main table displays search results with columns: Name, Type, Created, and Modified. The table contains several rows of reports, including '01. Geographic', '02. Sales Mix by City', '03. Store Segment Performance', '04. Product Results by Store Type', and '05. Accounts Report'.

Name	Type	Created ...	Modifie...
01. Geographic	Ad Hoc View	Yesterday	Yesterday
01. Geographic	Report	Yesterday	10/17/2...
02. Sales Mix by City	Ad Hoc View	Yesterday	Yesterday
02. Sales Mix by City Report	Report	Yesterday	8/28/2019
03. Store Segment Performance	Ad Hoc View	Yesterday	10/11/2...
03. Store Segment Performance ...	Report	Yesterday	10/11/2...
04. Product Results by Store Type	Ad Hoc View	Yesterday	Yesterday
04. Product Results by Store Typ...	Report	Yesterday	8/28/2019
05. Accounts Report	Report	Yesterday	8/28/2019
05. Unit Sales Trend	Ad Hoc View	Yesterday	10/11/2...

Figure 16: Custom Menu Items and Corresponding Action

Adding a New Main Menu

The following example shows two useful ways to add custom menu items:

- You can add your own menus to the main menu bar.
- You can add menu items to either existing or custom menus and link them outside the server.

This example creates a new menu named Accounts. It contains the internal search item 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.

```
...
<!--MyCompany custom menu-->
<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>
</actions>
```

Each menu item has 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, which you'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.

```
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. To edit the JavaScript source code, create a working directory where you can edit and build source code, as described in [Customizing JavaScript Source Code](#).
6. Edit your working copy of `jasperserver-ui/ce/jrs-ui/src/actionModel/actionModel.primaryNavigation.js`. Add the accounts navigation path if you didn't do so in the previous procedure:

```

...
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.



Because of space limitations, the line is shown broken across two lines. In your application, make sure it is on a single line.

7. In the same copy of the `jasperserver-ui/ce/jrs-ui/src/actionModel/actionModel.primaryNavigation.js` file, add the external search handler function to the end of the file, outside of all other definitions:

```

var externalSearchHandler = function(qstring) {
  window.location.href = "http://google.com/search?q=" + qstring;
};

window.primaryNavModule = primaryNavModule;
window.externalSearchHandler = externalSearchHandler;

export default primaryNavModule;
export {externalSearchHandler};

```



You are required to export the additional function to enable the custom menu.

8. Build the JavaScript source code and copy the output back to JasperReports Server as described in [Customizing JavaScript Source Code](#).
9. Reload the web app in the app server to see the changes (see [Reloading the JasperReports Server Web App](#)).
10. When the web app has reloaded, log into JasperReports Server as `joeuser`. You'll see the new Accounts menu with Accounts > Google MyCompany opening a Google

search. The following figure shows both the new menu and the external search results.

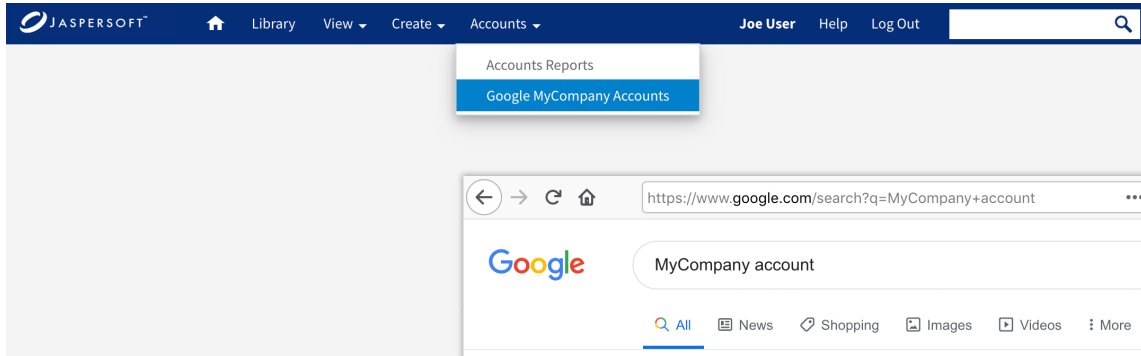


Figure 17: Creating a Custom Menu and an External Link



The menu shown 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 be closer to the metalinks; a second menu might overlap them. 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 metalinks or search field. In this case you could use themes to avoid overlap with the search field.

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`.

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`

Menus for the Dashboard Designer are not customizable in 6.x.

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 uses JDOM (Java-based Document Object Model for XML) to build and cache a Document.
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 in the action model definition files. In particular, you can define various conditions at several levels so menus appear only to certain users or based on the current state of the page.

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're directly equivalent to the menu levels defined in the popup-style JSP files used in previous releases.

Condition

A condition element invokes the specified server side test as a method on the view model. The enclosed actions are included in the client action model only if the test returns true.

If the test has a leading exclamation point (!), the condition tests for false:

- `test` – 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 comma-separated string.

Actions

Each defined action produces one or more rows in the generated menu. The following tables describe the several action types.

<code>simpleAction</code>	
A standalone menu action that, when clicked, fires the specified JavaScript method.	
Attributes	
<code>id</code>	The ID of the menu row DOM object.
<code>disabled</code>	If set to true, disable this row initially (shows a gray block instead).
<code>labelKey</code>	The text the menu should display. If it corresponds to an localization bundle key it is translated, otherwise it is displayed as is.
<code>labelCondition</code>	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 <code>labelCondition</code> defines two <code>labelOption</code> sub elements (see below).
<code>clientTest</code>	Only generate a row for this action if it passes the specified JavaScript method. If the <code>clientTest</code> has a leading exclamation point (!), tests for false.

simpleAction	
clientTestArgs	An array of parameters pass to the above test, expressed as a comma delimited string.
action	The JavaScript method to be fired when the action is taken.
actionArgs	An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant.
leaveMenuOpen	If true the menu stays displayed after action.
separator	
Not strictly an action. Outputs a separator bar. The style of the bar automatically adjusts to the current nesting level.	
Attributes	
attributesclientTest	Only generate a row for this action if it passes the specified JavaScript method.
clientTestArgs	An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant.
disabled	Disable this row initially (shows a gray block instead).
selectAction	
A drop down (roll down) parent. Selectors can be nested up to three levels deep. The menu automatically renders the roll-downs in a nested fashion.	
Attributes	
attributesdisabled	If set to true, disable this row initially (shows a gray block instead).
opened	If set to true, selector is opened initially.

selectAction	
labelKey	The text for the menu to display. If it corresponds to an localization bundle key it is translated, otherwise it is displayed as is.
labelCondition	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).
clientTest	Only generate a row for this action if it passes the specified JavaScript method.
clientTestArgs	An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant.

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 programmatically.	
Attributes	
id	The ID of the menu row DOM object.
disabled	If set to true, disable this row initially (shows a gray block instead).
button	If set to true the option displayed as a button (as in formula builder).
labelKey	The text for the menu to display. If it corresponds to an localization bundle key it is translated, otherwise it is displayed as is.

option	
labelCondition	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).
clientTest	Only generate a row for this action if it passes the specified JavaScript method.
clientTestArgs	An array of parameters to be passed to the above test, expressed as a comma delimited string.
allowsInputTest	Show an input box in this option if it passes the specified JavaScript method.
allowsInputTestArgs	An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant.
action	The JavaScript method to be fired when the action is taken.
actionArgs	An array of parameters to be passed to the above test, expressed as a comma delimited string.
isSelectedTest	Indicate a check mark next to this option if it passes the specified JavaScript method.
isSelectedTestArgs	An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant.
generatedOptions	
Child of selectAction. Defines a set of dynamically-defined options.	
Reserved Variables (in addition to standard reserved variables)	
#{optionId}	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.

generatedOptions	
<code>\${optionValue}</code>	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.
<code>\$R{<String>}</code>	When used in a label expression attempts to internationalize the enclosed String and if it fails it returns the literal value.
Attributes	
<code>id</code>	The ID of the menu row DOM object.
<code>function</code>	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).
<code>functionArgs</code>	An array of parameters to pass to the above test, expressed as a string delimited by the @@ string constant.
<code>labelKey</code>	The text for the menu to display. If it corresponds to an localization bundle key it is translated, otherwise it is displayed as is.
<code>labelCondition</code>	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).
<code>labelExpression</code>	Allows a custom label to be defined and allows full use of all reserved variables. (for example, <code>labelExpression="\${optionValue}" \$R{ADH_252_DATA_ROWS}</code>).
<code>clientTest</code>	Only generate a row for this action if it passes the specified JavaScript method.
<code>clientTestArgs</code>	An array of parameters to be passed to the above test, expressed

generatedOptions	
	as a string delimited by the @@ string constant. If no actionArgs are specified, the <code>{optionId}</code> variable is automatically assigned as an argument.
<code>allowsInputTest</code>	Show an input box in this option if it passes the specified JavaScript method.
<code>allowsInputTestArgs</code>	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 <code>{optionId}</code> variable is automatically assigned as an argument.
<code>action</code>	The JavaScript method to be fired when the action is taken.
<code>actionArgs</code>	An array of parameters to be passed to the above test, expressed as a comma delimited string. If no actionArgs are specified, the <code>{optionId}</code> variable is automatically be assigned as an argument.
<code>leaveMenuOpen</code>	If true the menu stays displayed after action (for all generated options).
<code>isSelectedTest</code>	Indicate a check mark next to this option if it passes the specified JavaScript method.
<code>isSelectedTestArgs</code>	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 <code>{optionId}</code> variable is automatically assigned as an argument.
Default Settings (DOM attributes set automatically on each generated Option)	
<code>id</code>	If not specified as an a attribute defaults to the <code>{optionId}</code> variable.
<code>clientTestArgs</code>	If not specified as an a attribute defaults to the <code>{optionId}</code> variable.

generatedOptions

actionArgs	If not specified as an attribute defaults to the <code>\${optionId}</code> variable.
------------	--

isSelectedArgs	If not specified as an attribute defaults to the <code>\${optionId}</code> 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

attributesfunction Response	A Boolean string. If the parent <code>labelCondition</code> result matches this Boolean value, this <code>labelOption</code> 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 <code>isgenerateFromTemplate</code> .
----------	--

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)

<code>\${templateInjection Index}</code>	The zero-based index of the current iteration.
--	--

<code>\${templateInjection Id}</code>	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 <code>toString</code> value of each element.
---	--

<code>\${templateInjection</code>	Programmatically assigned value. If function returns a Map this is
-----------------------------------	--

generateFromTemplate	
Value}	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 <code>#{templateInjectionId}</code> and <code>#{templateInjectionValue}</code> are interpreted (see above).
functionArgs	An array of parameters to be passed to the above test, expressed as a string delimited by the @@ string constant.
Child Elements	
Any valid action structure, which is used as the template.	

Special Expressions

The following special expressions are converted to built-in variable values when used as arguments for client functions:

Expression	Definition
<code>#{selected}</code>	The JavaScript array of selected objects on the client.
<code>#{event}</code>	The JavaScript event object.
<code>#{label}</code>	The generated label for this menu.

Menu DHTML API

This is a set of JavaScript functions defined on `actionModel.js` to dynamically manipulate a menu's look and feel after it's 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.

Function	Description
<code>hideInputForOption</code>	Hide the input box on the option.
<code>showInputForOption</code>	Show the input box on the option.
<code>setRowColor</code>	Set the <code>backgroundColor</code> to the specified color.
<code>resetRowColor</code>	Restore original background color.
<code>getLabel</code>	Return the current label for this row.
<code>setLabel</code>	Set the label for this row.
<code>disableRow</code>	Grey out the row.
<code>enableRow</code>	Restore the row to active state.

Customizing the Report Rendering Page

The report viewer is the UI module 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 provides the following functionality:

- Display Input controls so users can set report options, also called parameters.
- Enable navigation through the pages of the report.
- Enable users to export the report in various formats.

Each of these is customizable, which allows you to control many details of the report viewing experience. The Input controls can be set either globally or on individual reports. The report viewer can be set globally.

Customizing Input Controls

A report can have many Input controls. You can group them by some category and display them on different tabs in the Input controls dialog. This improves the user experience.

In the following example, Input controls are grouped by location and time.

To customize the Input controls

1. Create a report with Input controls named Country, Zip, Year, and Month.
2. Create a `customParametersForm.jsp` file and place it in the `jasperserver-pro/WEB-INF/jsp/modules/inputControls/` folder with the following content:

```
<%@ taglib prefix="t" uri="http://tiles.apache.org/tags-tiles" %>

<jsp:include page="InputControlTemplates.jsp" />
<script type="text/javascript" src="${pageContext.request.contextPath}/scripts/underscore-umd-min.js"></script>

<!-- styles we want to apply to new tabs on Input Controls dialog -->

<style type="text/css">

    .tabsButtons {
        margin-bottom: 30px;
    }

    #nicPanel {
        min-width:500px;
        min-height:400px;
    }

    .nicControlsPanel {
        border: 1px solid #e0e0e0;
        padding: 20px;
    }

    #nicCountrySearch {
        width: 300px;
    }

    #spanCountry {
        position: relative;
        top: -7px;
    }

    #inputControls {
        min-width: 550px;
    }
</style>
```

```

        min-height: 500px;
    }
</style>

<!-- adding tabs element from template -->
<div class="tabs tabsButtons">
    <t:insertTemplate template="/WEB-INF/jsp/templates/control_tabSet.jsp">
        <t:putAttribute name="type" value="buttons"/>
        <t:putAttribute name="containerClass" value="horizontal"/>
        <t:putListAttribute name="tabset">
            <!-- tab that will contain input controls related to Shipping Location -->
            <t:addListAttribute>
                <t:addAttribute>nicByShippingLocationTab</t:addAttribute>
                <t:addAttribute>By Shipping Location</t:addAttribute>
                <t:addAttribute>selected</t:addAttribute>
            </t:addListAttribute>
            <!-- tab that will contain input controls related to Shipping Date -->
            <t:addListAttribute>
                <t:addAttribute>nicByShippingDateTab</t:addAttribute>
                <t:addAttribute>By Shipping Date</t:addAttribute>
            </t:addListAttribute>
        </t:putListAttribute>
    </t:insertTemplate>
</div>
<div id="nicByShippingLocationPanel" class="nicControlsPanel">
</div>
<div id="nicByShippingDatePanel" class="nicControlsPanel"></div>

<script type="text/javascript">
    document.addEventListener("controls:initialized", function(event) {
        const controlsViewModel = event.detail;
        controlsViewModel.draw = function (jsonStructure) {

            //get and initialize Input Controls object
            const drawControl = function (container, jsonControl) {
                if (jsonControl.visible) {
                    const control = this.findControl({id: jsonControl.id});
                    control.render();
                    document.getElementById(container).append(control.getElem() && control.getElem()
[0]);
                }
            };

            //selecting only input controls named Country or Zip
            const leftPanelControls = _.filter(jsonStructure, function (controlStructure) {
                return _.indexOf(["Country", "Zip"], controlStructure.id) >= 0;
            });

            //assigning Country and Zip ICs to the left panel
            _.each(leftPanelControls, _.bind(drawControl, this, "nicByShippingLocationPanel"));

            //selecting only input controls named Year or Month
            const rightPanelControls = _.filter(jsonStructure, function (controlStructure) {
                return _.indexOf(["Month", "Year"], controlStructure.id) >= 0;
            });

            //assigning Year and Month ICs to the right panel
            _.each(rightPanelControls, _.bind(drawControl, this, "nicByShippingDatePanel"));
        };
    });

```

```

l").hidden = false;
                                document.querySelector("div.tabs.tabsButtons > ul > li.tab.last").
                                classList.add("selected");
                                document.querySelector("div.tabs.tabsButtons > ul > li.tab.first").
                                classList.remove("selected");
                                }
                                };

                                nic.clickHandlerMap = {
                                'nicByShippingLocationTab':nic.tabByShippingLocationClickHandler,
                                'nicByShippingDateTab':nic.tabByShippingDateClickHandler
                                };

                                // assign handlers
                                _.each(nic.clickHandlerMap, function (handler, selector) {
                                const shippingTabElement = document.getElementById(selector);
                                shippingTabElement.addEventListener('click', handler, false);
                                });
                                });
                                </script>

```

3. Download the underscore-umd-min.js script from <https://underscorejs.org/underscore-umd-min.js> and place it into /webapps/jasperserverpro/scripts/.
4. Restart JasperReports Server.
5. Go to View > Repository and click Edit.
6. Select Controls & Resources from the menu and then enter modules/inputControls/customParametersForm.jsp in the Optional JSP Location field.
7. Save and run the report.

You can see that the Input Controls dialog has two tabs "By Shipping Location" and "By Shipping Date". Each of these tabs contains the corresponding Input controls.

The screenshot displays the JasperReports Server interface. The main window shows a table titled "Shipping Report" with columns: Order ID, Order Date, Shipped Date, Ship Country, and Postal Code. The table contains 25 rows of data. On the right, the "Input Controls" panel is open, showing the "By Shipping Location" tab. It features a search box for "Country" with 21 available and 1 selected. Below the search box is a list of countries: Argentina, Austria, Belgium, Brazil, Canada, Denmark, Finland, France, Germany, and Ireland. There are also "Select All", "Deselect All", and "Invert" buttons. Below the country list is a search box for "Zip" with 4 available and 0 selected. Below the search box is a list of zip codes: 05021, 05022, 05023, and 05033. There are also "Select All", "Deselect All", and "Invert" buttons. At the bottom of the panel are "Apply", "OK", "Reset", "Cancel", and "Save" buttons.

Order ID	Order Date	Shipped Date	Ship Country	Postal Code
10298	18-Jul-17	25-Jul-17	Mexico	05022
10276	08-Aug-17	14-Aug-17	Mexico	05033
10293	29-Aug-17	11-Sep-17	Mexico	05033
10304	12-Sep-17	17-Sep-17	Mexico	05023
10308	18-Sep-17	24-Sep-17	Mexico	05021
10319	02-Oct-17	11-Oct-17	Mexico	05033
10322	04-Oct-17	23-Oct-17	Mexico	05033
10354	14-Nov-17	20-Nov-17	Mexico	05023
10365	27-Nov-17	02-Dec-17	Mexico	05023
10474	13-Mar-18	21-Mar-18	Mexico	05033
10502	10-Apr-18	29-Apr-18	Mexico	05023
10507	15-Apr-18	22-Apr-18	Mexico	05022
10518	25-Apr-18	05-May-18	Mexico	05033
10535	13-May-18	21-May-18	Mexico	05023
10573	19-Jun-18	20-Jun-18	Mexico	05023
10576	23-Jun-18	30-Jun-18	Mexico	05033
10625	08-Aug-18	14-Aug-18	Mexico	05021
10676	22-Sep-18	28-Sep-18	Mexico	05033
10877	22-Sep-18	28-Sep-18	Mexico	05023
10882	25-Sep-18	01-Oct-18	Mexico	05023
10759	28-Nov-18	12-Dec-18	Mexico	05021
10842	20-Jan-19	29-Jan-19	Mexico	05033
10856	28-Jan-19	18-Feb-19	Mexico	05023
10915	27-Feb-19	02-Mar-19	Mexico	05033
10926	04-Mar-19	11-Mar-19	Mexico	05021
10995	02-Apr-19	08-Apr-19	Mexico	05023
11089	04-May-19	06-May-19	Mexico	05033
11073	05-May-19	06-May-19	Mexico	05033

Figure 18: By Shipping Location Tab

The screenshot displays the JasperReports Server interface. The main window shows a table titled "Shipping Report" with columns: Order ID, Order Date, Shipped Date, Ship Country, and Postal Code. The table contains 25 rows of data. On the right, the "Input Controls" panel is open, showing the "By Shipping Date" tab. It features a search box for "Year" with 3 available and 0 selected. Below the search box is a list of years: 2017, 2018, and 2019. There are also "Select All", "Deselect All", and "Invert" buttons. Below the year list is a search box for "Month" with 12 available and 0 selected. Below the search box is a list of months: January, February, March, April, May, June, July, August, September, and October. There are also "Select All", "Deselect All", and "Invert" buttons. At the bottom of the panel are "Apply", "OK", "Reset", "Cancel", and "Save" buttons.

Order ID	Order Date	Shipped Date	Ship Country	Postal Code
10259	18-Jul-17	25-Jul-17	Mexico	05022
10276	08-Aug-17	14-Aug-17	Mexico	05033
10293	29-Aug-17	11-Sep-17	Mexico	05033
10304	12-Sep-17	17-Sep-17	Mexico	05033
10308	18-Sep-17	24-Sep-17	Mexico	05021
10319	02-Oct-17	11-Oct-17	Mexico	05033
10322	04-Oct-17	23-Oct-17	Mexico	05033
10354	14-Nov-17	20-Nov-17	Mexico	05033
10365	27-Nov-17	02-Dec-17	Mexico	05023
10474	13-Mar-18	21-Mar-18	Mexico	05033
10502	10-Apr-18	29-Apr-18	Mexico	05033
10507	15-Apr-18	22-Apr-18	Mexico	05023
10518	25-Apr-18	05-May-18	Mexico	05033
10535	13-May-18	21-May-18	Mexico	05023
10573	19-Jun-18	20-Jun-18	Mexico	05023
10576	23-Jun-18	30-Jun-18	Mexico	05023
10625	08-Aug-18	14-Aug-18	Mexico	05021
10676	22-Sep-18	28-Sep-18	Mexico	05033
10877	22-Sep-18	28-Sep-18	Mexico	05023
10882	25-Sep-18	01-Oct-18	Mexico	05023
10759	28-Nov-18	12-Dec-18	Mexico	05021
10842	20-Jan-19	29-Jan-19	Mexico	05033
10856	28-Jan-19	18-Feb-19	Mexico	05023
10915	27-Feb-19	02-Mar-19	Mexico	05033
10926	04-Mar-19	11-Mar-19	Mexico	05021
10995	02-Apr-19	08-Apr-19	Mexico	05033
11089	04-May-19	06-May-19	Mexico	05033
11073	05-May-19	06-May-19	Mexico	05033

Figure 19: By Shipping Date Tab

Customizing the Report Viewer

The report viewer creates the page with the controls for paging through a report, and then 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
```

To change the report viewer, modify and save the default files without changing their names. The procedure would be similar to [Customizing JSP and JavaScript for the Login Page](#). After redeploying the web app, your changes to the report view appear in every report for every user. The following example shows how to modify the ViewReport.jsp file so that the Save and Save As options are only visible to administrators.

To modify the Save and Save As buttons in the report viewer

1. Add the line to import the Spring authz tag near the beginning of the file. This line is necessary to implement access control in a JSP file:

```
<%@ taglib uri="http://www.springframework.org/security/tags"
prefix="authz"%>
```

```
...
<%@ taglib prefix="t" uri="http://tiles.apache.org/tags-tiles" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="js" uri="/WEB-INF/jasperserver.tld" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ taglib prefix="authz" uri="http://www.springframework.org/security/tags"%>
...
```

2. Insert authz:authorize tags around the elements that implement the Save and Save As options.

```
<!-- changes for Ult Guide to remove save buttons -->
<authz:authorize access="hasRole('ROLE_ADMINISTRATOR')">
  <c:if test="{isPro}">
    <li class="leaf"><button id="fileOptions" class="button capsule mutton up first"
      title="<spring:message code="button.save"/>" disabled="true">
      <span class="wrap"><span class="icon"></span>
      <span class="indicator"></span></span></button></li>
  </c:if>
  <c:if test="{!isPro}">
    <li class="leaf"><button id="fileOptions" class="button capsule mutton up first"
      title="<spring:message code="button.save"/>
      - <spring:message code="feature.pro.only"/>" disabled="true">
      <span class="wrap"><span class="icon"></span>
      <span class="indicator"></span></span></button></li>
  </c:if>
</authz:authorize>
```

The following figure shows that the Save icon at the top left, under the report title, still appears for an administrator:

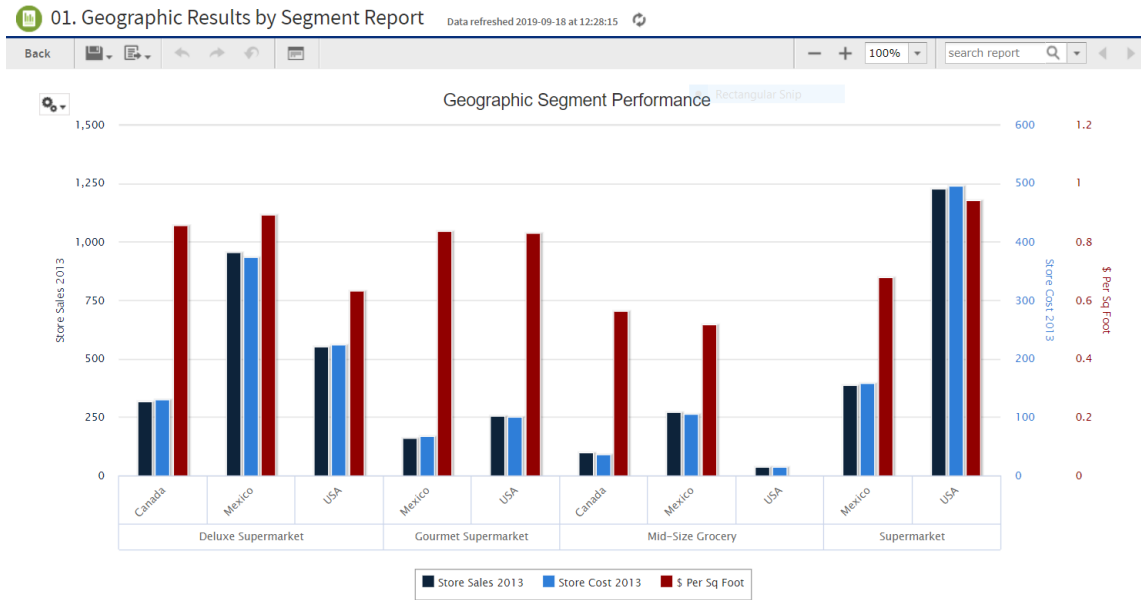


Figure 20: Administrator's View of Modified Report Viewer

The following figure shows the view for an end user, with the Save icon removed.

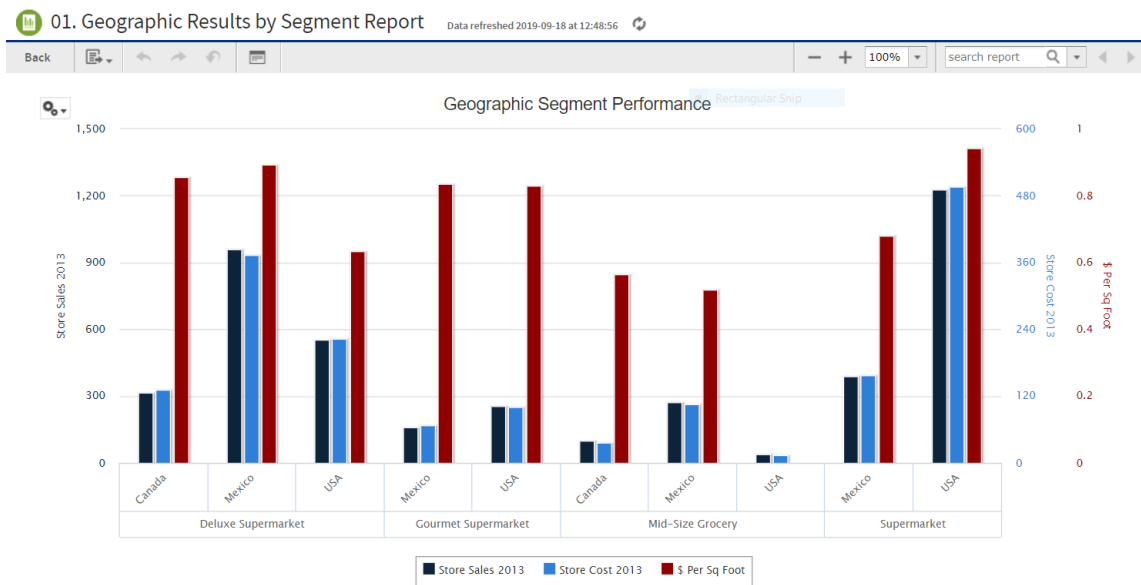


Figure 21: User's View of Modified Report Viewer

Working With Custom Java Classes

Some customizations 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 to deploy your custom features. When you redeploy the web application the files running in the application server are replaced with those you just compiled. For this reason, 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 build the source code and redeploy the web application.

To build and deploy the source code, follow the instructions in the *JasperReports Server Source Build Guide* within each distribution:

Commercial Editions: <https://www.jaspersoft.com/support>
Community Project: <https://community.jaspersoft.com>

In most cases, once you've successfully built and deployed the source code once, you don't need to edit the properties files or JVM settings when you rebuild after adding new customizations. You also don't need to create or load the databases.

The following procedure is 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're 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.

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, you 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. This example shows how to add a single page, but you could integrate a series of pages and the navigation among them.

To further integrate your pages with the server, you 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 business logic to a web flow involves creating Java methods to implement the logic. In general, the server UI contains most of the functionality in action class code that can be associated with one or more JSP pages. 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 easier to organize. In this example, the Java method simply returns success whenever it's invoked.

For more information, refer to the [Spring documentation](#) for flows and MVC.

This example requires you to work with the JasperReports Server source code, as explained in [Working With Custom Java Classes](#).

This example is divided into several tasks:

1. Adding a custom JSP file integrated into the server as a web flow.
2. Creating an action and adding it to the web flow.
3. Adding the web flow to a menu.

Example of adding a custom JSP file integrated into the server as a web flow

1. Create a subdirectory named `sampleFlow` for the JSP files in your flow module in `<js-src>/jasperserver-pro/jasperserver-war/src/main/webapp/WEB-INF/jsp/`. For example:

```
cd <js-src>/jasperserver-pro/jasperserver-war/src/main/webapp/WEB-INF/jsp/
mkdir sampleFlow
```

2. Create the following JSP file and save it as `sampleView.jsp` in `<js-src>/jasperserver-pro/jasperserver-war/src/main/webapp/WEB-INF/jsp/sampleFlow`:

```
<html>
<head><title>Sample Page</title>
</head>
<body class="oneColumn primary column">
<h1 class="textAccent">Hello World!</h1>
</body>
</html>
```

Note:

Design the layout of your custom content based on the UI components in the server. The UI components are defined in the CSS files in the `<js-src>/jasperserver/jasperserver-war/src/main/webapp/themes/default/` directory.

This example uses a one-column layout and the `textAccent` font.

3. Create the following XML file with a `flow` container element and an empty `view-state` element. In a later step, you will add an action state:

```
<?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="sampleView">

    <view-state id="sampleView"
      view="modules/sampleFlow/sampleView">
    </view-state>

    <end-state id="done"/>

</flow>

```

4. Name the file docSampleFlow.xml and save it in the <js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/flows directory.

Example of setting flow permissions

5. Set permissions for your flow:
 - a. Edit the file <js-src>/jasperserver/common/shared-config/applicationContext-security.xml.
 - b. Locate the flowVoter bean. This bean sets the permissions for flows.
 - c. Add a line for your flow and set the permissions to ROLE_ADMINISTRATOR.

```

<bean id="flowVoter"

class="com.jaspersoft.jasperserver.api.security.FlowRoleAccess
Voter">
  <property name="flowAccessAttribute" value="FLOW_ACCESS"/>
  <property name="flowDefinitionSource">
    <value>
      repoAdminFlow=ROLE_ADMINISTRATOR
      ...
      docSampleFlow=ROLE_ADMINISTRATOR
      <!--objectPermissionToUserFlow=ROLE_ADMINISTRATOR-->
      searchFlow=ROLE_USER,ROLE_ADMINISTRATOR
      *=ROLE_USER,ROLE_ADMINISTRATOR
    </value>
  </property>
</bean>

```

**Note:**

The final entry in the `flowVoter` bean, `*=ROLE_USER,ROLE_ADMINISTRATOR`, sets the default permissions for all flows not specified directly. If you don't create an entry for your flow, these permissions apply.

Example of creating an action and adding it to the flow

6. Create a java class that defines the controller in the Spring MVC framework. In this example, this file always returns success when invoked.
 - a. Go to the `<js-src>/jasperserver/jasperserver-war-jar/src/main/java/com/jaspersoft/ji/war/` directory. This is where the JasperReports Server source looks for java files used by the Spring web flow framework.
 - b. Create a subdirectory for your flow package, `<js-src>/jasperserver/jasperserver-war-jar/src/main/java/com/jaspersoft/ji/war/sampleFlow/`
 - c. Create a java file and save it as `<js-src>/jasperserver/jasperserver-war-jar/src/main/java/com/jaspersoft/ji/war/sampleFlow/SampleAction.java`

```
package com.jaspersoft.ji.war.sampleFlow;

import org.springframework.webflow.action.MultiAction;
import org.springframework.webflow.execution.Event;
import org.springframework.webflow.execution.RequestContext;

public class SampleAction extends MultiAction {
    public Event start(RequestContext context) throws
    Exception{

        // implement some logic

        return success();
    }
}
```

7. Create a bean for the `SampleAction` class that you created. To this, create the following file and save it as `<js-src>/jasperserver/jasperserver-war/src/main/webapp/WEB-INF/flows/docSampleBeans.xml`:

```

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"

       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-
       3.1.xsd
       http://www.springframework.org/schema/util
       http://www.springframework.org/schema/util/spring-util-
       3.1.xsd">
  <bean id="sampleAction"
class="com.jaspersoft.ji.war.sampleFlow.SampleAction">
  </bean>
</beans>

```

8. Modify docSampleFlow.xml to start with an action state that calls the SampleAction class you created:
 - a. Change the start-state to start.
 - b. Create an action state start that calls sampleAction and transitions to the view-state sampleView on success. Insert this state before sampleView.
 - c. At the end of the flow, import docSampleBeans.xml as a resource.

```

<?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/webf
      low
      http://www.springframework.org/schema/webflow/spring-webflow-
      2.0.xsd"

      start-state="start">

  <action-state id="start">
    <evaluate expression="sampleAction"/>
    <transition on="success" to="sampleView"/>
  </action-state>

  <view-state id="sampleView"
view="modules/sampleFlow/sampleView">
  </view-state>

  <view-state id="errorPage" view="modules/system/errorPage"/>

```

```

<end-state id="done"/>

<global-transitions>
  <transition on="backFromErrorPage" to="backFromError"/>
  <transition on-exception="java.lang.Throwable"
to="errorPage"/>
</global-transitions>

<!-- end exceptions handling -->

<bean-import resource="docSampleBeans.xml"/>

</flow>

```

9. Add error handling as shown in the code sample above:
 - a. Add a view-state `errorPage` to your flow. In this example, you add it immediately after `sampleView`.
 - b. Add a `global-transitions` state that handles Java exceptions by displaying `errorPage`.
10. (Optional) If you want, you can rebuild the source code and view your page:
 - Rebuild the source code and redeploy the web application according to the instructions in the Building JasperReports Server section in the *JasperReports Server Source Build Guide* within your distribution. See [Working With Custom Java Classes](#) for an overview of this process.
 - Log in to your JasperReports Server.
 - Navigate to the page you created using this URL:
`http://<hostname>:<port>/jasperserver-pro/flow.html?_flowId=docSampleFlow`

Example:

```

http://localhost:8080/jasperserver-pro/flow.html?_
flowId=docSampleFlow

```

Example of creating a menu item to call your flow

Now you can add a menu item to call the flow you created. The process is similar to the one described in [Adding an Item to the Main Menu](#), but because you're modifying the source, the file locations are different in this example.

**Note:**

Because the commercial source code includes the community source, most modifications to the menu are made in the community source files.

11. Edit the file `<js-src>/jasperserver/common/shared-config/actionModel-navigation.xml`. Locate the `actionModel` for the View menu in the file.
12. Add an `option` tag for the menu item, as shown in the following code sample. The `option` tag has attributes to specify the label key and the name of the action to perform.

```
<!--context for view option on primary menu-->
<context name="main_view_mutton" test="!banUserRole">
  <selectAction labelKey="menu.repository">
    ...
    <condition test="checkAuthenticationRoles" testArgs="ROLE_
ADMINISTRATOR">
      <separator/>
      <option labelKey="menu.samples" clientTest="!isIPad"
        action="primaryNavModule.navigationOption"
        actionArgs="samples"/>
    </condition>

    <separator/>
    <condition test="checkAuthenticationRoles" testArgs="ROLE_
ADMINISTRATOR">
      <option labelKey="NAV_028_DOC_SAMPLE"
        action="primaryNavModule.navigationOption"
        actionArgs="docSample"/>
    </condition>
  </selectAction>
</context>
```

13. Add a `condition` tag for the menu item, as shown in the code sample above.

**Note:**

The `condition` tag prevents the menu item from being displayed to users without the specified permissions. However, to ensure the flow can't be accessed via URL, set flow permissions as shown in [step 5](#).

14. 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:

```
NAV_001_HOME=Home
...
NAV_027_SEARCH=Search Results
NAV_028_DOC_SAMPLE>Hello World
NAV_031_USERS=Users
...
```

15. Create a working directory where you can edit and build JavaScript source code, as described in [Customizing JavaScript Source Code](#).
16. In a text editor, open your working copy of the file `<js-src>/jasperserver/jasperserver-war/src/main/webapp/scripts/actionModel.primaryNavigation.js`, and locate the `navigationPaths` section. Add a line that specifies the accounts flow to begin when the menu item is selected. If you are adding your line at the end of the section, make sure to add a comma to the previous line.

```
var primaryNavModule = {
  ...
  navigationPaths : {
    ...
    logSettings : {url : "log_settings.html"},
    docSample : {url : "flow.html", params : "_
flowId=docSampleFlow"}
  },
```

17. Create a file "sampleFlowMain.js" at "jasperserver-ui/ce/jrs-ui/src/sampleFlow/sampleFlowMain.js", as shown in the following code sample:

```
import '../util/webpackPublicPathSetup';
const importStartup = import('../util/mainPagesStartup')
const importCommonModule = () => import('../commons/commons.main')

importStartup.then(({default: startup}) => startup({
  importCommonModule
}))
```

18. Add the following file entry to "jasperserver-ui/ce/jrs-ui/webpack.config.js"

```
'system/systemErrorMain': './src/system/systemErrorMain',
'system/errorMain': './src/system/errorMain',
```

```
*'sampleFlow/sampleFlowMain': './src/sampleFlow/sampleFlowMain'*
```

19. (For JasperReports Server Professional edition only) Create a file "sampleFlowMain.js" at "jasperserver-ui/pro/jrs-ui-pro/src/sampleFlow/sampleFlowMain.js", as shown in the following code sample:

```
import 'jrs-ui/src/util/webpackPublicPathSetup';
const importStartup = import('jrs-ui/src/util/mainPagesStartup')
const importCommonModule = () => import('../commons/commons.main')

importStartup.then(({default: startup}) => startup({
  importCommonModule
}))
```

Then update the JSP file mentioned in step 2 to:

```
<html>
<head><title>Sample Page</title>
<script type="text/javascript"
src="${pageContext.request.contextPath}/runtime/${jsOptimizationPro
perties.runtimeHash}/JavaScriptServlet"></script>
  <script
src="${pageContext.request.contextPath}/runtime/${jsOptimizationPro
perties.runtimeHash}/scripts/sampleFlow/sampleFlowMain.js"
defer></script>

</head>
<body class="oneColumn primary column">
<h1 class="textAccent">Hello World!</h1>
</body>
</html>
```

Compile code and view changes

20. Build the JavaScript source code in your working directory and copy the output back to JasperReports Server as described in [Customizing JavaScript Source Code](#).
21. Rebuild the source code and redeploy the web application according to the instructions in the *JasperReports Server Source Build Guide* within your distribution. See [Working With Custom Java Classes](#) for an overview of this process.
22. Log in as an administrator. 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 sampleView.jsp file:

- The page uses the single column layout and the orange text is the `textAccent` class in CSS.

Customizing the Scheduler Pages

The Output Options pages of the report scheduler and dashboard scheduler display the output types available for a scheduled report or dashboard. You can customize these options in several ways:

- Show or hide specific output types on the scheduler page. When an output type is hidden, it does not show on the Output Options page, but can still be accessed via REST.
- Disable specific output types for the schedulers.

Customizing Display of Available Outputs

The Output Options pages of the report scheduler and dashboard scheduler display the output types available for a scheduled report or dashboard.

To show or hide output options in the scheduler

- Edit the file `<js-webapp>/WEB-INF/applicationContext-report-scheduling.xml`.
- Locate the bean for the page you want to edit:
 - For report output formats, locate the `availableReportJobOutputFormats` bean.
 - For dashboard output formats, locate the `availableDashboardJobOutputFormats` bean.

For example, the `availableReportJobOutputFormats` bean might look like this:

```
<util:list id="availableReportJobOutputFormats" value-type="java.lang.String">
  <value>CSV</value>
  <value>HTML</value>
  <value>RTF</value>
  <value>DOCX</value>
  <value>ODS</value>
  <value>XLSX_NOPAG</value>
  <value>XLS</value>
  <value>ODT</value>
```

```

<value>XLSX</value>
<value>XLS_NOPAG</value>
<value>PDF</value>
<value>PPTX</value>
<!--<value>TXT</value>-->
<value>DATA_SNAPSHOT</value> <!-- This one will not appear if data snapshot persistence is
turned off -->
</util:list>

```

- To hide an output format on the Output Options page, comment out the corresponding option. To enable an option, uncomment it. For example, to hide PowerPoint (PPTX) output and enable text (TXT) output, edit those lines:

```

<util:list id="availableReportJobOutputFormats" value-type="java.lang.String">
  <value>CSV</value>
  <value>HTML</value>
  <value>RTF</value>
  <value>DOCX</value>
  <value>ODS</value>
  <value>XLSX_NOPAG</value>
  <value>XLS</value>
  <value>ODT</value>
  <value>XLSX</value>
  <value>XLS_NOPAG</value>
  <value>PDF</value>
  <!--<value>PPTX</value>-->
  <value>TXT</value>
  <value>DATA_SNAPSHOT</value> <!-- This one will not appear if data snapshot persistence is
turned off -->
</util:list>

```

- Save the modified files and reload the web app in the app server to see the changes (see [Reloading the JasperReports Server Web App](#)).
- Open the report scheduler by navigating to a report in the repository, right-clicking on the report, and selecting Schedule... from the context menu. Go to the Output Options tab to see that the visible options reflect your uncommented options in the applicationContext-report-scheduling.xml file.

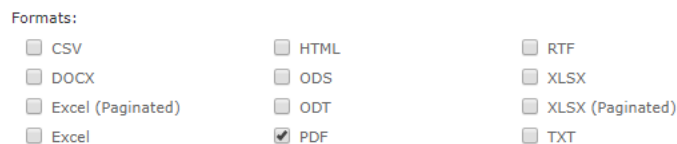


Figure 22: Modified List of Formats on Output Options Page

Disabling output formats

When you disable an output format, the corresponding output will not be generated even if it is selected in the UI or via REST.

1. Edit the file `<js-webapp>/WEB-INF/applicationContext-report-scheduling.xml`.
2. Locate the `outputFormatMap` bean:

```
<util:map id="outputFormatMap">
  <entry key="pdf" value-ref="pdfOutput"/>
  <entry key="html" value-ref="htmlOutput"/>
  <entry key="xls" value-ref="xlsOutput"/>
  <entry key="xlsNoPag" value-ref="xlsNoPagOutput"/>
  <entry key="rtf" value-ref="rtfOutput"/>
  <entry key="csv" value-ref="csvOutput"/>
  <entry key="odt" value-ref="odtOutput"/>
  <entry key="txt" value-ref="txtOutput"/>
  <entry key="docx" value-ref="docxOutput"/>
  <entry key="ods" value-ref="odsOutput"/>
  <entry key="xlsx" value-ref="xlsxOutput"/>
  <entry key="xlsxNoPag" value-ref="xlsxNoPagOutput"/>
  <entry key="pptx" value-ref="pptxOutput"/>
  <entry key="json" value-ref="jsonOutput"/>
</util:map>
```

3. Comment out the format(s) you do not want generated.
4. To avoid user confusion, hide disabled formats on the scheduler page(s), as described in the previous section. Even if the output format is selected on the scheduler page, the output will not be generated.
5. Save the modified files and reload the web app in the app server to see the changes (see [Reloading the JasperReports Server Web App](#)).

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 [Customizing the UI With Web App Files](#), 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.

- Commercial versions of JasperReports Server on Tomcat have an overlay upgrade procedure. Overlay upgrade allows you to pause the upgrade after the new files have been installed and manually migrate your customizations. Be careful to copy only your modifications from the saved file into the new server files. In some cases, the existing modifications no longer apply to the files or mechanisms in the new server. See the *JasperReports Server Upgrade Guide* for more information.
- 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 regular 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.

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 the community or commercial edition of JasperReports Server as long as all instances are the same edition, the same version, and all are configured identically.

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.

Clusters can be implemented on your own hardware, but JasperReports Server also provides more support for virtual machines in a cloud. Jaspersoft has partnered with Amazon to offer quick and affordable virtual instances in Amazon's EC2 (Elastic Compute Cloud) that can be used to create a cluster. By using CloudFormation Templates provided by Jaspersoft, you can easily launch pre-configured AMIs (Amazon Machine Instances) in your choice of size and location, and then connect them to your Amazon RDS (Relational Database Services) to create an on-demand cluster. You can create high-availability that is eminently scalable, requires no hardware, and you only pay for what you use.

One important detail of any cluster is how sessions are managed in case of a failure. JasperReports Server supports partial replication of sessions. In case of a node failure, the session is transferred automatically between nodes so that users are not logged out. However, work in the Ad Hoc editor and report viewer will be interrupted, though work saved in the repository can be resumed on the new 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, and should be used only 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](#)

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 JasperReports Server instances must access and scale them appropriately.

In general, a cluster may incorporate computers of different hardware and software configurations. For simplicity, we recommend deploying JasperReports Server as a cluster of identical nodes, with one instance on each node and all instances configured the same.

The following diagram shows the architecture of a sample JasperReports Server cluster:

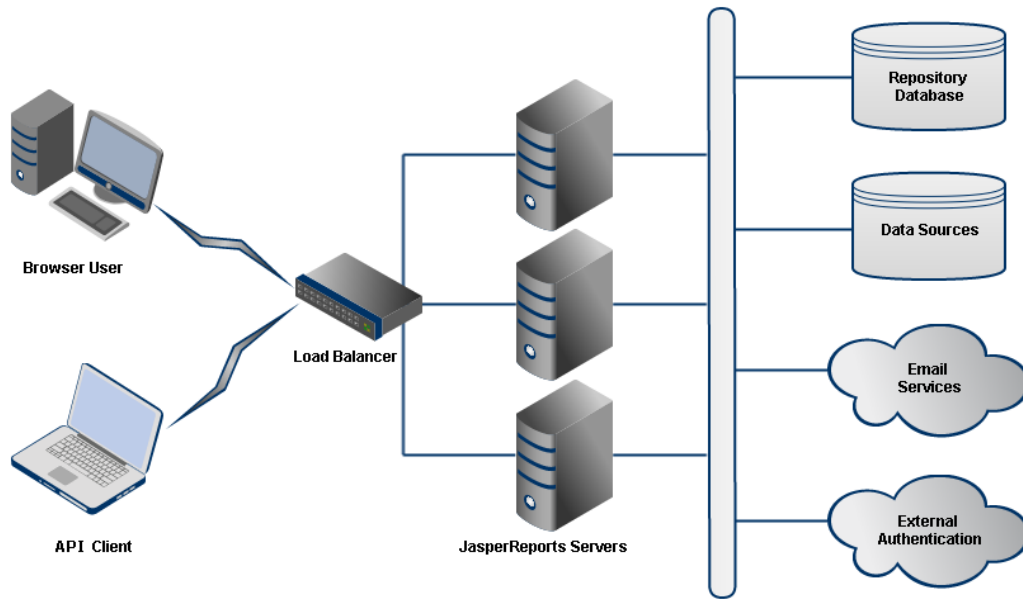


Figure 23: Architecture of a Sample JasperReports Server Cluster

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.
 - API clients – Applications that embed JasperReports Server functionality through the REST API or Visualize.js.
- 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 often required 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 beyond the scope of this chapter.

JasperReports Server Clients

JasperReports Server supports two fundamentally different client types:

- 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 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.
- API clients – Applications that use the REST APIs and Visualize.js to access the features of JasperReports Server. For example, Jaspersoft Studio relies on the REST API to access the JasperReports Server repository for reading and writing reports and their associated resources. A web application might use the Visualize.js library to embed JasperReports and dashboards within its own UI. For more information, see the *JasperReports Server REST API Reference* and *JasperReports Server Visualize.js Guide*.

An API client is an application that sends requests programmatically, and the server performs an action or provides structured information in a reply to the client. API calls are generally stateless operations, meaning that an entire operation is performed in a single request and does not rely on any previous requests. Complex structures such as a report resource must be fully defined and managed by the client before being sent in a request, as opposed to the interactive nature of the browser user.

User sessions in a cluster are further explained in [Session Management and Failover](#).

A significant task in the design of your cluster is to characterize client use, such as total number of potential clients, fraction of browser users and API 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 [Sizing a Cluster](#).

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 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 JasperReports Server supports partial session replication but not full replication, the load balancer must be configured so that browser users are always connected to the same server during a continuous session. Sessions should be transferred only if a node becomes disabled. For more information, see [Session Management and Failover](#). Beyond that requirement, JasperReports Server can work with any HTTP load balancer, either hardware or software. 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. Your cluster requirements ultimately determine your choice of load balancer.

JasperReports Server Instances

The JasperReports Server instances deployed in a cluster are normal servers with only minor configuration needed for the cluster. As with all server instances, they run in a Java servlet container that's usually implemented as an application server. While the load balancer may interact with the application servers, and the application servers and internal caches may communicate between nodes to maintain user sessions, JasperReports Server itself does not have any special behavior when deployed in the cluster environment.

In this document, a node refers to the computer hosting a JVM, an application server, and a single instance of JasperReports Server as the only application. Each node can be a physical computer or a cloud-based virtual computer, as long as the operating system is supported by JasperReports Server. For performance and stability reasons, we recommend that the only other software installed is 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.

We also recommend 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 for installing JasperReports Server in your choice of application server and to configure your JVM.

Alternatively, you can deploy virtual server instances pre-configured for high-availability in Amazon's Elastic Compute Cloud (EC2). By deploying virtual machines, you have the flexibility of scaling on-demand and paying only for what you use. If your repository also uses Amazon's Relational Database Service (RDS), you can create an entirely virtual cluster within Amazon's Web Service offerings. For CloudFormation Templates that can launch a variety of AMIs (Amazon Machine Instances) in your selected region, see [Launching Jaspersoft for AWS](#) on the Jaspersoft Community website.

The general procedure for deploying nodes in a cluster is as follows:

1. Install a single instance from the WAR file distribution or in the cloud, and configure it completely. This includes all your security settings, user settings, and customizations or authentication such as single sign-on (SSO).
2. Provision the physical or virtual machines for the other nodes of the cluster.
3. Copy the deployed WAR file to each of the other app servers 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.

4. In order to implement partial session replication, the application servers and internal caches must be configured to communicate with each other within the cluster. After cloning or deploying the server instances, you should configure each node separately, as described in [Sample Configurations](#).

Except for the session replication and internal caches, every node should have an identical configuration. While it is possible for each instance to be configured differently, it's not recommended. When instances are deployed on differing hardware with different configurations, the load balancer's algorithm may not be effective. Having identical JasperReports Server instances in your cluster greatly simplifies its installation and maintenance.

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 other's 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 load and average load in your cluster. 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.

You can also use Amazon RDS (Relational Database Service) as the repository to take advantage of its built-in scalability, high-availability, and backup security. If your server nodes are also EC2 (Elastic Compute Cloud) instances, you can create an entirely virtual cluster within Amazon's Web Service offerings. For more information, see [Launching Jaspersoft for AWS](#) on the Jaspersoft Community website.

Regardless of the architecture of the repository database, it's critical that it act as a single logical repository identified with a single IP address that all JasperReports Server instances can access.

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 23](#).

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 `<js-webapp>/WEB-INF/js.quartz.base.properties`. The default configuration of the scheduler includes the following settings that allow it to work in both stand-alone servers and clusters:

```
org.quartz.scheduler.instanceId = AUTO
org.quartz.jobStore.isClustered = true
```

Be sure to synchronize the clocks on all nodes, so 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.

Other Shared Resources

As with the repository database, the server instances in the cluster usually share any resource they need to access, including:

- Data sources – These are defined in the shared repository, so all servers in the cluster access the same data sources. Queries, reports, and Domains are all stored in the repository as well, so a report has access to exactly the same data, regardless of which instance the user session or job is running on. Make sure that data source and OLAP connections defined in the repository are able to handle connections from multiple nodes simultaneously.



Data sources can be defined as either JDBC connections directly to a database or JDBC connections from the application server to the database exposed through JNDI. We recommend 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 [JasperReports Server Instances](#).

- 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.

For resources that are configured in server files, such as email and external authentication, each node could have its own custom configuration. For example, each node 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, in order to provide the same reliability and scalability as the JasperReports Server cluster.

When resources are shared, they have a single address and set of credentials for access. This means that all JasperReports Server instances have identical configurations and are therefore easier to deploy and maintain.

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. Different behaviors result depending on how you define the XML/A connection in a cluster:

- In the default case, the definition of the XML/A provider points to the `localhost`, so the same node that receives the user request will both perform 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.



Because of the extra communication and connections, whether this configuration increases performance depends on your overall cluster load. If you have lots of analysis users at the same time, the load will be uniformly high, and there will be no benefit to calling the XML/A provider on a node that is equally busy. In fact, the extra overhead may impact performance, in particular if the connection loops back to the same node.

On the other hand, if your nodes are occasionally idle or if your load balancer detects real-time load on each node, this configuration will spread the analysis load and optimize performance for all users.

- You can set the connection to the URL of a specific XML/A provider. This could be a dedicated instance of JasperReports Server running Jaspersoft OLAP that's not connected to the cluster. You would need to size this instance to handle your expected analysis load and possibly implement two nodes as a cluster for availability. The advantage of having a dedicated XML/A provider instance is that it would centralize the cache for XML/A connections, thereby increasing cache hits.

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, we recommend 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.

Session Management and Failover

Failover is the ability of the cluster to minimize the impact of a node failure and continue serving clients with the remaining nodes. A failure is assumed to be any unplanned incident that causes a node, either its software or hardware, to become and remain unavailable. 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 the user creates interactively through the browser.

There are several types of session management in cluster design, each of which determines a different failover scenario:

- Fully replicated sessions – The state of every client session is continuously communicated among all nodes and stored on every node or in a shared location. This is usually managed by the app server. 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 unaware of the failure or the change. Some load balancers can transfer sessions when a node is overloaded but hasn't failed.



JasperReports Server does not support fully replicated sessions. See explanation below.

- 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 another node where they login and begin a new session on the new node.
- Partially replicated sessions – In this hybrid solution, only the vital parts of the session are replicated to allow some failover, but in the absence of node failure, sessions must remain pinned to a node.

Unlike small e-commerce sessions, JasperReports Server has larger and more complex sessions that would degrade performance if fully replicated. In particular, the JasperPrint object for an executed report is measured in megabytes and stored in the user session. To store and replicate the JasperPrint objects for all users across all nodes would impact performance significantly. Therefore, JasperReports Server uses partial replication so that user login information is replicated, but not large state objects such as JasperPrint objects.

By using partially replicated sessions, JasperReports Server balances the performance of the cluster with the risks and consequences of a failure. If a failure occurs when the user is browsing the repository, the load balancer will connect the user to a new node and work can continue because the login information and repository state in the session were replicated. If a failure occurs when a user is creating or viewing a report, the load balancer

will connect the user to a new node, but the report cannot be replicated. In this case users may lose unsaved reports, but they can resume work right away. The chance of a user experiencing a failure is much lower and the consequences are less severe than when working on a non-clustered server.

Even though the risk of failure is low, cluster designers should understand the impact on users. The following sections explain the exact impact of failures with partial session replication in JasperReports Server.

Impact on Browser Users

Thanks to partial session replication, a node failure has no impact on most users. If they're browsing the repository or scheduling a report, they won't even notice when failover to another node occurs. A node failure mainly affects interactive sessions where the user has unsaved work, such as in the Ad Hoc Editor and Dashboard Designer. In those cases, users are redirected to the home page on the new node, and they may need to re-create a report or dashboard, but they won't lose any critical functionality or data.

The following table lists the elements of the user session that cannot be replicated and explains the impact of failover on a user.

Non-Replicated Session Element	Interactive Feature Impacted upon Failover
State of Ad Hoc fields and data	The Ad Hoc editor display consists of fields placed in the canvas and the various filters and mode settings that determine what values are displayed, as well as visualizations such as table columns, crosstabs, and charts. These items are displayed with data from the data source. Upon failover, the state and data in the Ad Hoc editor cannot be restored, so any unsaved changes are lost.
State of Dashboard designer	The Dashboard designer displays reports and controls in a grid layout. Upon failover, the contents and state of the canvas cannot be restored, so any unsaved changes are lost.

Non-Replicated Session Element	Interactive Feature Impacted upon Failover
JasperPrint object from running a report	Reports in the report viewer rely on the JasperPrint object for pagination and exporting in other formats. Upon failover, reports in the report viewer cannot be restored, and any unsaved changes such as interactive sorting and filtering are lost.
State of the Domain designer	The Domain designer is a set of tabs for defining the tables, joins, filters, calculated fields, and default names for the fields of a Domain. Upon failover, the changes made in the Domain designer tabs cannot be restored and any unsaved changes are lost.
OLAP viewer state	The OLAP viewer lets you explore the dimensions and values in an OLAP cube by changing the MDX expression that defines the view. Upon failover any unsaved view is lost and would need to be created again through the same transformations (slicing and dicing).
Administration dialogs (when creating or editing an organization, user, or role)	Any information entered in an administration dialog is lost if it was not submitted before node failure.

For example, when a node fails during interaction with the Ad Hoc editor, the user is redirected by the load balancer to another node, which like all nodes, has only the partially replicated user session. Because it does not have the state of the user's Ad Hoc changes, it cannot save or even display the work started on the failed node. In this event, the interactive work is lost, and users see an error message on the new node. Upon acknowledging the message, users are redirected to the home page.

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 the user explicitly saves is stored in the shared repository and can be reopened in the new session on the new node. This includes any resources created, moved, or saved for any Save or Save As operation that completes before a failure.

For all cases not listed in the table above, work in progress is preserved upon failover, and the user can continue to work immediately on the new node. For example, if the user was searching the repository at the time of failure, all parts of the session needed for search are replicated. So the user can continue interacting with search results immediately on the new node – usually without even noticing there was a failure.

Also, information for the following dialogs are part of the replicated session, and are successfully transferred to a new node in case of a failover:

- Repository permissions dialog
- Add folder dialog
- Add resource dialogs, including adding or editing a data source, JasperReport, and other repository objects
- Copy, cut, and pasting resources in the repository
- Scheduling a report

Impact on API Clients

Applications can invoke to interact programmatically with JasperReports Server through the REST API and Visualize.js. API calls can also be interrupted by a node failure, but because operations are quicker, consisting of a short request and almost immediate response, this is much less likely. Applications that call APIs should implement a reasonable timeout and verify the return value to determine if the server instance completed the operation or had 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 direct to a remaining node.

In general, API calls are stateless, meaning the server does not store any information about the operation or the caller from one API request to the next. This is why APIs do not support interactive work such as designing an Ad Hoc view, a dashboard, a Domain, or exploring data in OLAP, all of which require the server to store an intermediate state.

In the case of the interactive wizards for creating and editing complex repository resources, API calls to do the equivalent operations are also stateless. When using APIs, the calling application has the responsibility to remember the state of the interaction between it and the server. For example, in the UI, an interactive wizard helps administrators define nested resources such as queries and input controls. When using web services, the client application must remember which nested resources have been created and how to reference them.

In practice, the REST API is not entirely stateless because it supports a login that allows the client to use the session cookie instead of HTTP authentication with every call. However, login information is included in the partially replicated session and therefore handled transparently in the case of node failure and failover. So if a login request completes on one node but that node eventually fails, the load balancer sends further requests to a new node that accepts the same session cookie because it has the partially replicated session.

There is one more case where web services rely on a session object: the JasperPrint object cannot be replicated, as explained in the following table.

Non-Replicated Session Element	REST API Impacted by Failure
JasperPrint object from running a report	The initial call to the <code>rest_v2/reportExecutions</code> service runs a report and stores a JasperPrint object in the user session. Upon failover, subsequent calls to the same service will return an error. Your application must have logic to detect this error and perform the initial call again.

Sample Configurations

The app server usually manages the user session for a web application and is responsible for the policies that allow the session to be replicated in a cluster environment. However, you must also configure parts of JasperReports Server for repository and session replication, including the Ehcache component.



This section describes how to configure JasperReports Server for session replication. To configure your application server, see the documentation for your application server. For additional information, see the section "Configuration for Session Persistence" in the *JasperReports Server Administrator Guide*.

This section describes two levels of replication for JasperReports Server:

- Ehcache replication only. The repository cache handles the folder structure and saved reports, and speeds up repository access in a given instance of JasperReports Server. Changes to permissions and folders are cached on the server where they occur, but they can take one to two minutes to be written to the repository database. To maintain performance and avoid collisions, you should configure Ehcache replication whenever you have multiple JasperReports Server instances that

share a single repository. Ehcache replication can be configured independently of session replication.

- Partial session replication for failover. Partial session replication shares, based on Ehcache replication, shares additional information about the logged-in users and allows for failover without requiring re-authentication. If you configure this, you must first configure Ehcache replication.

EhCache Replication

The repository cache in JasperReports Server is implemented internally via the Ehcache component. Edit the Ehcache configuration files as described below to replicate the repository cache among all instances that share a single repository. You do not have to configure a cluster for cache replication.

There are several replication mechanisms available:

- RMI – Remote Method Invocation is the simplest and fastest cache distribution mechanism. Use RMI distribution if your cluster runs on your own real or virtual computers, as long as their addresses will not change. You cannot use RMI distribution if your cluster is hosted in a cloud, such as with Amazon Redshift, because the IP addresses of the nodes may change. RMI distribution relies on IP multicast, which you must set up.
- JMS – Java Message Services can provide cache distribution for nodes in a cloud where IP addresses may change. Jaspersoft provides a configuration for using the [Apache ActiveMQ JMS Server](#). You must first install and configure ActiveMQ on one of the computers in your cluster.

On each node, you must edit the following cache configuration files. Make sure to uncomment only one of the options provided in each file:

- Ehcache for Hibernate: Edit the following file as described below.
 - `/WEB-INF/classes/ehcache_hibernate.xml`. (Once the file is fully configured, you will also copy it to `/WEB-INF/ehcache_hibernate.xml`.)
- Ehcache: Edit the following file as described below.
 - `<web-app>/WEB-INF/ehcache.xml`

To configure JasperReports Server nodes for repository cache replication

1. If you are using RMI distribution, you must make sure that the subnet that contains all the nodes is configured to allow IP multicasting.
2. For all distribution mechanisms, comment out the section marked "NO CLUSTERING" in both files as follows. By default, this section is uncommented. For example, in the ehcache_hibernate.xml, comment out the "NO CLUSTERING" section as follows:

```
<!-- ***** NO CLUSTERING ***** -->
<!-- START
<cache name="defaultRepoCache"
  maxElementsInMemory="100000"
  statistics="false"
  eternal="true"
  overflowToDisk="false"
  timeToIdleSeconds="36000"
  timeToLiveSeconds="180000"
  diskExpiryThreadIntervalSeconds="120"
  diskPersistent="false"/>
  END -->
<!-- ***** END of NO CLUSTERING ***** -->
```

3. (RMI only) To configure the nodes for RMI distribution:
 - a. Uncomment the RMI section in <web-app>/WEB-INF/classes/ehcache_hibernate.xml and <web-app>/WEB-INF/ehcache.xml on each node.
 - b. Set the RMI properties for your IP multicast.
 - c. Configure CacheManagerPeerListenerFactory with different ports in <web-app>/WEB-INF/classes/ehcache_hibernate.xml and <web-app>/WEB-INF/ehcache.xml. The port should be the same across all nodes. For example, you might set the port property as follows:
 - port=40001 in <web-app>/WEB-INF/classes/ehcache_hibernate.xml on all nodes
 - port=40011 in <web-app>/WEB-INF/ehcache.xml on all nodes
 - d. You must also add the hostname property with the value of the real IP address, in this example, 123.45.6.701. Add the hostName property to the cacheManagerPeerListenerFactory, right before the port. This specifies the real IP address of the host, as shown in the example above.

The following example shows the beginning of the RMI section of one of the files:

```
<!-- ===== RMI ===== -->
<cacheManagerPeerProviderFactory
  class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
  properties="peerDiscovery=automatic,multicastGroupAddress=228.0.0.1,
```



```

        multicastGroupPort=4446,timeToLive=32"/>
    <cacheManagerPeerListenerFactory
        class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"
        properties="hostName=123.45.6.701,port=40001,socketTimeoutMillis=120000"/>

    ...

<!-- ===== END OF RMI ===== -->

```

4. (JMS only) For JMS distribution:

- a. Install the JMS server on one computer in your cluster.
- b. Uncomment the JMS section in <web-app>/WEB-INF/classes/ehcache_hibernate.xml and <web-app>/WEB-INF/ehcache.xml on each node.
- c. Set the providerURL properties in both files to the address of your JMS server, for example 123.45.6.701. There are several providerURL properties to set in each file, only the first one is shown in the code example below.
- d. If you do not use the default values for replicationTopicBindingName and topicBindingName, make sure that these names are different for the two different files.

For example, you might set these properties as follows:

- myName1 in <web-app>/WEB-INF/classes/ehcache_hibernate.xml on all nodes.
- myName2 in <web-app>/WEB-INF/ehcache.xml on all nodes.

```

<!-- ===== JMS ===== -->

    <cacheManagerPeerProviderFactory
        class="net.sf.ehcache.distribution.jms.JMSCacheManagerPeerProviderFactory"
        properties="initialContextFactoryName=com.jaspersoft.jasperserver.api.
engine.replication.JRSActiveMQInitialContextFactory,
        providerURL=tcp://123.45.6.701:61616,
        replicationTopicConnectionFactoryBindingName=topicConnectionFactory,
        replicationTopicBindingName=ehcacheAcl,
        getQueueConnectionFactoryBindingName=queueConnectionFactory,
        getQueueBindingName=ehcacheQueueAcl,
        topicConnectionFactoryBindingName=topicConnectionFactory,
        topicBindingName=ehcacheAcl"
        propertySeparator=","/>

    ...

<!-- ===== END OF JMS ===== -->

```

5. On each node, edit the file <web-app>/META-INF/context.xml. Locate the Manager pathname near the end, and comment it out as follows:

```
<!-- <Manager pathname="" /> -->
```

6. Copy the correctly configured `<web-app>/WEB-INF/classes/ehcache_hibernate.xml` file to `<web-app>/WEB-INF/ehcache_hibernate.xml` on each node.
7. If you are only configuring cache replication, restart or redeploy JasperReports Server on each node. If you also want partial session replication, complete the additional configurations below before restarting.

Additional Configurations for Partial Session Replication

If you want to configure partial session replication/failover, first set up repository cache replication as described in [EhCache Replication](#), then make the additional changes described in this section.

On each node, edit *all* of the following three files as described below for your chosen distribution mechanism. You should make the same additional changes in all of them. Make sure to uncomment only one of the options provided in each file:

- `<web-app>/WEB-INF/ehcache_hibernate.xml`
- `<web-app>/WEB-INF/classes/ehcache_hibernate.xml`
- `<web-app>/WEB-INF/ehcache.xml`



Partial session replication, which is based on UDP (User Datagram Protocol), is not available for Amazon Web Services.

To configure JasperReports Server nodes for partial session replication

1. Make sure that the subnet that contains all the cluster nodes is configured to allow IP multicasting. This is usually required by the app server for replication, and it's also required by JasperReports Server's Ehcache component when using RMI distribution in a cluster environment.
2. On each node of the cluster, edit the file `<web-app>/WEB-INF/web.xml` to make the following changes:
 - a. Locate the `ClusterFilter` that's given in comments and uncomment it as follows:

```
<filter>
  <filter-name>ClusterFilter</filter-name>
  <filter-class>com.jaspersoft.jasperserver.war.TolerantSessionFilter</filter-class>
</filter>
```

- b. Locate the corresponding mapping for the `ClusterFilter` and uncomment it as well. You must also uncomment the `<distributable>` element.

```
<filter-mapping>
  <filter-name>ClusterFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<distributable/>
```

3. On each node of the cluster, enable session replication in your app server or web container. For example, to enable session replication on Apache Tomcat 9.x, edit the file `<tomcat>/conf/server.xml` as follows.

Add the `Cluster` definition within the `<Engine name="Catalina" defaultHost="localhost">` configuration. In this example, `123.45.6.701` is the IP address of the node being configured. This example uses `Delta Manager`, but you can also use `Backup Manager`:

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
  channelSendOptions="8">
  <Manager className="org.apache.catalina.ha.session.DeltaManager"
    expireSessionsOnShutdown="false"
    notifyListenersOnReplication="true"/>
  <Channel className="org.apache.catalina.tribes.group.GroupChannel">
    <Membership className="org.apache.catalina.tribes.membership.
      McastService"
      address="228.0.0.4"
      port="45564" frequency="500"
      dropTime="3000"/>
    <Sender className="org.apache.catalina.tribes.transport.
      ReplicationTransmitter">
      <Transport className="org.apache.catalina.tribes.
        transport.nio.PooledParallelSender"/>
    </Sender>
    <Receiver className="org.apache.catalina.tribes.transport.
      nio.NioReceiver"
      address="123.45.6.701" port="4000" autoBind="100"
      selectorTimeout="5000" maxThreads="6"/>
    <Interceptor className="org.apache.catalina.tribes.group.
      interceptors.TcpFailureDetector"/>
    <Interceptor className="org.apache.catalina.tribes.group.
      interceptors.MessageDispatch15Interceptor"/>
  </Channel>

  <!--<Valve className="org.apache.catalina.ha.tcp.ForceReplicationValve"/>-->
  <Valve className="org.apache.catalina.ha.tcp.ReplicationValve" filter=""/>
  <Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>
  <ClusterListener className="org.apache.catalina.ha.session.ClusterSessionListener"/>
</Cluster>
```

- Restart or redeploy JasperReports Server on each node.

Load Balancer Configuration

This section gives two examples of load balancer configuration, Apache web server (httpd) and HAProxy. These sample configuration files are meant only as examples for testing.

Apache HTTP Server Example

The following changes configure the [Apache HTTP server](#) (httpd) as a load balancer.

- Add following line to the end of the httpd.conf file:

```
Include conf/mod-jk.conf
```

- Create the following two files in the /conf folder of your httpd server.

- mod-jk.conf:

```
# Load mod_jk module
# Specify the filename of the mod_jk lib
LoadModule jk_module modules/mod_jk.so

# Where to find workers.properties
JkWorkersFile conf/workers.properties

# Where to put jk logs
JkLogFile logs/mod_jk.log

# Set the jk log level [debug/error/info]
JkLogLevel info

# Select the log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y]"

# JkOptions indicates to send SSK KEY SIZE
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories

# JkRequestLogFormat
JkRequestLogFormat "%w %V %T"

# Mount your applications
JkMount /* loadbalancer

# You can use external file for mount points.
# It will be checked for updates each 60 seconds.
# The format of the file is: /url=worker
# /examples/*=loadbalancer
```

```
#JkMountFile conf/uriworkermap.properties

# Add shared memory.
# This directive is present with 1.2.10 and
# later versions of mod_jk, and is needed for
# for load balancing to work properly
JkShmFile logs/jk.shm
#JkShmFile /var/log/httpd/mod_jk.shm

# Add jkstatus for managing runtime data
<Location /jkstatus>
    JkMount status
    Order deny,allow

    Allow from all
</Location>
```

- workers.properties:

```
# Define list of workers that will be used
# for mapping requests
worker.list=loadbalancer,status

# Define Node1
# modify the host as your host IP or DNS name.
worker.node1.port=8009
worker.node1.host=123.45.6.701
worker.node1.type=ajp13
worker.node1.lbfactor=1

# Define Node2
# modify the host as your host IP or DNS name.
worker.node2.port=8009
worker.node2.host=123.45.6.702
worker.node2.type=ajp13
worker.node2.lbfactor=1

# Load-balancing behaviour
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=node1,node2
worker.loadbalancer.sticky_session=1
#worker.list=loadbalancer

# Status worker for managing load balancer
worker.status.type=status
```

HAProxy Load Balancer Example

The following example is a configuration for the [HAProxy](#) load balancer. HAProxy is a high-performance, free and open source load balancer for Linux/x86 and Solaris/Sparc.

Edit the `/etc/haproxy.cfg` file as follows:

```

global
    log          127.0.0.1 local2 debug #log using syslog service on localhost
    maxconn     4096 # Total Max Connections. This is dependent on ulimit
    daemon

defaults
    mode        http
    maxconn     4096
    clitimeout  60000
    srvttimeout 30000
    contimeout  4000
    option      httpclose # Disable Keepalive
    log         global
    option      httplog

listen farm 123.45.6.700:80
    mode http
    stats uri /haproxy #show haproxy colsole
    balance roundrobin
    cookie farmID insert #assign a farmID cookie to each client
    option httpclose
    option httpchk
    option forwardfor

    ## Define your servers to balance
    server node1 123.45.6.701:8081 cookie farmID_node1 check
    server node2 123.45.6.702:8081 cookie farmID_node2 check

```

Troubleshooting

In some cases, when you are using a proxy or load balancer, your users may have problems with loading links and other resource files in hypermedia documents. This can occur when JasperReports Server generates links to a specific node instead of the host server.

For example, suppose you have a setup with a proxy host and two servers as follows:

```

proxy_host      http://bi.example.com
node 1          http://node1:8080/jasperserver-pro
node 2          http://node2:8080/jasperserver-pro

```

When the user accesses `http://bi.example.com`, JasperReports Server forwards the user to the login page on the first available node, `http://node1:8080/jasperserver-pro`. After login, when the user accesses hypermedia, JasperReports Server generates links to hypermedia resources using the base `http://node1:8080/jasperserver-pro` instead of `http://bi.example.com`.

To resolve this, set the following in the `/WEB-INF/js.config.properties` file on all nodes:

```

deploy.base.url=http://bi.example.com

```

Cluster Design Process

The rest of this chapter looks at 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.

Performance Requirements

The two sides to performance are performance requirements and load estimates. The goal is to anticipate the number of users and their activities to 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, such as “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'll interact with JasperReports Server. First, you should determine what volume of users will be browser clients (real people) and API clients (applications making API calls) and the ratio between them. See [Session Management and Failover](#) for information about these two types of clients and how their sessions differ.

Then 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 API 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, you may want 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 [Job Schedulers](#).

Client connectivity is also an issue, because the slowest link in the network creates a bottleneck. There's a difference in system performance and scalability between users working remotely over DSL and those in the office on your corporate T1 network.

Availability Requirements

High availability depends on the cluster's ability to effectively avoid downtime. In general, failover prevents the total system unresponsiveness that happens when a single server fails, but a properly designed cluster must also address the failure of other cluster components.

High availability is usually defined as a high percentage of uptime, such as 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 failure can bring the entire environment to a stop.

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. See [Session Management and Failover](#) for an explanation of what happens to users on a server instance that fails. You should also consider the overall performance degradation of the cluster and the impact of failures on other cluster components.

A further aspect of high availability is sizing the cluster so, should one server (hardware or software) become unavailable, the rest can still respond to anticipated 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 to properly assess 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.

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). You should also 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 ensure future availability. Your initial cluster design should recognize and allow for this expansion.

Once you have defined your scalability requirements in detail, use this information when sizing your cluster.

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 optimized for the user load and cluster environment.
- Shared resources, especially databases for the repository and data sources.
- Network service levels or upgrades.
- Policies and procedures for scaling and maintaining performance.

The following sections look at the trade-offs of various designs for each component.

Load Balancer

Determine whether you need a dedicated hardware load-balancer or if a basic software load-balancer will meet your requirements. You should also specify the load-balancing techniques you'll 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.

Cluster Nodes

The size and number of nodes determines 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 mismatched 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.

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 use.
- 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 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. For more information, see the chapter on server diagnostics in the *JasperReports Server Administrator Guide*.

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 hosting the shared repository must be carefully evaluated and specified in the cluster design.

Alternatively, you can use Amazon RDS as the repository, thus allowing you to create your cluster entirely within Amazon's Web Service offerings. For more information, see [Launching Jaspersoft for AWS](#) on the Jaspersoft Community website.

- 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 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.

Network

Finally, consider the availability and quality of your network. How remote are your users and how do they connect to the cluster?

The two main concerns about your network are:

- Network availability and capacity affects the cluster's availability and scalability. If you have strict availability requirements, you may need redundant network connections from different providers. For scalability and performance requirements, 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.
- 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 API clients. Network capacity (megabytes per second) is critical to being able to deliver these generated documents in reasonable time. The slower the network, the more load on the servers generating the report documents, as they'll take longer to deliver the same content, and potentially lead to more simultaneous report requests.

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 to 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 use 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 data explorer in JasperReports Server Professional and Enterprise editions. Starting from a predefined collection of fields, the Ad Hoc Editor lets you drag and drop fields, dimensions, and measures to explore data and create tables, charts, and crosstabs. These Ad Hoc views can be saved as reports.

Ad Hoc Report

In previous versions of JasperReports Server, a report was created through the Ad Hoc Editor. Such reports could be added to dashboards and be scheduled, but when edited in Jaspersoft Studio, lost their grouping and sorting. In the current version, the Ad Hoc Editor is used to explore views that in turn can be saved as reports. Such reports can be edited in Jaspersoft Studio without loss, and can be scheduled and added to dashboards.

Ad Hoc View

A view of data that is based on a Domain, Topic, or OLAP client connection. An Ad Hoc view can be a table, chart, or crosstab and is the entry point to analysis operations such as slice and dice, drill down, and drill through. Compare [OLAP View](#) You can save an Ad Hoc view as a report to edit it in the interactive viewer, schedule it, or add it to a dashboard.

Aggregate Function

An aggregate function is one that is computed using a group of values; for example, Sum or Average. Aggregate functions can be used to create calculated fields in Ad Hoc views. Calculated fields containing aggregate functions cannot be used as fields or added to groups in an Ad Hoc view and should not be used as filters. Aggregate functions allow you to set a level, which specifies the scope of the calculation; level values include Current (not available for PercentOf), ColumnGroup, ColumnTotal, RowGroup, RowTotal, Total.

Amazon Web Services (AWS)

Cloud platform, used to provide and host a family of services, such as RDS, S3, and EC2.

Analysis View

See [OLAP View](#).

Audit Archiving

To prevent audit logs from growing too large to be easily accessed, the 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 repository 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 an Ad Hoc view or a Domain, a field whose value is calculated from a user-defined formula that may include any number of fields, operators, and constants. For Domains, a calculated field becomes one of the items to which the Domain's security file and locale bundles can apply. There are more functions available for Ad Hoc view calculations than for Domains.

CloudFormation (CF)

Amazon Web Services CloudFormation gives developers and systems administrators an easy way to create and manage a collection of related AWS resources, provisioning, and updating them in an orderly and predictable fashion.

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 parametrize 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.

Dashlet

An element in a dashboard. Dashlets are defined by editable properties that vary depending on the dashlet type. Types of dashlet include reports, text elements, filters, and external web content.

Data Island

A single join tree or a table without joins in a Domain. A Domain may contain several data islands, but when creating an Ad Hoc view from a Domain, you can only select one of them to be available in the view.

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 > Server > Settings Ad Hoc Settings. By default, this setting is only available to the superuser account.

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 of 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.

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.

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 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 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

In Jaspersoft Studio, a frame is a rectangular element that can contain other elements and optionally draw a border around them. Elements inside a frame are positioned relative to the frame, not to the band, and when you move a frame, all the elements contained in the frame move together. A frame automatically stretches to fit its contents.

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 checkbox, dropdown 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.

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.

JasperReport

A combination of a report template and data that produces a complex document for viewing, printing, or archiving information. In the server, a JasperReport references other resources in the repository:

- The report template (in the form of a JRXML file)
- Information about the data source that supplies data for the report
- Any additional resources, such as images, fonts, and resource bundles referenced by the report template.

The collection of all the resources that are referenced in a JasperReport is sometimes called a report unit. End users usually see and interact with a JasperReport as a single resource in the repository, but report creators must define all of the components in the report unit.

JasperReports IO

An HTTP-based reporting service for JasperReports Library that provides a REST API for running, exporting, and interacting with reports and a JavaScript API for embedding reports and their input controls into your web pages and web applications.

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 Studio

A commercial open source tool for graphically designing reports that leverage all features of the JasperReports Library. Jaspersoft Studio lets you drag and drop fields, charts, and sub-reports onto a canvas, and also define parameters or expressions for each object to create pixel-perfect reports. You can generate the JRXML of the report directly in Jaspersoft Studio, or upload it to JasperReports Server. Jaspersoft Studio is implemented in Eclipse.

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.

Jaspersoft Studio

An open source tool for graphically designing reports that leverage all features of the JasperReports Library. Jaspersoft Studio lets you drag and drop fields, charts, and sub-

reports onto a canvas, and also define parameters or expressions for each object to create pixel-perfect reports. You can generate the JRXML of the report directly in Jaspersoft Studio, or upload it to JasperReports Server. Jaspersoft Studio is implemented in Eclipse.

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 a 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 Jaspersoft Studio and JasperReports Server. JRXML is an open format that uses the XML standard to define precisely all the structure and configuration of a report.

Level

Specifies the scope of an aggregate function in an Ad Hoc view. Level values include Current (not available for PercentOf), ColumnGroup, ColumnTotal, RowGroup, RowTotal, Total.

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. OLAP client connections populate OLAP views.

Mondrian Schema Editor

An open source Eclipse plug-in for creating Mondrian OLAP schemas.

Mondrian XML/A Source

A server-side XML/A source definition of a remote client-side XML/A connection used to populate an OLAP view using the XML/A standard.

MySQL

An open source relational database management system. For more 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 data to populate 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. Unlike Ad Hoc views, you can directly edit an OLAP view's MDX query to change the data and the way they are displayed. An OLAP view is the entry point for advanced analysis users who want to write their own queries. [Compare Ad Hoc View](#).

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 suborganizations 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 help desk 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.

Report

In casual usage, a report may refer to:

- A JasperReport. See [JasperReport](#).
- The main JRXML in a JasperReport.
- The file generated when a JasperReport is scheduled. Such files are also called content resources or output files.
- The file generated when a JasperReport is run and then exported.
- In previous JasperReports Server versions, a report is created in the Ad Hoc Editor. See [Ad Hoc Report](#).

Report Run

An execution of a report, Ad Hoc view, or dashboard, or a view or dashboard designer session, it measures and limits usage of Freemium instances of JasperReports Server. The executions apply to resources no matter how they are run (either in the web interface or through the various APIs, such as REST web services). Users of our Community Project and our full-use commercial licenses are not affected by the limit. For more information, please contact sales@jaspersoft.com.

Repository

Depending on the context:

- In JasperReports Server, the repository is the tree structure of folders that contain all saved reports, dashboards, OLAP views, and resources. Users can access the repository through the JasperReports Server web interface or through Jaspersoft Studio. Applications can access the repository through the web service API. Administrators use the import and export utilities to back up the repository contents.
- In JasperReports IO, the repository is where all the resources needed to create and run reports are stored. The repository can be stored in a directory on the host computer or in an S3 bucket hosted by Amazon Web Services. Users can access the repository through a file browser on the host machine or through the AWS console.

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.

S3 Bucket

Cloud storage system for Amazon Web Services. JasperReports IO can use an S3 bucket to store files for its repository.

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 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.

Stack

A collection of Amazon Web Services resources you create and delete as a single unit.

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 opens the Ad Hoc Editor.

Transactional Data

Data that describes 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.

View

Several meanings pertain to JasperReports Server:

- An Ad Hoc view. See [Ad Hoc View](#).
- An OLAP view. See [OLAP View](#).
- A database view. See http://en.wikipedia.org/wiki/View_%28database%29.

Virtual Data Source

A virtual data source allows you to combine data residing in multiple JDBC and/or JNDI data sources into a single data source that can query the combined data. Once you have created a virtual data source, you create Domains that join tables across the data sources to define the relationships between the data sources.

WCF

Web Component Framework. A low-level GUI component of JPivot. For more information, see <http://jpivot.sourceforge.net/wcf/index.html>.

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 the 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 access data on a remote server. OLAP client connections populate OLAP views.

Jaspersoft Documentation and Support Services

For information about this product, you can read the documentation, contact Support, and join Jaspersoft Community.

How to Access Jaspersoft Documentation

Documentation for Jaspersoft products is available on the [Product Documentation website](#), mainly in HTML and PDF formats.

The [Product Documentation website](#) is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The documentation for this product is available on the [JasperReports® Server Product Documentation](#) page.

How to Access Related Third-Party Documentation

When working with JasperReports® Server, you may find it useful to read the documentation of the following third-party products:

How to Contact Support for Jaspersoft Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our [product Support website](#).
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the [product Support website](#). If you do not have a username, you can request one by clicking **Register** on the website.

How to Join Jaspersoft Community

Jaspersoft Community is the official channel for Jaspersoft customers, partners, and employee subject matter experts to share and access their collective experience. Jaspersoft Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from Jaspersoft products. In addition, users can submit and vote on feature requests from within the [Jaspersoft Ideas Portal](#). For a free registration, go to [Jaspersoft Community](#).

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

Jaspersoft, JasperReports, Visualize.js, and TIBCO are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.tibco.com/patents>.

Copyright © 2005-2024. Cloud Software Group, Inc. All Rights Reserved.