



JASPERREPORTS SERVER WEB SERVICES GUIDE

RELEASE 5.2

<http://www.jaspersoft.com>

Copyright © 2013 JasperSoft Corporation. All rights reserved. Printed in the U.S.A. JasperSoft, the JasperSoft logo, JasperSoft iReport Designer, JasperReports Library, JasperReports Server, JasperSoft OLAP, and JasperSoft ETL are trademarks and/or registered trademarks of JasperSoft Corporation in the United States and in jurisdictions throughout the world. All other company and product names are or may be trade names or trademarks of their respective owners.

This is version 0613-JSP52-20 of the *JasperReports Server Web Services Guide*.

TABLE OF CONTENTS

Chapter 1 Introduction	9
1.1 REST Web Services Overview	9
1.2 REST Authentication	12
1.2.1 Login Encryption	13
1.2.2 Login Service	13
1.3 REST Server Information	15
1.4 SOAP Web Services Overview	16
1.5 SOAP Authentication	16
1.6 Syntax of resourceDescriptor	17
1.6.1 Overview	17
1.6.2 wsType Attribute	18
1.6.3 isNew Attribute	18
1.6.4 Resource Descriptor Parameters	19
1.6.5 Examples of resourceDescriptor	19
Chapter 2 REST - Repository Web Services	21
2.1 The resources Service	21
2.2 The resource Service	23
2.2.1 Requesting the Contents of a JasperReport	25
2.2.2 Requesting the Contents of a File Resource	28
2.2.3 Requesting the Values of a Query-Based Input Control	28
2.2.4 Creating a Resource	31
2.2.5 Setting the Temporary Upload Directory	32
2.2.6 Modifying a Resource	32
2.2.7 Copying or Moving a Resource	33
2.2.8 Deleting a Resource	33
2.3 Working with Dashboards	34
2.4 Working with Virtual Data Sources	37
2.5 Working with Domains	38
2.6 The permission Service	40

2.6.1	Viewing Permissions	40
2.6.2	Setting Permissions	42
2.7	The v2/permissions Service	43
2.7.1	Viewing Multiple Permissions	43
2.7.2	Viewing a Single Permission	44
2.7.3	Setting Multiple Permissions	44
2.7.4	Setting a Single Permission	46
2.7.5	Deleting Permissions in Bulk	46
2.7.6	Deleting a Single Permission	47
2.8	The v2/export Service	47
2.8.1	Checking the Export State	48
2.8.2	Fetching the Export Output	49
2.9	The v2/import Service	49
Chapter 3	REST - Report Web Services	51
3.1	The report Service	51
3.1.1	Running a Report	52
3.1.2	Downloading Report Output	53
3.1.3	Regenerating Report Output	53
3.2	The v2/reports Service	54
3.2.1	Running a Report	54
3.2.2	Finding Running Reports	55
3.2.3	Terminate Running Report	55
3.3	The v2/reportExecutions Service	56
3.3.1	Running a Report Asynchronously	56
3.3.2	Polling Report Execution	58
3.3.3	Requesting Report Execution Details	59
3.3.4	Requesting Report Output	60
3.3.5	Exporting a Report Asynchronously	61
3.3.6	Polling Export Execution	62
3.3.7	Finding Running Reports and Jobs	62
3.3.8	Stopping Running Reports and Jobs	63
3.4	The v2/inputControls Service	64
3.4.1	Listing Input Control Structure	64
3.4.2	Listing Input Control Values	65
3.4.3	Setting Input Control Values	66
3.5	The v2/options Service	67
3.5.1	Listing Report Options	67
3.5.2	Creating Report Options	68
3.5.3	Updating Report Options	69
3.5.4	Deleting Report Options	69
3.6	The jobsummary Service	70
3.7	The job Service	70
3.7.1	Viewing a Job Definition	71
3.7.2	Scheduling a Report	72

3.7.3	Editing a Job Definition	73
3.7.4	Deleting a Job Definition	73
3.8	The v2/jobs Service	73
3.8.1	Listing Report Jobs	74
3.8.2	Viewing a Job Definition	74
3.8.3	Extended Job Search	76
3.8.4	Scheduling a Report	76
3.8.5	Viewing Job Status	77
3.8.6	Editing a Job Definition	77
3.8.7	Updating Jobs in Bulk	77
3.8.8	Pausing Jobs	78
3.8.9	Resuming Jobs	79
3.8.10	Restarting Failed Jobs	79
3.8.11	Specifying FTP Output	80
3.8.12	Calendar Exclusion for the Scheduler	80
3.9	The v2/queryExecutor Service	85
Chapter 4 REST - Web Services for Administration		89
4.1	The organization Service	89
4.1.1	Viewing an Organization	89
4.1.2	Creating an Organization	91
4.1.3	Modifying Organization Properties	91
4.1.4	Deleting an Organization	91
4.2	The user Service	92
4.2.1	Creating a User	93
4.2.2	Editing a User	93
4.2.3	Deleting a User	94
4.3	The attribute Service	94
4.4	The role Service	95
4.4.1	Creating a New Role	96
4.4.2	Editing a Role	96
4.4.3	Deleting a Role	96
4.5	The v2/organizations Service	96
4.5.1	Searching for Organizations	97
4.5.2	Viewing an Organization	98
4.5.3	Creating an Organization	98
4.5.4	Modifying Organization Properties	99
4.5.5	Setting the Theme of an Organization	100
4.5.6	Deleting an Organization	100
4.6	The v2/users Service	101
4.6.1	Searching for Users	101
4.6.2	Viewing a User	102
4.6.3	Creating a User	104
4.6.4	Modifying User Properties	104
4.6.5	Deleting a User	105

4.7	The v2/attributes Service	105
4.7.1	Viewing User Attributes	106
4.7.2	Setting User Attributes	107
4.7.3	Deleting User Attributes	109
4.8	The v2/roles Service	110
4.8.1	Searching for Roles	110
4.8.2	Viewing a Role	111
4.8.3	Creating a Role	112
4.8.4	Modifying a Role	112
4.8.5	Setting Role Membership	113
4.8.6	Deleting a Role	113
Chapter 5 SOAP - Repository Web Service		115
5.1	Request and Operation Result	115
5.2	List Operation	117
5.3	Get Operation	119
5.4	Put Operation	124
5.5	Delete Operation	125
5.6	Move Operation	126
5.7	Copy Operation	126
5.8	runReport Operation	127
5.8.1	Report Output	128
5.8.2	Report Locales	128
5.9	Errors	129
5.10	Implementation Suggestions	130
Chapter 6 SOAP - Report Scheduling Web Service		131
6.1	Types Defined in the WSDL	131
6.2	Operations in the Scheduling Service	133
6.2.1	Operation Descriptions	133
6.2.2	Example Request and Operation Result	134
6.3	Java Client Classes	136
Chapter 7 SOAP - Domain Web Service		137
7.1	Types Defined in the WSDL	137
7.2	Operations in the Domain Service	137
7.2.1	The getDomainMetaData Operation	138
7.2.2	The executeDomainQuery Operation	142
7.2.3	Java Client Classes	144
Chapter 8 SOAP - Web Services for Administration		145
8.1	Types Defined in the WSDL	145
8.2	Users and Roles	147
8.2.1	findUsers	147
8.2.2	putUser	148

8.2.3	deleteUser	149
8.2.4	findRoles	149
8.2.5	putRole	149
8.2.6	updateRoleName	150
8.2.7	deleteRole	151
8.3	Organizations/Tenants	151
8.3.1	getTenant	151
8.3.2	getSubTenantList	151
8.3.3	putTenant	152
8.3.4	deleteTenant	152
8.4	Permissions	152
8.4.1	getPermissionsForObject	153
8.4.2	putPermission	153
8.4.3	deletePermission	154
8.5	Related Files	154
Appendix A ResourceDescriptor API Constants		155
Glossary		159

CHAPTER 1 INTRODUCTION



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

This document describes the JasperReports Server's web services that allow client applications to interact with the server programmatically.

There are two different Application Programming Interfaces (APIs):

- REST (REpresentational State Transfer) – The RESTful interface depends on the standard methods provided by HTTP: GET, PUT, POST, and DELETE. This interface is new and the API is still expanding.
- SOAP (Simple Object Access Protocol) – The SOAP interface sends and receives XML documents to process requests and provide results. This interface is still supported but is not enhanced with new features.

In order to describe the contents of resources in the repository, both REST and SOAP web services use a custom XML format called a `resourceDescriptor`. When the client requests information about resources, the server responds with lists of resource descriptors, and when the client creates or modifies a resource, it must send a well-formed `resourceDescriptor` that describes the resource. Newer RESTful APIs also support JSON (JavaScript Object Notation) objects.

This chapter contains the following sections:

- [REST Web Services Overview](#)
- [REST Authentication](#)
- [REST Server Information](#)
- [SOAP Web Services Overview](#)
- [SOAP Authentication](#)
- [Syntax of resourceDescriptor](#)

1.1 REST Web Services Overview

The RESTful interface of JasperReports Server responds to HTTP requests from client applications, in particular:

- GET to list, search and acquire information about repository resources.
- PUT to create new resources and execute reports.
- POST to modify resources.
- DELETE to remove resources.

In order to introduce new features and keep backwards compatibility, Jaspersoft has introduced a second RESTful API using the rest_v2 URL.

By default, the REST web services are available at the following URLs, where <host> is the name of the computer hosting JasperReports Server and <port> is the port you specified during installation. However, not all methods are supported on every service:

Table 1-1 REST - Web Services and URLs

Edition	Web Service	URLs
Commercial Editions	Login (optional)	<a href="http://<host>:<port>/jasperserver-pro/rest/GetEncryptionKey">http://<host>:<port>/jasperserver-pro/rest/GetEncryptionKey <a href="http://<host>:<port>/jasperserver-pro/rest/login">http://<host>:<port>/jasperserver-pro/rest/login
	Repository	<a href="http://<host>:<port>/jasperserver-pro/rest/resources">http://<host>:<port>/jasperserver-pro/rest/resources <a href="http://<host>:<port>/jasperserver-pro/rest/resource">http://<host>:<port>/jasperserver-pro/rest/resource <a href="http://<host>:<port>/jasperserver-pro/rest/permission">http://<host>:<port>/jasperserver-pro/rest/permission <a href="http://<host>:<port>/jasperserver-pro/rest_v2/permissions">http://<host>:<port>/jasperserver-pro/rest_v2/permissions <a href="http://<host>:<port>/jasperserver-pro/rest_v2/export">http://<host>:<port>/jasperserver-pro/rest_v2/export <a href="http://<host>:<port>/jasperserver-pro/rest_v2/import">http://<host>:<port>/jasperserver-pro/rest_v2/import
	Reports	<a href="http://<host>:<port>/jasperserver-pro/rest/report">http://<host>:<port>/jasperserver-pro/rest/report <a href="http://<host>:<port>/jasperserver-pro/rest_v2/reports">http://<host>:<port>/jasperserver-pro/rest_v2/reports <a href="http://<host>:<port>/jasperserver-pro/rest_v2/reports/URI/inputControls">http://<host>:<port>/jasperserver-pro/rest_v2/reports/URI/inputControls <a href="http://<host>:<port>/jasperserver-pro/rest_v2/reports/URI/options">http://<host>:<port>/jasperserver-pro/rest_v2/reports/URI/options <a href="http://<host>:<port>/jasperserver-pro/rest/jobsummary">http://<host>:<port>/jasperserver-pro/rest/jobsummary <a href="http://<host>:<port>/jasperserver-pro/rest/job">http://<host>:<port>/jasperserver-pro/rest/job <a href="http://<host>:<port>/jasperserver-pro/rest_v2/jobs">http://<host>:<port>/jasperserver-pro/rest_v2/jobs <a href="http://<host>:<port>/jasperserver-pro/rest_v2/queryExecutor">http://<host>:<port>/jasperserver-pro/rest_v2/queryExecutor
	Administration without organizations	<a href="http://<host>:<port>/jasperserver-pro/rest/user">http://<host>:<port>/jasperserver-pro/rest/user <a href="http://<host>:<port>/jasperserver-pro/rest_v2/users">http://<host>:<port>/jasperserver-pro/rest_v2/users <a href="http://<host>:<port>/jasperserver-pro/rest/attribute">http://<host>:<port>/jasperserver-pro/rest/attribute <a href="http://<host>:<port>/jasperserver-pro/rest_v2/users/userID/attributes">http://<host>:<port>/jasperserver-pro/rest_v2/users/userID/attributes <a href="http://<host>:<port>/jasperserver-pro/rest/role">http://<host>:<port>/jasperserver-pro/rest/role <a href="http://<host>:<port>/jasperserver-pro/rest_v2/roles">http://<host>:<port>/jasperserver-pro/rest_v2/roles
	Administration with organizations	<a href="http://<host>:<port>/jasperserver-pro/rest/organization">http://<host>:<port>/jasperserver-pro/rest/organization <a href="http://<host>:<port>/jasperserver-pro/rest_v2/organizations">http://<host>:<port>/jasperserver-pro/rest_v2/organizations <a href="http://<host>:<port>/jasperserver-pro/rest/user">http://<host>:<port>/jasperserver-pro/rest/user <a href="http://<host>:<port>/jasperserver-pro/rest_v2/organizations/orgID/users">http://<host>:<port>/jasperserver-pro/rest_v2/organizations/orgID/users <a href="http://<host>:<port>/jasperserver-pro/rest/attribute">http://<host>:<port>/jasperserver-pro/rest/attribute <a href="http://<host>:<port>/jasperserver-pro/rest_v2/organizations/orgID/users/userID/attributes">http://<host>:<port>/jasperserver-pro/rest_v2/organizations/orgID/users/userID/attributes <a href="http://<host>:<port>/jasperserver-pro/rest/role">http://<host>:<port>/jasperserver-pro/rest/role <a href="http://<host>:<port>/jasperserver-pro/rest_v2/organizations/orgID/roles">http://<host>:<port>/jasperserver-pro/rest_v2/organizations/orgID/roles <a href="http://<host>:<port>/jasperserver-pro/rest_v2/serverInfo">http://<host>:<port>/jasperserver-pro/rest_v2/serverInfo

Table 1-1 REST - Web Services and URLs, continued

Edition	Web Service	URLs
Community Project	Login (optional)	<a href="http://<host>:<port>/jasperserver/rest/GetEncryptionKey">http://<host>:<port>/jasperserver/rest/GetEncryptionKey <a href="http://<host>:<port>/jasperserver/rest/login">http://<host>:<port>/jasperserver/rest/login
	Repository	<a href="http://<host>:<port>/jasperserver/rest/resources">http://<host>:<port>/jasperserver/rest/resources <a href="http://<host>:<port>/jasperserver/rest/resource">http://<host>:<port>/jasperserver/rest/resource <a href="http://<host>:<port>/jasperserver/rest/permission">http://<host>:<port>/jasperserver/rest/permission <a href="http://<host>:<port>/jasperserver/rest_v2/permissions">http://<host>:<port>/jasperserver/rest_v2/permissions <a href="http://<host>:<port>/jasperserver/rest_v2/export/">http://<host>:<port>/jasperserver/rest_v2/export/ <a href="http://<host>:<port>/jasperserver/rest_v2/import">http://<host>:<port>/jasperserver/rest_v2/import
	Reports	<a href="http://<host>:<port>/jasperserver/rest/report">http://<host>:<port>/jasperserver/rest/report <a href="http://<host>:<port>/jasperserver/rest_v2/reports">http://<host>:<port>/jasperserver/rest_v2/reports <a href="http://<host>:<port>/jasperserver/rest_v2/reports/URI/inputControls">http://<host>:<port>/jasperserver/rest_v2/reports/URI/inputControls <a href="http://<host>:<port>/jasperserver/rest_v2/reports/URI/options">http://<host>:<port>/jasperserver/rest_v2/reports/URI/options <a href="http://<host>:<port>/jasperserver/rest/jobsummary">http://<host>:<port>/jasperserver/rest/jobsummary <a href="http://<host>:<port>/jasperserver/rest/job">http://<host>:<port>/jasperserver/rest/job <a href="http://<host>:<port>/jasperserver/rest_v2/jobs">http://<host>:<port>/jasperserver/rest_v2/jobs
	Administration	<a href="http://<host>:<port>/jasperserver/rest/user">http://<host>:<port>/jasperserver/rest/user <a href="http://<host>:<port>/jasperserver/rest_v2/users">http://<host>:<port>/jasperserver/rest_v2/users <a href="http://<host>:<port>/jasperserver/rest/attribute">http://<host>:<port>/jasperserver/rest/attribute <a href="http://<host>:<port>/jasperserver/rest_v2/users/userID/attributes">http://<host>:<port>/jasperserver/rest_v2/users/userID/attributes <a href="http://<host>:<port>/jasperserver/rest/role">http://<host>:<port>/jasperserver/rest/role <a href="http://<host>:<port>/jasperserver/rest_v2/roles">http://<host>:<port>/jasperserver/rest_v2/roles <a href="http://<host>:<port>/jasperserver/rest_v2/serverInfo">http://<host>:<port>/jasperserver/rest_v2/serverInfo



The context name (by default jasperserver or jasperserver-pro) may also depend on the specific installation of JasperReports Server.

As with any RESTful service, the URLs usually include a path to the resource being acted upon, as well as any arguments that are accepted by the method. For example, to search for input control resources in the /datatypes folder, your application would send the following HTTP request:

```
GET http://<host>:<port>/jasperserver-pro/rest/resources/datatypes?type=inputControl
```

The reference chapters in this book give the full description of the methods supported by each URL, the path or resource expected for each method, and the arguments that are required or optional. The description of each method includes a sample of the return value.

Applications may receive the machine-readable XML description of all supported REST v2 services in Web Application Description Language (WADL) at the following URL:

```
http://<host>:<port>/jasperserver-pro/rest_v2/application.wadl
```

JasperReports Server REST services return standard HTTP status codes. In case of an error, a detailed message may be present in the body in form of plain text. Client error codes are of type 4xx, while server errors are of type 5xx. The following table lists all the standard HTTP codes.

Table 1-2 REST - HTTP Return Codes

Success Messages		Client Error		Server Errors	
Code	Message	Code	Message	Code	Message
100	Continue	400	Bad Request	500	Internal Server Error
101	Switching Protocols	401	Unauthorized	501	Not Implemented
200	OK	402	Payment Required	502	Bad Gateway
201	Created	403	Forbidden	503	Service Unavailable
202	Accepted	404	Not Found	504	Gateway Time-out
203	Non-Authoritative Information	405	Method Not Allowed	505	HTTP Version Not Supported
204	No Content	406	Not Acceptable		
205	Reset Content	407	Proxy Authentication Required		
206	Partial Content	408	Request Time-out		
300	Multiple Choices	409	Conflict		
301	Moved Permanently	410	Gone		
302	Found	411	Length Required		
303	See Other	412	Precondition Failed		
304	Not Modified	413	Request Entity Too Large		
305	Use Proxy	414	Request-URI Too Large		
307	Temporary Redirect	415	Unsupported Media Type		
		416	Requested Range Not Satisfiable		
		417	Expectation Failed		

1.2 REST Authentication

When using web services, the calling application must provide a valid user ID and password to JasperReports Server. The REST web services in JasperReports Server support two types of authentication:

- HTTP Basic Authentication, where the user ID and password are sent in the header with every request. Basic Authentication with REST is the same as described in section 1.5, “[SOAP Authentication](#),” on page 16.
- The special login service that allows authentication using a POST request to create a session and return a session ID that is used with subsequent requests. Use of the login service is optional, and it is useful only when HTTP Basic Authentication does not work.

One example where you may need to use the login service is when the username or password contain UTF-8 characters that may be corrupted by the basic authentication mechanism.

Normally, RESTful implementations do not rely on the use of persistent sessions, such as the login service. However, the JasperReports Server architecture automatically creates user sessions internally, and the login service takes advantage of this. By using the login service, you can avoid making the server verify user credentials for each API call. If your server is configured with external authentication, repeatedly verifying credentials may be a performance issue you can avoid with the login service.

1.2.1 Login Encryption

JasperReports Server 4.7 introduced the ability to encrypt plain-text passwords over non-secure HTTP. Encryption does not make passwords more secure, it only prevents them from being readable to humans. For more information about security and how to enable login encryption, see the *JasperReports Server Administrator Guide*.

When login encryption is enabled, passwords in both HTTP Basic Authentication and using the login service must be encrypted by the client. Login encryption has two modes:

- Static key encryption – The server only uses one key that never changes. The client only needs to encrypt the password once and can use it for every web service call.
- Dynamic key encryption – The server changes the encryption key for every session. The client must request the new key and re-encrypt the password before every call using HTTP Basic Authentication or every session using the login service.

The GetEncryptionKey service does not take any arguments or content input.

Method	URL
GET	http://<host>:<port>/jasperserver[-proj]/GetEncryptionKey/
Return Value on Success	Typical Return Values on Failure
200 OK – Body contains a JSON representation of public key: <pre>{ "maxdigits": "131", "e": "10001", "n": "9f8a2dc4baa260a5835fa33ef94c..." }</pre>	200 OK – Body contains {Error: Key generation is off}

To encrypt a password with the server's public key, use the Bouncy Castle provider with the RSA/NONE/NoPadding algorithm.

1.2.2 Login Service

When making a login request, the user ID and password can be pass as URL arguments or as content in the request body:

Method	URL	
POST	http://<host>:<port>/jasperserver[-proj]/rest/login/	
GET	http://<host>:<port>/jasperserver[-proj]/rest/login?<arguments>	
Argument	Type/Value	Description
j_username	Text	The user ID. In commercial editions of the server that implement multiple organizations, the argument must specify the organization ID or alias in the following format: j_username%7Corganization_id (%7C is the character).
j_password?	Text	The user's password. If the server has login encryption enabled, the password must be encrypted as explained in section 1.2.1. The argument is optional but authentication will fail without the password.
Content-Type	Content	
application/x-www-form-urlencoded	j_username=<userID>[%7C<organization_id>]&j_password=<password> Example: j_username=jasperadmin&j_password=jasperadmin or j_username=jasperadmin%7Corganization_1&j_password=jasperadmin	
Return Value on Success	Typical Return Values on Failure	
200 OK – Session ID in cookie (POST only), empty body.	401 Unauthorized – Empty body. 302 – License expired or otherwise not valid.	

The login service has several uses:

- POST method – Applications should use the POST method, because it returns the session cookie to use in future requests.
- GET method – Developers can test the login service and the user credentials from a browser, which uses the GET method.
- Credentials in arguments – When testing the login service in a browser, credentials are passed as arguments in the URL:
`http://<host>:<port>/jasperserver[-pro]/rest/login?j_username=<userID>[%7C<organization_id>]
&j_password=<password>`
- Credentials in content – When using the POST method, credentials can either be sent in the URL arguments as shown above, or sent in the content of the request, as shown in the second example below.

The following example shows the HTTP request and response when testing the login service in a browser. In this case, the user credentials are passed as arguments and the browser sends a GET request. Because the GET request is meant only for testing, it does not return a cookie with the session ID.

```
GET /jasperserver/rest/login?j_username=jasperadmin&j_password=jasperadmin HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.0; rv:5.0) Gecko/20100101 Firefox/5.0
Connection: keep-alive

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Pragma: No-cache
Cache-Control: no-cache
Expires: Wed, 31 Dec 1969 16:00:00 PST
Content-Length: 0
Date: Fri, 19 Aug 2011 00:52:48 GMT
```

The following example shows the content of a POST request where the credentials are passed in the content.

```
POST /jasperserver/rest/login HTTP/1.1
User-Agent: Jakarta Commons-HttpClient/3.1
Host: localhost:8080
Content-Length: 45
Content-Type: application/x-www-form-urlencoded

j_username=jasperadmin&j_password=jasperadmin

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=52E79BCEE51381DF32637EC69AD698AE; Path=/jasperserver
Content-Length: 0
Date: Fri, 19 Aug 2011 01:52:48 GMT
```

For optimal performance, the session ID from the cookie should be used to keep the session open. To do this, include the cookie in future requests to the other RESTful services. For example, given the response to the POST request above, future requests to the repository services should include the following line in the header:

```
Cookie: $Version=0; JSESSIONID=52E79BCEE51381DF32637EC69AD698AE; $Path=/jasperserver
```

However, maintaining a session with cookies is not mandatory, and your application can use any combination of session cookie, HTTP Basic Authentication, or both.

1.3 REST Server Information

Use the following service to verify the server information, the same as the **About JasperReports Server** link in the user interface.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo
Options	
accept: application/xml accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – Body described below.	This request should always succeed when the server is running.

The server returns a structure containing the information in the requested format, XML or JSON:

<pre><serverInfo> <build>20121029_1532</build> <edition>PRO</edition> <editionName>Enterprise</editionName> <expiration/> <features>Fusion AHD EXP DB AUD ANA MT</features> <licenseType>Commercial</licenseType> <version>5.0.0</version> </serverInfo></pre>	<pre>{ "version" : "5.0.0", "edition" : "PRO", "editionName" : "Enterprise", "licenseType" : "Commercial", "build" : "20121029_1532", "expiration" : "", "features" : "Fusion AHD EXP DB AUD ANA MT "</pre>
--	---

You can access each value separately with the following URLs:

Method	URL
GET	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/version">http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/version <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/edition">http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/edition <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/editionName">http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/editionName <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/build">http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/build <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/licenseType">http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/licenseType <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/features">http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/features <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/expiration">http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo/expiration
Return Value on Success	Typical Return Values on Failure
200 OK – The requested value.	These requests should always succeed when the server is running.

1.4 SOAP Web Services Overview

By default, the SOAP web services are available at the following URLs, where `<host>` is the name of the computer hosting JasperReports Server and `<port>` is the port you specified during installation:

Table 1-3 SOAP - Web Services and URLs

Edition	Web Service	URL
Community Project	Repository	http://<host>:<port>/jasperserver/services/repository
	Scheduling	http://<host>:<port>/jasperserver/services/ReportScheduler
	Administration	http://<host>:<port>/jasperserver/services/UserAndRoleManagementService
Commercial Editions	Repository	http://<host>:<port>/jasperserver-pro/services/repository
	Scheduling	http://<host>:<port>/jasperserver-pro/services/ReportScheduler
	Domains	http://<host>:<port>/jasperserver-pro/services/DomainServices
	Administration	http://<host>:<port>/jasperserver-pro/services/UserAndRoleManagementService



The context name (by default `jasperserver` or `jasperserver-pro`) may also depend on the specific installation of JasperReports Server.

The web services take as input an XML document (the request) and return another XML document (the operation result). Because they use XML, the web services provide easy, natural integration with the most common programming languages and environments.

Jaspersoft provides two complete sample applications that demonstrate the SOAP web service: a simple J2EE (Java 2 Enterprise Edition) web application and the same application written in PHP (PHP Hypertext Preprocessor).

The SOAP web services often refer a namespace with the value of `http://www.jasperforge.org/jasperserver/ws` namespace. This namespace is only an identifier; it is not intended to be a valid URL.

1.5 SOAP Authentication

The calling application must supply a valid user and password with HTTP Basic Authentication to access the web services. In basic authentication, the user ID and password are concatenated with a colon (`:`) and Base64 encoded in the HTTP request header. Usually, your client library does this for you. For example, the administrator's default credentials are `jasperadmin:jasperadmin`, which is encoded as follows:

```
Authorization: Basic amFzcGVyYWRtaW46amFzcGVyYWRtaW4=
```

The web services accept the same accounts and credentials as the JasperReports Server user interface.

- If there is only one organization, such as in the JasperReports Server default installation, you should specify the user name only: `WSUser`. For example, `jasperadmin`.
- In deployments with multiple organizations, the organization ID or alias must be added, in the form `WSUser|TenantId` or `WSUser|TenantAlias`. For example, you could use `jasperadmin|organization_1` (`WSUser|TenantId`) or `jasperadmin|CanadaBranch` (`WSUser|TenantAlias`).
- See section 8.3, “Organizations/Tenants,” on page 151, for explanations of `WSUser`, `TenantId`, and `TenantAlias`.

If login encryption is enabled in your server, you must encrypt the password before base64-encoding it with the username. To encode the password, see section 1.2.1, “Login Encryption,” on page 13.

To simplify the development of web services in Java, Jaspersoft provides a set of helper classes, including a ready-to-use client that can make it easier to integrate an external application with JasperReports Server, be it web- or desktop-based. These classes include an object model that represents resources and creates requests and operation results, along with a Marshaller

and an Unmarshaller class to quickly move between XML and the Java object model. The presentation of each service includes code samples that show how to use these classes.

1.6 Syntax of resourceDescriptor

Resources (such as reports, images, queries, and content resources) are stored in a repository, which is organized like a file system, with a root and a hierarchical set of folders. Each object in the repository is considered a resource: a folder is a resource of type folder, a JRXML resource is a resource of type file, just as images and JAR files are of type file. Some resources are more abstract, such as connection definitions and an input controls. The repository web services operates on all resources.

1.6.1 Overview

A resource is identified by:

- A name.
- A label.
- A unique Uniform Resource Identifier (URI) that defines the location of the resource in the repository. A URI is similar to a Unix path (for example, /reports/samples/AllAccounts).

A resource can have a set of properties (depending on its type) and a set of children resources.

The resource descriptor is a complex structure that transfers data regarding a specific resource between the server and the client. A request can include only one resource descriptor. Often, the request only includes a small portion of the entire resource descriptor definition: the part that describes the specific details of the resource in question.

For example, when a `resourceDescriptor` is used as an input parameter in a request document (for example, to specify a folder to list or a file to download), the descriptor includes only a small portion of the entire resource descriptor definition: the part that describes the specific resource details in question. In many cases, the only information required to identify a resource in the repository is its `wsType`, `name`, and `URI`.

The resource descriptors that the server sends are completely populated with all the data about the resources being described.

A `resourceDescriptor` tag is defined by the following DTD (Document Type Definition):

```
<!ELEMENT resourceDescriptor (label, description?, resourceProperty*,
resourceDescriptor*, parameter*)>
<!ATTLIST resourceDescriptor
  name CDATA #REQUIRED
  wsType CDATA #REQUIRED
  uriString CDATA #REQUIRED
  isNew ( true | false ) false
>
<!ELEMENT resourceProperty (value?, resourceProperty*)>
<!ATTLIST resourceProperty
  name CDATA #REQUIRED
>
<!ELEMENT value (#PCDATA)>
<!ELEMENT parameter (#PCDATA)>
<!ATTLIST parameter
  name CDATA #REQUIRED
  isListItem ( true | false ) false
>
```

The following sections describe the DTD:

- [wsType Attribute](#)

- [isNew Attribute](#)
- [Resource Descriptor Parameters](#)
- [Examples of resourceDescriptor](#)

1.6.2 wsType Attribute

The `wsType` attribute defines the nature of the resource. The possible values for this attribute are:

Table 1-4 Values for wsType

wsType Value	Description
aws	Amazon Web Services data source
bean	Data source of type Spring bean
contentResource	The output of a report
datasource	Generic data source – This type is normally used for a data source ReportUnit child resource when it is not defined locally to the ReportUnit.
dataType	Datatype (used with the input controls)
folder	Folder
font	Font file (normally a True Type font)
img	Image file
inputControl	Input control
jar	JAR file
jdbc	Data source of type JDBC
jndi	Data source of type JNDI
jrxml	JRXML source file
lov	List of values (used with input controls)
olapMondrianCon	OLAP Mondrian connection. A direct connection to an OLAP source.
olapMondrianSchema	OLAP Mondrian Schema
olapXmlaCon	OLAP XMLA connection. A remote connection to an OLAP source.
prop	Resource bundle file (ending with .properties) for specific reports
query	Query used to retrieve data from a data source
reference	Reference to another resource. References are only present in report units
reportUnit	A complete report that can be run in JasperReports Server
virtual	Virtual data source – This type has a child ResourceDescriptor for each data source contained in the virtual data source.
xmlaConnection	XML/A Connection

For all the other resource types found in the repository, the repository web service sets the attribute `wsType` to UNKNOWN.

1.6.3 isNew Attribute

The `isNew` attribute is used with the `put` operation to indicate whether the resource being uploaded is new or replaces an existing resource in the repository.

1.6.4 Resource Descriptor Parameters

A resource descriptor can contain one or more parameters: they do not describe the resource; they store the values users select when the `runReport` service is invoked.

A `resourceProperty` is normally a simple pair of a name and a value. The Java class `com.jaspersoft.jasperserver.api.metadata.xml.domain.impl.ResourceDescriptor` contains constants for each property name. For a list of parameter names, see [Appendix A, “ResourceDescriptor API Constants,” on page 155](#). Jaspersoft may add further constants in future releases.

1.6.5 Examples of resourceDescriptor

The following `resourceDescriptor` sample contains a set of simple properties that describe a JDBC connection resource:

```
<resourceDescriptor name="JServerJdbcDS" wsType="jdbc"
    uriString="/datasources/JServerJdbcDS" isNew="false">
  <label>JServer Jdbc data source</label>
  <description>JServer Jdbc data source</description>
  <resourceProperty name="PROP_PARENT_FOLDER">
    <value>/datasources</value>
  </resourceProperty>
  <resourceProperty name="PROP_VERSION">
    <value>0</value>
  </resourceProperty>
  <resourceProperty name="PROP_DATASOURCE_DRIVER_CLASS">
    <value>com.mysql.jdbc.Driver</value>
  </resourceProperty>
  <resourceProperty name="PROP_DATASOURCE_CONNECTION_URL">
    <value>jdbc:mysql://localhost/test?autoReconnect=true</value>
  </resourceProperty>

  <resourceProperty name="PROP_DATASOURCE_USERNAME">
    <value>username</value>
  </resourceProperty>
  <resourceProperty name="PROP_DATASOURCE_PASSWORD">
    <value>password</value>
  </resourceProperty>
</resourceDescriptor>
```

Some properties cannot be represented by a simple value. To accommodate more complicated properties, a `resourceProperty` can recursively contain other `resourceProperties`. This is the case for a List of Values type resource (used to define input controls for report parameters); the list values are contained in the `resourceProperty` named `PROP_LOV` and are represented by sub-`resourceProperties`. For example:

```
<resourceDescriptor name="SampleLOV" wsType="lov" uriString="/datatypes/SampleLOV"
isNew="false">
  <label>Sample List of Values</label>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.ListOfValues
    </value>
  </resourceProperty>
  <resourceProperty name="PROP_PARENT_FOLDER">
    <value>/datatypes</value>
  </resourceProperty>
  <resourceProperty name="PROP_VERSION">
    <value>-1</value>
  </resourceProperty>
  <resourceProperty name="PROP_HAS_DATA">
    <value>>false</value>
  </resourceProperty>
  <resourceProperty name="PROP_IS_REFERENCE">
    <value>>false</value>
  </resourceProperty>

  <resourceProperty name="PROP_LOV">
    <resourceProperty name="US">
      <value>United States</value>
    </resourceProperty>
    <resourceProperty name="CA">
      <value>Canada</value>
    </resourceProperty><resourceProperty name="IN">
      <value>India</value>
    </resourceProperty>
    <resourceProperty name="IT">
      <value>Italy</value>
    </resourceProperty>
    <resourceProperty name="DE">
      <value>Germany</value>
    </resourceProperty>
    <resourceProperty name="RO">
      <value>Romania</value>
    </resourceProperty>
  </resourceProperty>
</resourceDescriptor>
```

This example defined a list of countries. Notice that, for each list item, the `resourceProperty` name represents the item value, and the `resourceProperty` value contains the item label.

CHAPTER 2 REST - REPOSITORY WEB SERVICES

This chapter documents the HTTP methods (sometimes called verbs) and parameters for each of these requests. In every case, you specify the folder, resource, or report to be acted up by adding its repository URI to the request URL. This chapter uses the following notation:

```
http://<host>:<port>/jasperserver[-pro]/rest/<service>/path/to/object
```

Arguments are passed in the URL with the conventional syntax:

```
http://<host>:<port>/jasperserver[-pro]/rest/<service>/path/to/object?<arg1>=<value>&<arg2>=<value>&...
```

The documentation for each method gives the list of arguments it supports. Optional arguments are listed with a question mark after the name, for example `<arg2>?`. Arguments that are not marked optional are mandatory and must be included in the URL with a valid value.

For authentication using the REST web services, see section [1.2, “REST Authentication,” on page 12](#).

The RESTful repository services gives responses that contain the same XML data structure that are used in the SOAP repository web service. These data structures are shown as examples throughout the chapter and documented in section [1.6, “Syntax of resourceDescriptor,” on page 17](#), with reference material in [Appendix A, “ResourceDescriptor API Constants,” on page 155](#).

This chapter includes the following sections:

- [The resources Service](#)
- [The resource Service](#)
- [Working with Dashboards](#)
- [Working with Virtual Data Sources](#)
- [Working with Domains](#)
- [The permission Service](#)
- [The v2/permissions Service](#)
- [The v2/export Service](#)
- [The v2/import Service](#)

2.1 The resources Service

The resources service lets you browse or search the repository. When used without arguments, it gives the list of resources in the folder specified in the URL. With the arguments, you can search for terms in the resource names or descriptions, search for all resources of a given type, and specify whether to search in subfolders. This service is similar to the `list` operation in the SOAP web services.

The resources service is a read only service. Requests for PUT, POST, and DELETE operations receive the error 405, method not allowed.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest/resources/path/to/folder/	
Argument	Type/Value	Description
q?	String	Match only resources having the specified text in the name or description
type?	wsType	Match only resources of the given type. Valid types are listed in Table 1-4, “Values for wsType,” on page 18 , for example: datasource, reportUnit, img, folder.
recursive?	0 1	Search for resources recursively and not only in the specified folder. This parameter is used only when a search criteria is specified (either q or type). When not specified, the default is 0, meaning only in the specified folder. Note that searching recursively in the whole repository may cause performance issues, because the number of resources returned may be huge.
limit?	Integer >= 0	Maximum number of items returned to the client. The default is 0, meaning no limit.
Return Value on Success		Typical Return Values on Failure
200 OK – The body is XML containing the list of resourceDescriptors.		404 Not Found – The specified URI is not found in the repository.

The XML content in the result consists of resourceDescriptors described in section [1.6, “Syntax of resourceDescriptor,” on page 17](#). However, the list may be empty in the following conditions:

- If the specified URI is a resource instead of a folder.
- If the folder is empty or the search returns no results.

The following example shows the request to list the resources in the /reports folder:

```
GET /jasperserver/rest/resources/reports HTTP/1.1
User-Agent: Jakarta Commons-HttpClient/3.1
Authorization: Basic amFzcGVyYWRtaW46amFzcGVyYWRtaW4=
Host: localhost:8080
Cookie: $Version=0; JSESSIONID=6854BF45EC89F3D3CE3E6F4FD6FF1BBD; $Path=/jasperserver
```

Because the example is not a recursive search, it simply returns the contents of the folder, in this case a subfolder and a report:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Pragma: No-cache
Cache-Control: no-cache
Expires: Thu, 01 Jan 1970 01:00:00 CET
Content-Length: 1518
Date: Fri, 24 Jun 2011 12:09:45 GMT
```

```

<resourceDescriptors>
  <resourceDescriptor name="samples" wsType="folder" uriString="/reports/samples"
    isNew="false">
    <label>Samples</label>
    <description>Samples</description>
    <creationDate>1302268917000</creationDate>
    <resourceProperty name="PROP_HAS_DATA"><value>false</value></resourceProperty>
    <resourceProperty name="PROP_RESOURCE_TYPE">
      <value>com.jaspersoft.jasperserver.api.metadata.common.domain.Folder</value>
    </resourceProperty>
    <resourceProperty name="PROP_PARENT_FOLDER"><value>/reports</value>
    </resourceProperty>
    <resourceProperty name="PROP_SECURITY_PERMISSION_MASK"><value>31</value>
    </resourceProperty>
    <resourceProperty name="PROP_VERSION"><value>0</value></resourceProperty>
  </resourceDescriptor>

  <resourceDescriptor name="test" wsType="reportUnit" uriString="/reports/test"
    isNew="false">
    <label>A test</label>
    <creationDate>1303206124000</creationDate>
    <resourceProperty name="PROP_RESOURCE_TYPE">
      <value>com.jaspersoft.jasperserver.api.metadata.jasperreports.domain.
        ReportUnit</value>
    </resourceProperty>
    <resourceProperty name="PROP_PARENT_FOLDER"><value>/reports</value>
    </resourceProperty>
    <resourceProperty name="PROP_SECURITY_PERMISSION_MASK"><value>31</value>
    </resourceProperty>
    <resourceProperty name="PROP_VERSION"><value>19</value></resourceProperty>
    <resourceProperty name="PROP_RU_ALWAYS_PROPMT_CONTROLS"><value>true</value>
    </resourceProperty>
    <resourceProperty name="PROP_RU_CONTROLS_LAYOUT"><value>1</value>
    </resourceProperty>
  </resourceDescriptor>
</resourceDescriptors>

```

The following sample request is intended to list all the reports available in the /reports folder and subfolders. The result, not shown, is a long list of resourceDescriptors for reports in the designated folders.

```

GET /jasperserver/rest/resources/reports?type=reportUnit&recursive=1 HTTP/1.1
User-Agent: Jakarta Commons-HttpClient/3.1
Authorization: Basic amFzcGVyYWRTaW46amFzcGVyYWRTaW4=
Host: localhost:8080
Cookie: $Version=0; JSESSIONID=60B573BDC47098E6379FC867B24C5C0E; $Path=/jasperserver

```

2.2 The resource Service

The resource service supports several HTTP methods to view, download, create, and modify resources in the repository.

GET is used to show the information about a specific resource. Getting a resource can serve several purposes:

- In the case of JasperReports, also known as report units, this service returns the structure of the JasperReport, including resourceDescriptors for any linked resources.
- For resources that contain files, specifying the `fileData=true` argument downloads the file content.
- Specifying a query-based input control with arguments for running the query returns the dynamic values for the control.



A new service is also available to interact with report options. See section 3.5, “The v2/options Service,” on page 67.

Method	URL	
GET	http://<host>:<port>/jasperserver[-proj]/rest/resource/path/to/resource/?<arguments>	
Argument	Type/Value	Description
fileData?	Boolean	For resources that contain a file, set this argument to true to download the file. When not specified, this argument is false by default and the method returns the description of the resource.
IC_GET_QUERY_DATA?	String	Used to get the items to fill an input control which subtend a query resource. The value of this parameter must be the URI of the data source to use to execute the query. Set the null string to use the default data source.
P_<param name>?	String	If the IC_GET_QUERY_DATA is specified, one or more parameters can be specified to be used in the query: <ul style="list-style-type: none"> • Use the "P_" prefix for single values. • Use the "PL_" prefix for list of values.
PL_<param name>?	String	
Return Value on Success		Typical Return Values on Failure
200 OK – The body is either: <ul style="list-style-type: none"> • XML giving the resourceDescriptors that make up the resource, including nested descriptors. • The native content of the specified file. 		404 Not Found – When the specified resource URI is not found in the repository

The GET method returns the structure and definition of resources in the repository, and using that information can be used to download any files attached to the resources. Resources are defined through `resourceDescriptor` tags in XML.

The following example shows the resource descriptor of a folder:

```
<resourceDescriptor name="datasources" wsType="folder" uriString="/datasources"
    isNew="false">
  <label>Data Sources</label>
  <description>Data Sources used by reports</description>
  <creationDate>1317838605320</creationDate>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.Folder</value>
  </resourceProperty>
  <resourceProperty name="PROP_PARENT_FOLDER"><value>/</value></resourceProperty>
  <resourceProperty name="PROP_VERSION"><value>0</value></resourceProperty>
  <resourceProperty name="PROP_HAS_DATA"><value>>false</value></resourceProperty>
</resourceDescriptor>
```


The following example shows the resource descriptor of a data source. The various `resourceProperty` tags define the properties of the data source, specific to the JNDI type:

```
<resourceDescriptor name="SugarCRMDataSourceJNDI" wsType="jndi"
  uriString="/analysis/datasources/SugarCRMDataSourceJNDI" isNew="false">
  <label>SugarCRM Data Source JNDI</label>
  <description>SugarCRM Data Source JNDI</description>
  <creationDate>1318380229907</creationDate>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.jasperreports.domain.
      JndiJdbcReportDataSource</value>
  </resourceProperty>
  <resourceProperty name="PROP_PARENT_FOLDER"><value>/analysis/datasources</value>
  </resourceProperty>
  <resourceProperty name="PROP_VERSION"><value>0</value></resourceProperty>
  <resourceProperty name="PROP_DATASOURCE_JNDI_NAME"><value>jdbc/sugarcrm</value>
  </resourceProperty>
</resourceDescriptor>
```

The following example shows the resource descriptor of a query resource, with properties for the query string and query language:

```
<resourceDescriptor name="CustomerCityQuery" wsType="query"
  uriString="/datatypes/CustomerCityQuery" isNew="false">
  <label>Customer City Query</label>
  <description>Retrieves names of all customers' home cities</description>
  <creationDate>1318380317602</creationDate>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.Query</value>
  </resourceProperty><resourceProperty name="PROP_PARENT_FOLDER">
    <value>/JUNIT_NEW_FOLDER</value></resourceProperty>
  <resourceProperty name="PROP_VERSION"><value>0</value></resourceProperty>
  <resourceProperty name="PROP_HAS_DATA"><value>>false</value></resourceProperty>
  <resourceProperty name="PROP_IS_REFERENCE"><value>>false</value></resourceProperty>
  <resourceProperty name="PROP_QUERY">
    <value>select distinct customer.city from customer</value></resourceProperty>
  <resourceProperty name="PROP_QUERY_LANGUAGE"><value>sql</value></resourceProperty>
</resourceDescriptor>
```

2.2.1 Requesting the Contents of a JasperReport

A `JasperReport` is a complex resource that contains many parts such as a data source, input controls, and file resources. These can be either references to other resources in the repository or resources that are fully defined internally to the report.

In the following example, a simple request gives the contents of a `JasperReport`:

```
GET http://localhost:8080/jasperserver/rest/resource/reports/samples/AllAccounts
```

The following response in this example shows the content of the `AllAccounts` report:

- The `reportUnit`, which is the container for all the resources of the report.
- The data source, which is an external link to a data source in the repository.
- The main JRXML, which is a file defined internally to this resource.
- Two image files, one of which is defined internally to this resource, the other references a file resource in the repository.

The structure of the JasperReport is defined through nested `resourceDescriptor` tags in XML. In the nested descriptor for each file that is part of the JasperReport, we can find its URI and use `fileData=true` to retrieve that file:

```
<resourceDescriptor name="AllAccounts" wsType="reportUnit"
    uriString="/reports/samples/AllAccounts" isNew="false">
  <label>Accounts Report</label>
  <description>All Accounts Report</description>
  <creationDate>1302268918000</creationDate>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.jasperreports.domain.
      ReportUnit</value>
  </resourceProperty>
  <resourceProperty name="PROP_PARENT_FOLDER"><value>/reports/samples</value>
  </resourceProperty>
  <resourceProperty name="PROP_VERSION"><value>2</value></resourceProperty>
  <resourceProperty name="PROP_RU_ALWAYS_PROPMT_CONTROLS"><value>>false</value>
  </resourceProperty>
  <resourceProperty name="PROP_RU_CONTROLS_LAYOUT"><value>1</value>
  </resourceProperty>

  <resourceDescriptor name="" wsType="datasource" uriString="" isNew="false">
    <label>null</label>
    <resourceProperty name="PROP_REFERENCE_URI">
      <value>/datasources/JServerJNDIDS</value>
    </resourceProperty>
    <resourceProperty name="PROP_IS_REFERENCE"><value>>true</value>
    </resourceProperty>
  </resourceDescriptor>

  <resourceDescriptor name="AllAccountsReport" wsType="jrxml" uriString="/reports/
    samples/AllAccounts_files/AllAccountsReport" isNew="false">
    <label>All Accounts Jasper Report</label>
    <description>All Accounts Jasper Report</description>
    <creationDate>1302268918000</creationDate>
    <resourceProperty name="PROP_RESOURCE_TYPE">
      <value>com.jaspersoft.jasperserver.api.metadata.common.domain.FileResource
        </value>
    </resourceProperty>
    <resourceProperty name="PROP_PARENT_FOLDER">
      <value>/reports/samples/AllAccounts_files</value>
    </resourceProperty>
    <resourceProperty name="PROP_VERSION"><value>2</value></resourceProperty>
    <resourceProperty name="PROP_IS_REFERENCE"><value>>false</value>
    </resourceProperty>
    <resourceProperty name="PROP_HAS_DATA"><value>>true</value></resourceProperty>
    <resourceProperty name="PROP_ATTACHMENT_ID"><value>attachment</value>
    </resourceProperty>
    <resourceProperty name="PROP_RU_IS_MAIN_REPORT"><value>>true</value>
    </resourceProperty>
  </resourceDescriptor>
```

```
<resourceDescriptor name="AllAccounts_Res2" wsType="img" uriString="/reports/
    samples/AllAccounts_files/AllAccounts_Res2" isNew="false">
  <label>AllAccounts_Res2</label>
  <description>AllAccounts_Res2</description>
  <creationDate>1302268918000</creationDate>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.FileResource
    </value>
  </resourceProperty>
  <resourceProperty name="PROP_PARENT_FOLDER">
    <value>/reports/samples/AllAccounts_files</value></resourceProperty>
  <resourceProperty name="PROP_VERSION"><value>0</value></resourceProperty>
  <resourceProperty name="PROP_IS_REFERENCE"><value>>false</value>
  </resourceProperty>
  <resourceProperty name="PROP_HAS_DATA"><value>>true</value></resourceProperty>
  <resourceProperty name="PROP_ATTACHMENT_ID"><value>attachment</value>
  </resourceProperty>
</resourceDescriptor>

<resourceDescriptor name="AllAccounts_Res3" wsType="img" uriString="/reports/
    samples/AllAccounts_files/AllAccounts_Res3" isNew="false">
  <label>AllAccounts_Res3</label>
  <description>AllAccounts_Res3</description>
  <creationDate>1302268918000</creationDate>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.FileResource
    </value></resourceProperty>
  <resourceProperty name="PROP_PARENT_FOLDER">
    <value>/reports/samples/AllAccounts_files</value></resourceProperty>
  <resourceProperty name="PROP_VERSION"><value>0</value></resourceProperty>
  <resourceProperty name="PROP_IS_REFERENCE"><value>>false</value>
  </resourceProperty>
  <resourceProperty name="PROP_HAS_DATA"><value>>true</value></resourceProperty>
  <resourceProperty name="PROP_ATTACHMENT_ID"><value>attachment</value>
  </resourceProperty>
</resourceDescriptor>

<resourceDescriptor name="LogoLink" wsType="reference" uriString="/reports/
    samples/AllAccounts_files/LogoLink" isNew="false">
  <label>LogoLink_label</label>
  <description>LogoLink description</description>
  <creationDate>1302268918000</creationDate>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.FileResource
    </value></resourceProperty>
  <resourceProperty name="PROP_PARENT_FOLDER">
    <value>/reports/samples/AllAccounts_files</value></resourceProperty>
  <resourceProperty name="PROP_VERSION"><value>0</value></resourceProperty>
  <resourceProperty name="PROP_IS_REFERENCE"><value>>true</value>
  </resourceProperty>
  <resourceProperty name="PROP_REFERENCE_URI"><value>/images/JRLogo</value>
  </resourceProperty>
```

```

    <resourceProperty name="PROP_HAS_DATA"><value>true</value></resourceProperty>
    <resourceProperty name="PROP_ATTACHMENT_ID"><value>attachment</value>
  </resourceProperty>
</resourceDescriptor>
</resourceDescriptor>

```

2.2.2 Requesting the Contents of a File Resource

In order to retrieve the contents of a file, first retrieve the descriptor of its enclosing resource to find the resource URI. The resource can either be internal to the report, or referenced from the repository. In either case it has a URI in the repository that we can extract from the parent resource.

In the previous sample, we see the attachments that are internal to the AllAccounts report. There is an image named AllAccounts_Res3 that is not a reference and has data. To download the image, we create the following request to the resource service, giving it the internal URI of the image and `fileData=true`.

```

GET /jasperserver/rest/resource/reports/samples/AllAccounts_files/AllAccounts_Res3?
fileData=true HTTP/1.1
User-Agent: Jakarta Commons-HttpClient/3.1
Authorization: Basic amFzcGVyYWRtaW46amFzcGVyYWRtaW4=
Host: localhost:8080
Cookie: $Version=0; JSESSIONID=6854BF45EC89F3D3CE3E6F4FD6FF1BBD; $Path=/jasperserver

```



Notice that the URI to the file includes the path AllAccounts_files. This is a local path that exists to access the local resources of the report, not a folder that exists in the repository.

In the case of a resource that is referenced in the repository, you can download the contents of the file in the same way. From the resource descriptor in the previous section, we see there is another resource where `PROP_IS_REFERENCE` is true. We can extract the repository URI from the `PROP_REFERENCE_URI` property and use that with `fileData=true` to obtain the image:

```

GET /jasperserver-pro/rest/resource/images/JRLogo?fileData=true HTTP/1.1
User-Agent: Jakarta Commons-HttpClient/3.1
Authorization: Basic amFzcGVyYWRtaW46amFzcGVyYWRtaW4=
Host: localhost:8080
Cookie: $Version=0; JSESSIONID=6854BF45EC89F3D3CE3E6F4FD6FF1BBD; $Path=/jasperserver

```

In both cases, the response contains the binary contents of the file. The response header contains the information to decode it:

```

Content-Disposition: attachment; filename=JRLogo
Content-Type: application/octet-stream
Content-Length: 1491

```



The descriptor does not store the extension nor the format of a file resource. Jaspersoft recommends using the filename *with the file extension* as the resource ID when creating file resources so that the extension is available when downloading the file.

2.2.3 Requesting the Values of a Query-Based Input Control



A newer service is available to interact with input controls, including query-based input controls. See section 3.4, “The `v2/inputControls` Service,” on page 64.

The following sample request specifies a resource that is a query-based input control, and by specifying the appropriate parameters, we can receive the current values. In this case, one of the parameters to the query is a list of two values, USA and Mexico:

```
GET http://localhost:8080/jasperserver/rest/resource/reports/samples/Cascading_multi_select_report_files/
Cascading_state_multi_select?IC_GET_QUERY_DATA=/datasources/JServerJNDIDS&
PL_Country_multi_select=USA&PL_Country_multi_select=Mexico
```

The following response shows the resource descriptor for the requested input control, and it contains extra properties that give all the values that are the results of the query. You can see they are from Mexico and USA. The resource descriptor also includes the nested descriptor for the query that is part of the input control.



If a selection-type input control has a null value, it is given as ~NULL~. If no selection is made, its value is given as ~NOTHING~.

```
<resourceDescriptor name="Cascading_state_multi_select" wsType="inputControl"
  uriString="/reports/samples/Cascading_multi_select_report_files/
  Cascading_state_multi_select" isNew="false">
  <label>Cascading state multi select control</label>
  <description>Cascading state multi select control</description>
  <creationDate>1302268918000</creationDate>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.InputControl
    </value>
  </resourceProperty>
  <resourceProperty name="PROP_PARENT_FOLDER">
    <value>/reports/samples/Cascading_multi_select_report_files</value>
  </resourceProperty>
  <resourceProperty name="PROP_VERSION"><value>0</value></resourceProperty>
  <resourceProperty name="PROP_HAS_DATA"><value>>false</value></resourceProperty>
  <resourceProperty name="PROP_IS_REFERENCE"><value>>false</value></resourceProperty>
  <resourceProperty name="PROP_INPUTCONTROL_IS_MANDATORY"><value>>true</value>
  </resourceProperty>
  <resourceProperty name="PROP_INPUTCONTROL_IS_READONLY"><value>>false</value>
  </resourceProperty>
  <resourceProperty name="PROP_INPUTCONTROL_IS_VISIBLE"><value>>true</value>
  </resourceProperty>
  <resourceProperty name="PROP_INPUTCONTROL_TYPE"><value>7</value>
  </resourceProperty>
  <resourceProperty name="PROP_QUERY_VALUE_COLUMN">
    <value>billing_address_state</value></resourceProperty>
  <resourceProperty name="PROP_QUERY_VISIBLE_COLUMNS">
    <resourceProperty name="PROP_QUERY_VISIBLE_COLUMN_NAME">
      <value>billing_address_country</value></resourceProperty>
    <resourceProperty name="PROP_QUERY_VISIBLE_COLUMN_NAME">
      <value>billing_address_state</value></resourceProperty>
    </resourceProperty>
  <resourceProperty name="PROP_QUERY_DATA">
    <resourceProperty name="PROP_QUERY_DATA_ROW"><value>DF</value>
    <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
      <value>Mexico</value></resourceProperty>
    <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
      <value>DF</value></resourceProperty>
    </resourceProperty>
  </resourceProperty>
```

```

...
<resourceProperty name="PROP_QUERY_DATA_ROW"><value>Zacatecas</value>
  <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
    <value>Mexico</value></resourceProperty>
  <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
    <value>Zacatecas</value></resourceProperty>
</resourceProperty>
<resourceProperty name="PROP_QUERY_DATA_ROW"><value>CA</value>
  <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
    <value>USA</value></resourceProperty>
  <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
    <value>CA</value></resourceProperty>
</resourceProperty>
...
<resourceProperty name="PROP_QUERY_DATA_ROW"><value>WA</value>
  <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
    <value>USA</value></resourceProperty>
  <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
    <value>WA</value></resourceProperty>
</resourceProperty>
</resourceProperty>
<resourceDescriptor name="Cascading_state_query" wsType="query" uriString="/
  reports/samples/Cascading_multi_select_report_files/
  Cascading_state_multi_select_files/Cascading_state_query"
  isNew="false">
  <label>Cascading state query</label>
  <creationDate>1302268918000</creationDate>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.Query</value>
  </resourceProperty>
  <resourceProperty name="PROP_PARENT_FOLDER">
    <value>/reports/samples/Cascading_multi_select_report_files/
      Cascading_state_multi_select_files</value></resourceProperty>
  <resourceProperty name="PROP_VERSION"><value>0</value></resourceProperty>
  <resourceProperty name="PROP_HAS_DATA"><value>>false</value></resourceProperty>
  <resourceProperty name="PROP_IS_REFERENCE"><value>>false</value>
  </resourceProperty>
  <resourceProperty name="PROP_QUERY">
    <value>select distinct billing_address_state, billing_address_country
      from accounts where ${IN, billing_address_country,
        Country_multi_select} order by billing_address_country,
        billing_address_state</value>
  </resourceProperty>
  <resourceProperty name="PROP_QUERY_LANGUAGE"><value>sql</value>
  </resourceProperty>
</resourceDescriptor name="" wsType="datasource" uriString="" isNew="false">
  <label>null</label>
  <resourceProperty name="PROP_REFERENCE_URI">
    <value>/datasources/JServerJNDIDS</value></resourceProperty>
  <resourceProperty name="PROP_IS_REFERENCE"><value>>true</value>
  </resourceProperty>

```

```

    </resourceDescriptor>
  </resourceDescriptor>
</resourceDescriptor>

```

2.2.4 Creating a Resource

The PUT method on the resource service is used to create a new resource. If the resource has one or more file resources, they must be provided using a multipart request.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest/resource/path/to/resource/	
Argument	Type/Value	Description
Resource Descriptor	String	This parameter identifies the part with an XML resource descriptor in a multipart request. This is a required argument when using multipart requests.
Content-Type		Content
multipart/form-data text/plain (in the first part) application/octet-stream (for files)		A well-formed XML resourceDescriptor that fully describes the resource, including any locally defined resources. File resources are uploaded in separate parts.
Return Value on Success		Typical Return Values on Failure
201 Created – The body is XML containing the resourceDescriptor of the resource just created.		An error if the resource cannot be created for some reason. If you have very large files, see section 2.2.5, “Setting the Temporary Upload Directory,” on page 32.

In the following sample request, the URI is the location where we want to create the resource, in this case / (the root), and the content includes the resource descriptor for a new folder called myfolder.

```

PUT /jasperserver/rest/resource/ HTTP/1.1
Content-Length: 473
Content-Type: multipart/form-data; boundary=1afdzzMUQLfSOmu0Pgb2F-nmEnTwWuPf3
Host: localhost:8080
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
Cookie: JSESSIONID=3370EC843B09363C0A8DD09A2D1F21E3
Cookie2: $Version=1

--1afdzzMUQLfSOmu0Pgb2F-nmEnTwWuPf3
Content-Disposition: form-data; name="ResourceDescriptor"
Content-Type: text/plain; charset=US-ASCII
Content-Transfer-Encoding: 8bit

<resourceDescriptor name="myfolder" wsType="folder" uriString="/myfolder"
    isNew="false">
  <label>REST created folder</label>
  <resourceProperty name="PROP_PARENT_FOLDER">
    <value>/</value>
  </resourceProperty>
</resourceDescriptor>

--1afdzzMUQLfSOmu0Pgb2F-nmEnTwWuPf3--

```

Also, the example above shows a multi-part request even though it is only sending the plain-text resource descriptor and not a binary file. Usually, such requests could be sent without multiple parts, and multiple parts are used to send a binary file, for example when creating a report.



When the ResourceDescriptor contains the ResourceProperty PROP_PARENT_FOLDER, that property overrides the path/to/resource given as the URL and determines the location of the new resource.

The response to the PUT request is the complete resource descriptor for the new folder:

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Cache-Control: no-cache
Content-Length: 648
Date: Mon, 01 Aug 2011 14:44:05 GMT

<resourceDescriptor name="myfolder" wsType="folder" uriString="/myfolder"
    isNew="false">
  <label>REST created folder</label>
  <creationDate>1312209845000</creationDate>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.Folder</value>
  </resourceProperty>
  <resourceProperty name="PROP_PARENT_FOLDER">
    <value>/</value>
  </resourceProperty>
  <resourceProperty name="PROP_VERSION">
    <value>0</value>
  </resourceProperty>
  <resourceProperty name="PROP_HAS_DATA">
    <value>>false</value>
  </resourceProperty>
</resourceDescriptor>
```

2.2.5 Setting the Temporary Upload Directory

When you create a resource by uploading a large file, the server stores the file in a temporary location while receiving and processing it. Uploaded files may be huge and cause errors if the local disk is limited or full. If you are having issues when creating resources with large files, set the following property to a directory location with sufficient capacity.

Temporary Upload Directory for Web Services		
Configuration File		
...\\WEB-INF\applicationContext-webservices.xml		
Property	Bean	Description
attachmentsTempFolder	management ServiceImpl	Change this property to an absolute path such as /tmp/jasperserver/axis_attachments or a relative path such as {java.io.tmpdir}/jasperserver/axis_attachments.

2.2.6 Modifying a Resource

The POST method on the resource service is used to modify a resource. If the resource has one or more file resources, they must be provided using a multipart request. A POST operates on the URL of an existing resource, otherwise it is identical to the PUT method for a new resource.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest/resource/path/to/resource/	
Argument	Type/Value	Description
Resource Descriptor	String	This parameter identifies the part with an XML resource descriptor in a multipart request. This is a required argument when using multipart requests.
Content-Type		Content
multipart/form-data text/plain (in the first part) application/octet-stream (for files)		A well-formed XML resourceDescriptor that fully describes the modified resource, including any locally defined resources. File resources are uploaded in separate parts.
Return Value on Success		Typical Return Values on Failure
201 Created – The body is XML containing the resourceDescriptor of the resource just modified.		An error if the resource cannot be modified for some reason.

2.2.7 Copying or Moving a Resource

The POST method on the resource service also has parameters to copy or move a resource. Both folders and individual resources can be copied or moved. The ID of the resource being copied or moved must be unique within the destination folder, otherwise the operation will fail. This implies that copying cannot be used to duplicate a resource within the same folder.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest/resource/path/to/resource/?<argument>	
Argument	Type/Value	Description
copyTo?	/path/to/folder	Destination folder in the repository.
moveTo?	/path/to/folder	Destination folder in the repository.
Return Value on Success		Typical Return Values on Failure
200 OK– The resource was successfully moved or copied.		An error if the resource cannot be moved or copied, for example if a resource with the same ID already exists in the destination folder.

2.2.8 Deleting a Resource

The DELETE method can be used with either a folder or a resource. For the delete to succeed:

- The logged in user must have read-delete, read-write-delete, or administer permission on the folder or resource.
- The resource must not be a dependency of any other resource, for example the data source of a JasperReport. In this case, you must modify or delete the other resource first.
- If the target is a folder, the above requirements must be satisfied for every resource and folder it contains, including any those contained recursively in subfolders to any level.

Method	URL	
DELETE	http://<host>:<port>/jasperserver[-pro]/rest/resource/path/to/resource/	
Return Value on Success		Typical Return Values on Failure
200 OK – The resource was deleted.		404 Not Found – When the specified resource URI is not found in the repository

2.3 Working with Dashboards



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 resource service also gives access to dashboard resources in commercial editions of JasperReports Server. Dashboards are managed as normal resources whose descriptors can be created, viewed, modified, or deleted with the resource service. However, dashboards can be viewed only through the web interface of JasperReports Server because they do not have any output format that can be generated or transmitted through the REST API.

Therefore, an application using the REST API can only manipulate the definition of the dashboard, that is the selection of reports to display and their layout. In order to work with a dashboard, your application must parse its resource descriptor, make changes, and generate a new, valid descriptor to send back to the server.

The general structure of a dashboard descriptor contains:

- Typical descriptor properties such as `label`, `description`, and `PROP_PARENT_FOLDER`.
- The `dashboardState` descriptor containing:
 - The `ADHOC_FRAMES` property that lists the reports, labels, and buttons, and gives their coordinates in the dashboard.
 - The `ADHOC_PROPERTIES` property that gives the overall dashboard layout properties.
- reference descriptors for each of the reports included in the `ADHOC_FRAMES` property. These references ensure that the reports can't be deleted from the repository as long as they are used in this dashboard.

The following example shows the contents of a dashboard's resource descriptor:

```
<resourceDescriptor name="SampleDashboard"
  wsType="dashboard" uriString="/Dashboards/SampleDashboard" isNew="false">
  <label>Sample Dashboard</label>
  <description>Created in Dashboard Designer, viewed through REST.</description>
  <creationDate>1318380317305</creationDate>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.ji.adhoc.DashboardResource</value></resourceProperty>
  <resourceProperty name="PROP_PARENT_FOLDER"><value>/Dashboards</value>
  </resourceProperty>
  <resourceProperty name="PROP_VERSION"><value>0</value></resourceProperty>
  <resourceProperty name="PROP_HAS_DATA"><value>>false</value></resourceProperty>
  <resourceDescriptor name="" wsType="dashboardState" uriString="" isNew="false">
    <label>null</label>
    <creationDate>1318380317305</creationDate>
    <resourceProperty name="PROP_RESOURCE_TYPE"><value>dashboardState</value>
    </resourceProperty>
    <resourceProperty name="PROP_PARENT_FOLDER">
      <value>/Dashboards/SampleDashboard</value></resourceProperty>
    <resourceProperty name="ADHOC_PAPER_SIZE"><value>content</value>
    </resourceProperty>
    <resourceProperty name="ADHOC_FRAMES">
      <value>frame_1,com.jaspersoft.ji.adhoc.DashboardContentFrame,dashFrameLeft=0;
frame_1,com.jaspersoft.ji.adhoc.DashboardContentFrame,dashFrameTop=0;
frame_1,com.jaspersoft.ji.adhoc.DashboardContentFrame,dashFrameWidth=246;
frame_1,com.jaspersoft.ji.adhoc.DashboardContentFrame,dashFrameHeight=405;
frame_1,com.jaspersoft.ji.adhoc.DashboardContentFrame,dashFrameResourceType=
  com.jaspersoft.jasperserver.api.metadata.jasperreports.domain.ReportUnit;
frame_1,com.jaspersoft.ji.adhoc.DashboardContentFrame,dashFrameResourceName=
  Top Fives Report;
```

```
frame_1, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameSource=%2Fflow.html
%3F_flowId%3DviewReportFlow%26viewAsDashboardFrame%3Dtrue%26reportUnit%3D;
frame_1, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashResourceIndex=0;
frame_1, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameScrollBars=false;

frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameLeft=254;
frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameTop=0;
frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameWidth=450;
frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameHeight=418;
frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameResourceType=
com.jaspersoft.jasperserver.api.metadata.jasperreports.domain.ReportUnit;
frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameResourceName=
Sales By Month Report;

frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameSource=%2Fflow.html
%3F_flowId%3DviewReportFlow%26viewAsDashboardFrame%3Dtrue%26reportUnit%3D;
frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashResourceIndex=1;
frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameScrollBars=false;
frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameResourceType=
com.jaspersoft.jasperserver.api.metadata.jasperreports.domain.ReportUnit;
frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameResourceName=
Sales By Month Report;

frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameSource=%2Fflow.html
%3F_flowId%3DviewReportFlow%26viewAsDashboardFrame%3Dtrue%26reportUnit%3D;
frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashResourceIndex=1;
frame_2, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameScrollBars=false;
frame_3, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameLeft=712;
frame_3, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameTop=0;
frame_3, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameWidth=200;
frame_3, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameHeight=350;
frame_3, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameResourceType=
com.jaspersoft.jasperserver.api.metadata.jasperreports.domain.ReportUnit;
frame_3, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameResourceName=
Sales Gauges Report;

frame_3, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameSource=%2Fflow.html
%3F_flowId%3DviewReportFlow%26viewAsDashboardFrame%3Dtrue%26reportUnit%3D;
frame_3, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashResourceIndex=2;
frame_3, com.jaspersoft.ji.adhoc.DashboardContentFrame, dashFrameScrollBars=false;

text_2, com.jaspersoft.ji.adhoc.DashboardTextFrame, dashFrameLeft=736;
text_2, com.jaspersoft.ji.adhoc.DashboardTextFrame, dashFrameTop=352;
text_2, com.jaspersoft.ji.adhoc.DashboardTextFrame, dashFrameWidth=66;
text_2, com.jaspersoft.ji.adhoc.DashboardTextFrame, dashFrameHeight=16;
text_2, com.jaspersoft.ji.adhoc.DashboardTextFrame, dashTextFrameLabel=Start Month;
text_2, com.jaspersoft.ji.adhoc.DashboardTextFrame, fontResizes=false;
text_2, com.jaspersoft.ji.adhoc.DashboardTextFrame, dashTextFrameFontSize=11;
text_2, com.jaspersoft.ji.adhoc.DashboardTextFrame, maxFontSize=11;
control_2, com.jaspersoft.ji.adhoc.DashboardControlFrame, dashFrameLeft=816;
control_2, com.jaspersoft.ji.adhoc.DashboardControlFrame, dashFrameTop=352;
control_2, com.jaspersoft.ji.adhoc.DashboardControlFrame, dashFrameWidth=85;
control_2, com.jaspersoft.ji.adhoc.DashboardControlFrame, dashFrameHeight=16;
```

```

control_2,com.jaspersoft.ji.adhoc.DashboardControlFrame,dashControlFrameParamName=
    startMonth;
control_2,com.jaspersoft.ji.adhoc.DashboardControlFrame,dashControlFrameParamValue=
    1;
control_2,com.jaspersoft.ji.adhoc.DashboardControlFrame,dashControlFrameDefaultParam
    Value=1;
control_2,com.jaspersoft.ji.adhoc.DashboardControlFrame,dashControlFrameDataType=
    String;

text_3,com.jaspersoft.ji.adhoc.DashboardTextFrame,dashFrameLeft=744;
text_3,com.jaspersoft.ji.adhoc.DashboardTextFrame,dashFrameTop=376;
text_3,com.jaspersoft.ji.adhoc.DashboardTextFrame,dashFrameWidth=59;
text_3,com.jaspersoft.ji.adhoc.DashboardTextFrame,dashFrameHeight=16;
text_3,com.jaspersoft.ji.adhoc.DashboardTextFrame,dashTextFrameLabel=End Month;
text_3,com.jaspersoft.ji.adhoc.DashboardTextFrame,fontResizes=false;
text_3,com.jaspersoft.ji.adhoc.DashboardTextFrame,dashTextFrameFontSize=11;
text_3,com.jaspersoft.ji.adhoc.DashboardTextFrame,maxFontSize=11;

control_3,com.jaspersoft.ji.adhoc.DashboardControlFrame,dashFrameLeft=816;
control_3,com.jaspersoft.ji.adhoc.DashboardControlFrame,dashFrameTop=376;
control_3,com.jaspersoft.ji.adhoc.DashboardControlFrame,dashFrameWidth=85;
control_3,com.jaspersoft.ji.adhoc.DashboardControlFrame,dashFrameHeight=16;
control_3,com.jaspersoft.ji.adhoc.DashboardControlFrame,dashControlFrameParamName=
    endMonth;
control_3,com.jaspersoft.ji.adhoc.DashboardControlFrame,dashControlFrameParamValue=
    12;
control_3,com.jaspersoft.ji.adhoc.DashboardControlFrame,dashControlFrameDefaultParam
    Value=12;
control_3,com.jaspersoft.ji.adhoc.DashboardControlFrame,dashControlFrameDataType=
    String;

button_1,com.jaspersoft.ji.adhoc.DashboardClickableFrame,dashFrameLeft=832;
button_1,com.jaspersoft.ji.adhoc.DashboardClickableFrame,dashFrameTop=400;
button_1,com.jaspersoft.ji.adhoc.DashboardClickableFrame,dashFrameWidth=72;
button_1,com.jaspersoft.ji.adhoc.DashboardClickableFrame,dashFrameHeight=24;

button_1,com.jaspersoft.ji.adhoc.DashboardClickableFrame,dashClickableFrameID=
    submit;
button_1,com.jaspersoft.ji.adhoc.DashboardClickableFrame,dashClickableFrameType=
    button;

button_2,com.jaspersoft.ji.adhoc.DashboardClickableFrame,dashFrameLeft=744;
button_2,com.jaspersoft.ji.adhoc.DashboardClickableFrame,dashFrameTop=400;
button_2,com.jaspersoft.ji.adhoc.DashboardClickableFrame,dashFrameWidth=72;
button_2,com.jaspersoft.ji.adhoc.DashboardClickableFrame,dashFrameHeight=24;
button_2,com.jaspersoft.ji.adhoc.DashboardClickableFrame,dashClickableFrameID=reset;
button_2,com.jaspersoft.ji.adhoc.DashboardClickableFrame,dashClickableFrameType=
    button;
    </value>
</resourceProperty>

```

```

<resourceProperty name="ADHOC_PROPERTIES">
  <value>layoutSize=1024x768;
    useAbsoluteSizing=true;
    paperSize=content;
    LayoutLeft=286;
    paramValuesChanged=true;
    LayoutTop=176;
    localDatePattern=MM-dd-yyyy;
  </value>
</resourceProperty>
</resourceDescriptor>

<resourceDescriptor name="" wsType="reference" uriString="" isNew="false">
  <label>null</label>
  <resourceProperty name="PROP_REFERENCE_URI">
    <value>/supermart/details/TopFivesReport</value></resourceProperty>
</resourceDescriptor>
<resourceDescriptor name="" wsType="reference" uriString="" isNew="false">
  <label>null</label>
  <resourceProperty name="PROP_REFERENCE_URI">
    <value>/supermart/salesByMonth/SalesByMonthReport</value></resourceProperty>
</resourceDescriptor>
<resourceDescriptor name="" wsType="reference" uriString="" isNew="false">
  <label>null</label>
  <resourceProperty name="PROP_REFERENCE_URI">
    <value>/supermart/revenueAndProfit/SalesGaugesReport</value>
  </resourceProperty>
</resourceDescriptor>
</resourceDescriptor>

```

2.4 Working with Virtual Data Sources

Data sources define a connection to a database or other source of data for running reports. Data sources are resources in the repository that can be created, modified, and deleted with the repository service.

As with all descriptors, the descriptors for data sources contain properties and values that define the data source. Different types of data sources have different properties, but all are self-explanatory. For example, the following call returns the descriptor for a virtual data source in the sample data:

```
GET http://localhost:8080/jasperserver/rest/resource/datasources/SugarFoodmartVDS
```

The resource descriptor is shown below. The data sources that make up the virtual data source are given as children descriptors of type generic datasource. Each child descriptor has an ID within the virtual data source (PROP_DATASOURCE_SUB_DS_ID) and a repository URI (PROP_REFERENCE_URI):

```

<resourceDescriptor name="SugarFoodmartVDS" wsType="virtual"
  uriString="/datasources/SugarFoodmartVDS" isNew="false">
  <label>SugarCRM-Foodmart Virtual Data Source</label>
  <description>Virtual Data Source Combining SugarCRM and Foodmart</description>
  <creationDate>1366267873303</creationDate>

```

```

<resourceProperty name="PROP_RESOURCE_TYPE">
  <value>com.jaspersoft.jasperserver.api.metadata.jasperreports.domain.
    VirtualReportDataSource</value>
</resourceProperty>
<resourceProperty name="PROP_PARENT_FOLDER">
  <value>/datasources</value>
</resourceProperty>
<resourceProperty name="PROP_VERSION">
  <value>0</value>
</resourceProperty>
<resourceProperty name="PROP_SECURITY_PERMISSION_MASK">
  <value>33</value>
</resourceProperty>
<resourceDescriptor wsType="datasource" isNew="false">
  <resourceProperty name="PROP_REFERENCE_URI">
    <value>/analysis/datasources/SugarCRMDDataSourceJNDI</value>
  </resourceProperty>
  <resourceProperty name="PROP_IS_REFERENCE">
    <value>true</value>
  </resourceProperty>
  <resourceProperty name="PROP_DATASOURCE_SUB_DS_ID">
    <value>SugarCRMDDataSource</value>
  </resourceProperty>
</resourceDescriptor>
<resourceDescriptor wsType="datasource" isNew="false">
  <resourceProperty name="PROP_REFERENCE_URI">
    <value>/analysis/datasources/FoodmartDataSourceJNDI</value>
  </resourceProperty>
  <resourceProperty name="PROP_IS_REFERENCE">
    <value>true</value>
  </resourceProperty>
  <resourceProperty name="PROP_DATASOURCE_SUB_DS_ID">
    <value>FoodmartDataSource</value>
  </resourceProperty>
</resourceDescriptor>
</resourceDescriptor>

```

If you wanted more information about the child data sources, use the resource service again to request their descriptors, for example:

```
GET http://localhost:8080/jasperserver/rest/resource/analysis/datasources/FoodmartDataSourceJNDI
```

2.5 Working with Domains

The resource service can also be used to read and write Domains. Like JasperReport resources, Domains are complex resources that contain other files. The files that make up a Domain are its Domain schema, its optional security files, and its optional language bundles. You can find the name and location of these files by requesting the Domain itself. For example, the following request shows the contents of the Supermart Domain:

```
GET http://localhost:8080/jasperserver-pro/rest/resource/Domains/supermartDomain
```

Looking carefully through the resulting descriptor, we find the relevant information:

```

<resourceDescriptor name="supermartDomain" wsType="domain"
  uriString="/Domains/supermartDomain" isNew="false">
  <label>Supermart Domain</label>
  <description>Comprehensive example of Domain (pre-joined table sets for complex
reporting, custom query based dataset, column and row security, I18n bundles)</
description>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.commons.semantic.datasource.SemanticLayerDataSource
  ...

<resourceDescriptor name="supermartDomain_schema" wsType="xml"
  uriString="/Domains/supermartDomain_files/supermartDomain_schema" isNew="false">
  <label>supermartDomain_schema</label>
  <description>supermartDomain_schema</description>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.FileResource
  ...
</resourceDescriptor>

<resourceDescriptor name="supermart_domain.properties" wsType="prop"
  uriString="/Domains/supermartDomain_files/supermart_domain.properties"
  isNew="false">
  <label>supermart_domain.properties</label>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.FileResource
  ...
</resourceDescriptor>

<resourceDescriptor name="supermart_domain_en_US.properties" wsType="prop"
  uriString="/Domains/supermartDomain_files/supermart_domain_en_US.properties"
  isNew="false">
  <label>supermart_domain_en_US.properties</label>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.FileResource
  ...
</resourceDescriptor>
...

<resourceDescriptor name="supermartDomain_domain_security" wsType="xml"
  uriString="/Domains/supermartDomain_files/supermartDomain_domain_security"
  isNew="false">
  <label>supermartDomain_domain_security</label>
  <description>supermartDomain_domain_security</description>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.FileResource
  ...
  <resourceProperty name="PROP_SECURITY_BUNDLE">
    <value>true</value>
  </resourceProperty>
</resourceDescriptor>

```

```
<resourceDescriptor name="dsFoodMart" wsType="reference" referenceType="jndi"
  uriString="/analysis/datasources/FoodmartDataSourceJNDI" isNew="false">
  <resourceProperty name="PROP_REFERENCE_URI">
    <value>/analysis/datasources/FoodmartDataSourceJNDI</value>
  </resourceProperty>
</resourceDescriptor>
</resourceDescriptor>
```

The files contained in this Domain are:

- /Domains/supermartDomain_files/supermartDomain_schema
- /Domains/supermartDomain_files/supermartDomain_domain_security
- /Domains/supermartDomain_files/supermart_domain.properties
- /Domains/supermartDomain_files/supermart_domain_en_US.properties
- ...

To download these files, use the same syntax as for downloading file resource contents:

```
GET http://localhost:8080/jasperserver-pro/rest/resource/Domains/supermartDomain_files/
supermartDomain_schema?fileData=true
```

Domain schema files, security files, and language bundles have a special syntax described in the *JasperReports Server User Guide*. To process these files automatically, you would need to write your own parser.

If you have valid schema files, security files, and language bundles, you can also create Domain resources in the repository using the PUT method described in section 2.2.4, “Creating a Resource,” on page 31. The following procedure outlines the step required to create a Domain resource programmatically through the REST API:

1. Create the resource descriptor for the Domain using the descriptor shown above as an example. Make sure the files are referenced correctly in the descriptor.
2. Create the PUT request to the resource service using the resource descriptor created in the previous step.
3. Add the files to the request as multi-part request. For example use the class `org.apache.http.entity.mime.MultipartEntity` to build a multi-part request. Add each file as a separate part to the request.
4. Send the multi-part request to JasperReports Server.
5. Process the response to verify the request was successful.

2.6 The permission Service

The permission service lets you view and set access permission on repository folders and resources. Only administrative users and users granted the Administer permission may view and set permissions.

As it is implemented, the permission service returns and sets only explicit permissions on resources. The lack of an explicit permission for a given role or user means that the permission is inherited from its parent folder. To find the value of inherited permissions on a given resource, you must obtain the permissions of all of its parent folders.

2.6.1 Viewing Permissions

The GET method retrieves the permissions defined on a resource, including both user-based and role-based permissions.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest/permission/path/to/resource/
Return Value on Success	Typical Return Values on Failure
200 OK – The body is XML that describes all permissions for the resource.	404 Not Found – When the specified resource URI is not found in the repository.

The permissions for each user and each role are indicated by the following values. These values are not a true mask; they should be treated as constants:

- No access: 0
- Administer: 1
- Read-only: 2
- Read-delete: 18
- Read-write-delete: 30
- Execute-only: 32

The response to the GET request is an `entityResource` that defines each permission on the resource. Permissions for roles or users that are not specified are inherited from their parent folder:

```
<entityResource>
  <Item xsi:type="objectPermissionImpl"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <permissionMask>2</permissionMask>
    <permissionRecipient xsi:type="roleImpl">
      <externallyDefined>>false</externallyDefined>
      <roleName>ROLE_USER</roleName>
    </permissionRecipient>
    <URI>repo:/path/to/resource</URI>
  </Item>

  <Item xsi:type="objectPermissionImpl"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <permissionMask>30</permissionMask>
    <permissionRecipient xsi:type="roleImpl">
      <externallyDefined>>false</externallyDefined>
      <roleName>ROLE_DEMO</roleName>
    </permissionRecipient>
    <URI>repo:/path/to/resource</URI>
  </Item>

  <Item xsi:type="objectPermissionImpl"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <permissionMask>30</permissionMask>
    <permissionRecipient xsi:type="userImpl">
      <externallyDefined>>false</externallyDefined>
      <fullName>California User</fullName>
      <tenantId>organization_1</tenantId>
      <username>CaliforniaUser</username>
    </permissionRecipient>
    <URI>repo:/path/to/resource</URI>
  </Item>
```

```

<Item xsi:type="objectPermissionImpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <permissionMask>30</permissionMask>
  <permissionRecipient xsi:type="userImpl">
    <externallyDefined>>false</externallyDefined>
    <fullName>Joe User</fullName>
    <tenantId>organization_1</tenantId>
    <username>joeuser</username>
  </permissionRecipient>
  <URI>repo:/path/to/resource</URI>
</Item>
</entityResource>

```

2.6.2 Setting Permissions

PUT is implemented as a synonym of POST for the permission service. Both will create an explicit permission for a given role or user on a resource, overriding the previous explicit permission for the same role or user. To set a permission use either method and include the permission descriptors (`objectPermissionImpl`) such as those returned by the GET method.

Method	URL
PUT or POST	http://<host>:<port>/jasperserver[-pro]/rest/permission/path/to/resource/
Content-Type	Content
text/plain	A well-formed XML entityResource that defines the permissions you want to set.
Return Value on Success	Typical Return Value on Failure
200 OK	404 Not Found – When the specified resource URI is not found in the repository.

Setting a permission creates an explicit permission for the given user or role. To reset the inherited permission value, remove the explicit permission with the DELETE method. This method does not take a permission descriptor, instead specify the roles and users to reset to the inherited permission as parameters.

Method	URL	
DELETE	http://<host>:<port>/jasperserver[-pro]/rest/permission/path/to/resource/?<argument>	
Argument	Type/Value	Description
roles	String	A comma-separated list of role names. The access permission for the specified roles will revert to the resource's inherited permission for those roles.
users	String	A comma-separated list of user IDs. The access permission for the specified users will revert to the resource's inherited permission for those users.
Return Value on Success	Typical Return Values on Failure	
200 OK	404 Not Found – When the specified resource URI is not found in the repository.	

2.7 The v2/permissions Service

In the REST v2/permissions service, the syntax is expanded so that you can specify the resource, the recipient (user name or role name) and the permission value within the URL. This makes it simpler to set permissions because you don't need to send a resource descriptor to describe the permissions. In order to set, modify, or delete permissions, you must use credentials or login with a user that has "administer" permissions on the target resource.

The permissions for each user and each role are indicated by the following values. These values are not a true mask; they should be treated as constants:

- No access: 0
- Administer: 1
- Read-only: 2
- Read-delete: 18
- Read-write-delete: 30
- Execute-only: 32

Because a permission can apply to either a user or a role, the permissions service uses the concept of a "recipient." A recipient specifies whether the permission applies to a user or a role, and gives the ID of the user or role, including any organization, for example:

```
role:/ROLE_SUPERUSER
user:/organization_1/joeuser
```

Recipients are listed when viewing permissions, and they are also used to set a permission. A recipient can be used in the URL when allowed, but in this case, the forward slash (/) characters must be encoded as %2F.

There are two qualities of a permission:

- The assigned permission is one that is set explicitly for a given resource and a given user or role. Not all permissions are assigned, in which case the permission is inherited from the parent folder.
- The effective permission is the permission that is being enforced, whether it is assigned or inherited.

2.7.1 Viewing Multiple Permissions

The GET method of the v2/permissions service lists permissions on a given resource according to several arguments.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions/path/to/resource/?<arguments>	
Argument	Type/Value	Description
effectivePermissions	boolean optional	When set to true, the effective permissions are returned. By default, this argument is false and only assigned permissions are returned.
recipientType	String optional	Either <code>user</code> or <code>role</code> . When not specified, the recipient type is the role.
recipientId	String optional	Id of the user or role. In environments with multiple organizations, specify the the organization as %2F<orgID>%2F<recipientID>
resolveAll	boolean optional	When set to true, shows the effective permissions for all users and all roles.
Options		
accept: application/xml (default)		
accept: application/json		
Return Value on Success		Typical Return Values on Failure
200 OK – The body describes the requested permissions for the resource.		400 Bad Request – When the recipient type is invalid. 404 Not Found – When the specified resource URI is not found in the repository or the recipient ID cannot be resolved.

For example, the following request shows all permission for a resource, similar to the permissions dialog in the user interface:

GET http://localhost:8080/jasperserver-pro/rest_v2/permissions/public?resolveAll=true

```
<permissions>
  <permission>
    <mask>0</mask>
    <recipient>user:/anonymousUser</recipient>
  </permission>
  <permission>
    <mask>0</mask>
    <recipient>user:/organization_1/CaliforniaUser</recipient>
  </permission>
  ...
  <permission>
    <mask>2</mask>
    <recipient>role:/ROLE_USER</recipient>
    <uri>/public</uri>
  </permission>
</permissions>
```

2.7.2 Viewing a Single Permission

Specify the recipient in the URL to see a specific assigned permission. To view effective permissions, use the form above.

Method	URL	
GET	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions/path/to/resource;recipient=<recipient>">http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions/path/to/resource;recipient=<recipient>	
Argument	Type/Value	Description
recipient	string required	The recipient format specifies <code>user</code> or <code>role</code> , the organization if necessary, and the object ID. The slash characters must be encoded, for example: user:%2Forganization_1%2Fjoeuser
Options		
accept: application/xml (default)		
accept: application/json		
Return Value on Success		Typical Return Values on Failure
200 OK – The body describes the requested permission.		404 Not Found – When the specified resource URI or recipient is invalid.

2.7.3 Setting Multiple Permissions

The POST method assigns any number of permissions to any number of resources specified in the body of the request. All permissions must be newly assigned, and the request will fail if a recipient already has an assigned (not inherited) permission. Use the PUT method to update assigned permissions.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions	
Content-Type	Content	
application/collection+json	<p>A JSON object that describes a set of permissions, for example:</p> <pre>{ "permission" : [{ "uri": "/properties", "recipient": "role:/ROLE_USER", "mask": "1" }, { "uri": "/properties", "recipient": "role:/ROLE_ADMIN", "mask": "32" }] }</pre>	
Return Value on Success	Typical Return Values on Failure	
201 Created – The request was successful.	400 Bad Request – A permission is already assigned or the given permission mask is invalid.	

The PUT method modifies existing permissions (already assigned).

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions/path/to/resource	
Content-Type	Content	
application/collection+json	<p>A JSON object that describes a set of permissions. Because a single resource is specified in the URL, all permissions apply to the same resource, and the server ignores the uri field in the JSON object.</p> <pre>{ "permission" : [{ "uri": "/foo", "recipient": "role:/organization_1/ROLE_MANAGER", "mask": "30" }, { "uri": "/bar", "recipient": "user:/organization_1/joeuser", "mask": "32" }] }</pre>	
Return Value on Success	Typical Return Values on Failure	
200 OK – The request was successful.	400 Bad Request – If a recipient or mask is invalid. 404 Not Found – If the resource in the URL is invalid.	

2.7.4 Setting a Single Permission

The POST method accepts a single permission descriptor.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions	
Content-Type	Content	
application/json	A JSON object that describes a single permission on a single resource, for example: <pre>{ "uri": "/properties", "recipient": "role:/ROLE_USER", "mask": "1" }</pre>	
Return Value on Success		Typical Return Values on Failure
201 Created – The request was successful.		400 Bad Request – The permission is already assigned or the given mask is invalid.

The PUT method accepts a resource and recipient in the URL.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions/path/to/resource;recipient=<recipient>	
Argument	Type/Value	Description
recipient	string required	The recipient format specifies <code>user</code> or <code>role</code> , the organization if necessary, and the object ID. The slash characters must be encoded, for example: user:%2Forganization_1%2Fjoeuser
Content-Type	Content	
application/json	A JSON object that describes only the mask, for example: <pre>{ "uri": null, "recipient": null, "mask": "2" }</pre>	
Return Value on Success		Typical Return Values on Failure
200 OK – The request was successful, and the response body contains the single permission that was modified.		400 Bad Request – If the mask is invalid. 404 Not Found – If the resource or the recipient in the URL is invalid.

2.7.5 Deleting Permissions in Bulk

The DELETE method removes all assigned permissions from the designated resource. After returning successfully, all effective permissions for the resource are inherited.

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions/path/to/resource
Return Value on Success	Typical Return Values on Failure
204 No Content – The request was successful.	404 Not Found – If the resource in the URL is invalid.

2.7.6 Deleting a Single Permission

Specify a recipient in the URL of the DELETE method to remove only that permission.

Method	URL	
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions/path/to/resource;recipient=<recipient>	
Argument	Type/Value	Description
recipient	string required	The recipient format specifies <code>user</code> or <code>role</code> , the organization if necessary, and the object ID. The slash characters must be encoded, for example: user:%2Forganization_1%2Fjoeuser
Return Value on Success	Typical Return Values on Failure	
204 No Content – The request was successful.	404 Not Found – If the resource or the recipient in the URL is invalid.	

2.8 The v2/export Service

The export service works asynchronously: first you request the export with the desired options, then you monitor the state of the export, and finally you request the output file. Each step requires a different service call. You must be authenticated as the system admin (superuser) for the export services.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/export/
Content-Type	Content
application/json	A JSON object that describes the export options.
Return Value on Success	Typical Return Values on Failure
200 OK – Returns a JSON object that gives the ID of the started export operation.	401 Unauthorized – Export is available only to the system admin user (superuser).

The content to send describes the export options, for example:

```
{
  roles: ["ROLE_USER", "organization_1|ROLE_MANAGER"],
  users: ["jasperadmin", "organization_1|joe"],
  uris: ["path/to/resource/1", "organizations/organization_1/path/to/resource/2"]
  parameters: ["role-users", "repository-permissions"]
}
```

As shown above, commercial editions must use the organization syntax for all roles, users, and URIs. The following table describes the export parameters:

Export Options	
everything	Export everything except audit and monitoring: all repository resources, permissions, report jobs, users, roles, and server settings.
role-users	When this option is present, each role export triggers the export of all users belonging to the role. This option should only be used if roles are specified
repository-permissions	When this option is present, repository permissions are exported along with each exported folder and resource. This option should only be used if uris are specified.
include-access-events	When this option is present, access events (date, time, and user name of last modification) are exported along with each exported folder and resource. This option should only be used if uris are specified.
include-audit-events	Include audit data for all resources and users in the export. The audit feature must be enabled in the server configuration.
include-monitoring-events	Include monitoring events. The monitoring feature must be enabled in the server configuration.

The body of the response contains the ID of the export operation needed to check its status and later download the file:

```
{
  id: "njkhfs8374",
  state: {
    phase: "inprogress",
    message: "Progress..."
  }
}
```

2.8.1 Checking the Export State

After receiving the export ID, you can check the state of the export operation.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/export/<export-id>/state
Return Value on Success	Typical Return Values on Failure
200 OK – Returns a JSON object that gives the current state of the export operation.	404 Not Found – When the specified export ID is not found.

The body of the response contains the current state of the export operation:

{ phase: "inprogress", message: "Progress..." }	{ phase: "ready", message: "Ready!" }	{ phase: "failure", message: "Not enough space on disk" }
--	--	---

2.8.2 Fetching the Export Output

When the export state is `ready`, you can download the zip file containing the export catalog.

Method	URL
GET	http://<host>:<port>/jasperserver[-proj]/rest_v2/export/<export-id>/<fileName>
Return Value on Success	Typical Return Values on Failure
200 OK – Returns the exported catalog as a zip file with the given <fileName>.	404 Not Found – When the specified export ID is not found.

2.9 The v2/import Service

Use the following service to upload a catalog as a zip file and import it with the given options. Specify options as arguments in the format <argument>=true. Arguments that are omitted are assumed to be false. You must be authenticated as the system admin (superuser) for the import service.

Jaspersoft does not recommend uploading files greater than 2 gigabytes.

Method	URL	
POST	http://<host>:<port>/jasperserver[-proj]/rest_v2/import?<arguments>	
Argument	Value	Description
update?	true	Resources in the catalog replace those in the repository if their URIs and types match.
skip-user-update?	true	When used with update=true, users in the catalog are not imported or updated. Use this option to import catalogs without overwriting currently defined users.
include-access-events?	true	Restores the date, time, and user name of last modification if they are included in the catalog to import.
include-audit-events?	true	Imports audit events if they are included in the catalog.
include-monitoring-events?	true	Imports audit events if they are included in the catalog.
include-server-setting?	true	Imports server settings if they are included in the catalog.
Content-Type	Content	
application/zip	The catalog file to import.	
Return Value on Success	Typical Return Values on Failure	
200 OK – Returns a JSON object that indicates the import was a success.	401 Unauthorized – Import is available only to the system admin user (superuser).	

The body of the response indicates the success of the import:

```
{
  phase: "ready",
  message: "Ready!"
}
```


CHAPTER 3 REST - REPORT WEB SERVICES

For authentication using the REST web services, see section 1.2, “[REST Authentication](#),” on page 12.

The RESTful report services gives responses that contain the same XML data structure that are used in the SOAP repository web service. These data structures are documented in section 1.6, “[Syntax of resourceDescriptor](#),” on page 17, with reference material in [Appendix A, “ResourceDescriptor API Constants](#),” on page 155.

This chapter includes the following sections:

- [The report Service](#)
- [The v2/reports Service](#)
- [The v2/reportExecutions Service](#)
- [The v2/inputControls Service](#)
- [The v2/options Service](#)
- [The jobsummary Service](#)
- [The job Service](#)
- [The v2/jobs Service](#)
- [The v2/queryExecutor Service](#)

3.1 The report Service



The rest/report service is superseded by “[The v2/reports Service](#)” on page 54 and “[The v2/reportExecutions Service](#)” on page 56.

The report service uses a combination of the PUT, GET, and POST methods to run reports and make them available in multiple ways through the API:

- The PUT method generates the report in any number of formats, stores the output in the session, and returns an identifier.
- The GET method with the identifier and file ID downloads any one of the file outputs.
- The POST method can be used to regenerate the report, for example in different formats, and supports page-by-page downloading of the PDF output.

The report service relies on the user session to store the report output in memory. While this does not follow the stateless nature of REST implementations, it reflects the performance optimization strategies in the JasperReports Server architecture. Filling and generating a report is resource intensive, so it is better to fill and generate once and store multiple output files temporarily than to generate the report from scratch for every output type that is requested.

3.1.1 Running a Report

The PUT request for the report service runs the report and generates the specified output. The response contains the ID of the saved output for downloading later with a GET request.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest/report/path/to/report	
Argument	Type/Value	Description
RUN_OUTPUT_FORMAT?	OutputType	The format of the report output. Possible values: PDF, HTML, XLS, RTF, CSV, XML, JRPRINT. The Default is PDF.
<Resource Descriptor>	String	Argument used to pass a transformer key to be used when running a report using JRPRINT as output format. The transformer key will be used to transform generic elements in the generated report as per net.sf.jasperreports.engine.export.GenericElementReportTransformer. This is a required argument when using multipart requests.
interactive?	Boolean	In a commercial editions of the server where HighCharts are used in the report, this property determines whether the JavaScript necessary for interaction is generated when exporting to HTML. By default it is true. If set to false, the chart is generated as a non-interactive image file.
IMAGES_URI?	String	The uri prefix used for images when exporting in HTML. The default is <code>images</code> .
X-Method-Override?	POST	This method can be used to perform a POST instead of a PUT.
PAGE?	Integer > 0	An integer value used to export a specific page
ignore Pagination	Boolean	When true, the report output will be generated on a single page in all export formats. When false or omitted, all export formats will be paginated.
onePagePer Sheet	Boolean	Valid only for the XLS format. When true, each page of the report is on a separate spreadsheet. When false or omitted, the entire report is on a single spreadsheet. If your reports are very long, set this argument to true, otherwise the report will not fit on a single spreadsheet and cause an error.
Return Value on Success		Typical Return Values on Failure
200 OK – The body is XML containing a summary of the report execution (UUID, pages, generated files, etc...). See sample return below.		404 Not Found – When the specified report URI is not found in the repository.

The body of the PUT request should contain a resource descriptor of type `reportUnit` with the URI of the report unit to run. The resource descriptor can contain one or more parameter tags to specify parameters. Lists can be passed using parameters with the same name and the `isListItem` attribute set to true.

The arguments can be placed in the URL of the request or by encoding them in the multipart request. However, some application servers such as Apache Tomcat do not process arguments that are `www-url-encoded` in the request when sent to a PUT method, so be sure you are using a multipart request or you are using the GET style parameters when using this method.

The return value of the PUT request provides the UUID of the report output in this session and the ID of the files.

```
<report>
  <uuid>d7bf6c9-9077-41f7-a2d4-8682e74b637e</uuid>
  <originalUri>/reports/samples/AllAccounts</originalUri>
  <totalPages>43</totalPages>
  <startPage>1</startPage>
  <endPage>43</endPage>
  <file type="image/png">img_0_0_0</file>
  <file type="image/gif">px</file>
  <file type="text/html">report</file>
  <file type="image/jpeg">img_0_42_27</file>
  <file type="image/png">img_0_42_26</file>
</report>
```

3.1.2 Downloading Report Output

Once a report has been generated with the PUT request, it is possible to download its files using a GET request.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest/report/<UUID>?<arguments> (see example below)	
Argument	Type/Value	Description
file?	String	One of the files specified in the report xml. If the file parameter is not specified, the service returns the report descriptor.
Return Value on Success		Typical Return Values on Failure
200 OK – The content is the requested file.		404 Not Found – When the specified UUID is not found in the user's session.

For example, the URL to download the HTML of the report generated in the previous example is:

```
http://<host>:<port>/jasperserver[-pro]/rest/report/d7bf6c9-9077-41f7-a2d4-8682e74b637e?file=report
```

As a side effect of storing the report output in the user session, the UUID in the URL is visible only to the currently logged user. Other applications using different user IDs cannot access this report output.



JasperReports Server does not support exporting Highcharts charts with background images to PDF, ODT, DOCX, or RTF formats. When exporting or downloading reports with Highcharts that have background images to these formats, the background image is removed from the chart. The data in the chart is not affected.

3.1.3 Regenerating Report Output

To export a report in a different format after its first execution, or to export a specific page, use the POST method of the report service. For example, it is possible to download the report one page at a time by repeatedly sending the appropriate POST and GET requests.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest/report/<UUID>?<arguments> (see example below)	
Argument	Type/Value	Description
RUN_OUTPUT_FORMAT?	OutputType	The format of the report output. Possible values: PDF, HTML, XLS, RTF, CSV, XML, JRPRINT. The Default is PDF.
IMAGES_URI?	String	The uri prefix used for images when exporting in HTML. The default is images.
PAGE?	Integer > 0	An integer value used to export a specific page.
Return Value on Success		Typical Return Values on Failure
200 OK – The new details of the report. The old files produces are discarded and replaced with new ones.		404 Not Found – When the specified UUID is not found in the user’s session.

For example, the following request exports page 10 of the PDF report:

POST http://host/rest/report/d7bf6c9-9077-41f7-a2d4-8682e74b637e?PAGE=10&RUN_OUTPUT_FORMAT=PDF

You then need to take the file name from the return value and create a GET request for it.

3.2 The v2/reports Service

The rest_v2/reports service reimplements the functionality of the rest/report service. The new service simplifies the API for obtaining report output, such as PDF and XLS. The new service also provides more functionality to interact with running reports, report options, and input controls.

3.2.1 Running a Report

The new v2/reports service allows clients to receive report output in a single request-response. The output format is specified in the URL as a file extension to the report URI.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/path/to/report.<format>?<arguments> where <format> is one of the following: html, pdf, xls, rtf, csv, xml, jrprint	
Argument	Type/Value	Description
page?	Integer > 0	An integer value used to export a specific page
<inputControl>	String	Any input control that is defined for the report. Input controls that are multi-select may appear more than once. See examples below.
interactive?	Boolean	In a commercial editions of the server where HighCharts are used in the report, this property determines whether the JavaScript necessary for interaction is generated when exporting to HTML. By default it is true. If set to false, the chart is generated as a non-interactive image file.
onePagePerSheet?	Boolean	Valid only for the XLS format. When true, each page of the report is on a separate spreadsheet. When false or omitted, the entire report is on a single spreadsheet. If your reports are very long, set this argument to true, otherwise the report will not fit on a single spreadsheet and cause an error.
Return Value on Success		Typical Return Values on Failure
200 OK – The content is the requested file.		404 Not Found – When the specified report URI is not found in the repository.

The follow examples show various combinations of formats, arguments, and input controls:

```
http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/AllAccounts.html (all pages)
http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/AllAccounts.html?page=43
http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/AllAccounts.pdf (all pages)
http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/AllAccounts.pdf?page=1
http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/EmployeeAccounts.html?EmployeeID=sarah_id
http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/Cascading_multi_select_report.html?
Country_multi_select=USA&Cascading_state_multi_select=WA&Cascading_state_multi_select=CA
```



JasperReports Server does not support exporting Highcharts charts with background images to PDF, ODT, DOCX, or RTF formats. When exporting or downloading reports with Highcharts that have background images to these formats, the background image is removed from the chart. The data in the chart is not affected.

3.2.2 Finding Running Reports

The new v2/reports service provides functionality to stop reports that are running. Reports can be running from user interaction, web service calls, or scheduling. The following method provides several ways to find reports that are currently running, in case the client wants to stop them.



This syntax of the v2/reports service is deprecated. See [“The v2/reportExecutions Service” on page 56](#).

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/path/to/report/ http://<host>:<port>/jasperserver[-pro]/rest_v2/reports?<arguments>	
Argument	Type/Value	Description
jobID?	String	Find the running report based on its jobID in the scheduler.
jobLabel?	String	Find the running report based on its jobLabel in the scheduler.
userName?	String	Name of user who has scheduled a report, in the format <username>%7C<organizationID>. The <organizationID> is required for all users except system admins (superuser).
fireTime From?	date/time	Date and time in the following pattern: yyyy-MM-dd'T'HH:mmZ. Together, these arguments create a time range to find when the running report was started. Both of the range limits are inclusive. Either argument may be null to signify an open-ended range.
fireTimeTo?	date/time	
Return Value on Success		Typical Return Values on Failure
200 OK – The content is a list of execution IDs that can be used for cancellation.		404 Not Found – When the specified report URI is not found in the repository.

For security purposes, the search for running reports is has the following restrictions:

- The system administrator (`superuser`) can see and cancel any report running on the server.
- An organization admin (`jasperadmin`) can see every running report, but can cancel only the reports that were started by a user of the same organization or one of its child organizations.
- A regular user can see every running report, but can cancel only the reports that he initiated.

3.2.3 Terminate Running Report

Use the following method to stop a running report, as found with the previous method.



This syntax of the v2/reports service is deprecated. See “[The v2/reportExecutions Service](#)” on page 56.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/<executionID>/status/
Content-Type	Content
application/xml	Either an empty instance of the ReportExecutionCancellation class or <status>cancelled</status>.
Return Value on Success	Typical Return Values on Failure
200 OK – The content also contains: <status>cancelled</status>.	204 No Content – When the specified execution ID is not found on the server, and the response body is empty.

3.3 The v2/reportExecutions Service

The v2/reports service allows clients to easily run a report, wait for the reply, and receive the output. This is called synchronous report execution because the client must wait for the response. When managing large reports that may take minutes to complete, or when running a large number of reports simultaneously, synchronous report execution slows down the client or uses many threads, each waiting for a report.

The v2/reportExecutions service provides asynchronous report execution, so that the client does not need to wait for report output. Instead, the client obtains a request ID and periodically checks the status of the report to know when it is ready. When the report is finished, the client can download the output. The client can also send an asynchronous request for other export formats (PDF, Excel, and others) of the same report. Again the client can check the status of the export and download the result when the export has completed.

Reports being scheduled on the server also run asynchronously, and the v2/reportExecutions allows you to access jobs that are triggered by the scheduler. Finally, the v2/reportExecutions service allows the client to stop any report execution or job that has been triggered.

3.3.1 Running a Report Asynchronously

In order to run a report asynchronously, the v2/reportExecutions service provides a method to specify all the parameters needed to launch a report. Report parameters are all sent as a `reportExecutionRequest` object. The response from the server contains the request ID needed to track the execution until completion.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions
Content-Type	Content
application/xml application/json	A complete <code>ReportExecutionRequest</code> in either XML or JSON format. See the example and table below for an explanation of its properties.
Return Value on Success	Typical Return Values on Failure
200 OK – The content contains a <code>ReportExecution</code> descriptor. See below for an example	403 Forbidden – When the logged-in user does not have permission to access the report in the request. 404 Not Found – When the report URI specified in the request does not exist.

The following example shows the structure of the `ReportExecutionRequest`:

```
<reportExecutionRequest>
  <reportUnitUri>/supermart/details/CustomerDetailReport</reportUnitUri>
  <async>true</async>
  <freshData>false</freshData>
  <saveDataSnapshot>false</saveDataSnapshot>
  <outputFormat>html</outputFormat>
  <interactive>true</interactive>
  <ignorePagination>false</ignorePagination>
  <pages>1-5</pages>
  <parameters>
    <reportParameter name="someParameterName">
      <value>value 1</value>
      <value>value 2</value>
    </reportParameter>
    <reportParameter name="someAnotherParameterName">
      <value>another value</value>
    </reportParameter>
  </parameters>
</reportExecutionRequest>
```

The following table describes the properties you can specify in the `ReportExecutionRequest`:

Table 3-1 Report Execution Properties

Property	Required or Default	Description
<code>reportUnitUri</code>	Required	Repository path (URI) of the report to run. For commercial editions with organizations, the URI is relative the the logged-in user's organization.
<code>outputFormat</code>	Required	Specifies the desired output format; one of html, pdf, xls, rtf, csv, xml, or jrprint.
<code>freshData</code>	false	When data snapshots are enabled, specifies whether the report should get fresh data by querying the data source or if false, use a previously saved data snapshot (if any). By default, if a saved data snapshot exists for the report it will be used when running the report.
<code>saveDataSnapshot</code>	false	When data snapshots are enabled, specifies whether the data snapshot for the report should be written or overwritten with the new data from this execution of the report.
<code>interactive</code>	true	In a commercial editions of the server where HighCharts are used in the report, this property determines whether the JavaScript necessary for interaction is generated and returned as an attachment when exporting to HTML. If false, the chart is generated as a non-interactive image file (also as an attachment).
<code>ignorePagination</code>	false	When set to true, the report is generated as a single long page. This can be used with HTML output to avoid pagination.
<code>pages</code>	Optional	Specify a page range to generate a partial report. The format is <code><startPageNumber>-<endPageNumber></code>

Table 3-1 Report Execution Properties

Property	Required or Default	Description
async	false	Determines whether reportExecution is synchronous or asynchronous. When set to true, the response is sent immediately and the client must poll the report status and later download the result when ready. By default, this property is false and the operation will wait until the report execution is complete, forcing the client to wait as well, but allowing the client to download the report immediately after the response.
transformerKey	Optional	Advanced property used when requesting a report as a JasperPrint object. This property can specify a JasperReports Library generic print element transformers of class net.sf.jasperreports.engine.export.GenericElementTransformer. These transformers are pluggable as JasperReports.extensions
attachmentsPrefix	attachments	For HTML output, this property specifies the URL path to use fo downloading the attachment files (JavaScript and images). The full path of the devault value is: <code>{contextPath}/rest_v2/reportExecutions/{reportExecutionId}/exports/{exportOptions}/attachments/</code> You can specify a different URL path using the placeholders <code>{contextPath}</code> , <code>{reportExecutionId}</code> and <code>{exportOptions}</code> .
parameters		A list of input control parameters and their values.

When successful, the reply from the server contains the `reportExecution` descriptor. This descriptor contains the request ID and status needed in order for the client to request the output. There are two statuses, one for the report execution itself, and one for the chosen output format. The following descriptor shows that the report is still executing (`<status>execution</status>`).

```
<reportExecution>
  <currentPage>1</currentPage>
  <exports>
    <export>
      <id>html</id>
      <status>queued</status>
    </export>
  </exports>
  <reportURI>/supermart/details/CustomerDetailReport</reportURI>
  <requestId>677629003_1355225037212_1</requestId>
  <status>execution</status>
</reportExecution>
```

The value of the `async` property in the request determines whether or not the report output is available when the response is received. Your client should implement either synchronous or asynchronous processing of the response depending on the value you set for the `async` property.

3.3.2 Polling Report Execution

When requesting reports asynchronously, use the following method to poll the status of the report execution. The request ID in the URL is the one returned in the `reportExecution` descriptor.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/status/
Options	
accept: application/xml (default) accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – The content contains the report status, for example: XML: <status>ready</status> JSON: { "value": "ready" }	404 Not Found – When the request ID specified in the request does not exist.

3.3.3 Requesting Report Execution Details

Once the report is ready, your client must determine the names of the files to download by requesting the `reportExecution` descriptor again. Specify the `requestID` in the URL as follows:

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID
Options	
accept: application/xml (default) accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – The content contains a <code>ReportExecution</code> descriptor. See below for an example.	404 Not Found – When the request ID specified in the request does not exist.

The report execution descriptor now contains the list of exports for the report, including the report output itself and any other file attachments. File attachments such as images and JavaScript occur only with HTML export.

```
<reportExecution>
  <reportURI>/reports/samples/AllAccounts</reportURI>
  <requestId>912382875_1366638024956_2</requestId>
  <status>ready</status>
  <totalPages>47</totalPages>
  <currentPage>1</currentPage>
  <exports>
    <export>
      <id>html</id>
      <outputResource>
        <contentType>text/html</contentType>
      </outputResource>
    </export>
  </exports>
  <status>ready</status>
```

```

    <attachments>
      <attachment>
        <contentType>image/jpeg</contentType>
        <fileName>img_0_46_1</fileName>
      </attachment>
      <attachment>
        <contentType>image/png</contentType>
        <fileName>img_0_0_0</fileName>
      </attachment>
      <attachment>
        <contentType>image/png</contentType>
        <fileName>img_0_46_0</fileName>
      </attachment>
    </attachments>
  </export>
</exports>
</reportExecution>

```

3.3.4 Requesting Report Output

After requesting a report execution and waiting synchronously or asynchronously for it to finish, your client is ready to download the report output.

Every export format of the report has an ID that is used to retrieve it. For example, the HTML export in the previous example has the ID `html`. To download the main report output, specify this export ID in the following method:

Method	URL
GET	<code>http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/exports/exportID/outputResource</code>
Return Value on Success	Typical Return Values on Failure
200 OK – The content is the main output of the report, in the format specified by the <code>contentType</code> property of the <code>outputResource</code> descriptor, for example: <code>text/html</code>	404 Not Found – When the request ID specified in the request does not exist.

For example, to download the main HTML of the report execution response above, use the following URL:

```
GET http://localhost:8080/jasperserver-pro/rest_v2/reportExecutions/912382875_1366638024956_2/exports/html/outputResource
```



JasperReports Server does not support exporting Highcharts charts with background images to PDF, ODT, DOCX, or RTF formats. When exporting or downloading reports with Highcharts that have background images to these formats, the background image is removed from the chart. The data in the chart is not affected.

To download file attachments for HTML output, use the following method. You must download all attachments to display the HTML content properly. The given URL is the default path, but it can be modified with the `attachmentsPrefix` property in the `reportExecutionRequest`, as described in [Table 3-1 on page 57](#).

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/exports/exportID/attachments/fileName
Return Value on Success	Typical Return Values on Failure
200 OK – The content is the attachment in the format specified in the <code>contentType</code> property of the attachment descriptor, for example: image/png	404 Not Found – When the request ID specified in the request does not exist.

For example, to download the one of the images for the HTML report execution response above, use the following URL:

```
GET http://localhost:8080/jasperserver-pro/rest_v2/reportExecutions/912382875_1366638024956_2/exports/html/attachments/img_0_46_0
```

3.3.5 Exporting a Report Asynchronously

After running a report and downloading its content in a given format, you can request the same report in other formats. As with exporting report formats through the user interface, the report does not run again because the export process is independent of the report.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/exports/
Content-Type	Content
application/xml application/json	Send an <code>export</code> descriptor in either XML or JSON format to specify the format and details of your request. For example: <pre><export> <outputFormat>html</outputFormat> <pages>10-20</pages> <attachmentsPrefix>./images/</attachmentsPrefix> </export></pre>
Options	
accept: application/xml (default) accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – The content contains an <code>exportExecution</code> descriptor. See below for an example.	404 Not Found – When the request ID specified in the request does not exist.

The following example shows the `exportExecution` descriptor that the server sends in response to the export request:

```
<exportExecution>
  <id>html;attachmentsPrefix=./images/</id>
  <status>ready</status>
  <outputResource>
    <contentType>text/html</contentType>
  </outputResource>
</exportExecution>
```

3.3.6 Polling Export Execution

As with the execution of the main report, you can also poll the execution of the export process.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/exports/exportID/status	
Options		
accept: application/xml (default) accept: application/json		
Return Value on Success		Typical Return Values on Failure
200 OK – The content contains the status of the export process, similar to the status field of the <code>exportExecution</code> descriptor. For example: XML: <status>ready</status> JSON: { "value": "ready" }		404 Not Found – When the request ID specified in the request does not exist.

For example, to get the status of the HTML export in the previous example, use the following URL:

```
GET http://localhost:8080/jasperserver-pro/rest_v2/reportExecutions/912382875_1366638024956_2/exports/html/status
```

When the status is “ready” your client can download the new export output and any attachments as described in section 3.3.4, “Requesting Report Output,” on page 60. For example:

```
GET http://localhost:8080/jasperserver-pro/rest_v2/reportExecutions/912382875_1366638024956_2/exports/html/outputResource
```

```
GET http://localhost:8080/jasperserver-pro/rest_v2/reportExecutions/912382875_1366638024956_2/exports/html/images/img_0_46_0
```

3.3.7 Finding Running Reports and Jobs

The `v2/reportExecutions` service provides a method to search for reports that are running on the server, including report jobs triggered by the scheduler.

To search for running reports, use the search arguments with the following URL:

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions?<arguments>	
Argument	Type/Value	Description
<code>reportURI</code>	Optional String	This string matches the repository URI of the running report, relative the currently logged-in user’s organization.
<code>jobID</code>	Optional String	For scheduler jobs, this argument matches the ID of the job that triggered the running report.
<code>jobLabel</code>	Optional String	For scheduler jobs, this argument matches the name of the job that triggered the running report.
<code>userName</code>	Optional String	For scheduler jobs, this argument matches the user ID that created the job.

fireTimeFrom	Optional Date/Time	For scheduler jobs, the fire time arguments define a range of time that matches if the job that is currently running was triggered during this time. You can specify either or both of the arguments. Specify the date and time in the following pattern: yyyy-MM-dd'T'HH:mmZ.
fireTimeTo		
Options		
accept: application/xml (default) accept: application/json		
Return Value on Success		Typical Return Values on Failure
200 OK – The content is a descriptor for each of the matching results. 204 No Content – When the search results are empty.		

The response contains a list of summary `reportExecution` descriptors, for example in XML:

```
<reportExecutions>
  <reportExecution>
    <reportURI>repo:/supermart/details/CustomerDetailReport</reportURI>
    <requestId>2071593484_1355224559918_5</requestId>
  </reportExecution>
</reportExecutions>
```

Given the request ID, you can obtain more information about each result by downloading the full `reportExecution` descriptor, as described in section 3.3.3, “[Requesting Report Execution Details](#),” on page 59.

For security purposes, the search for running reports is has the following restrictions:

- The system administrator (`superuser`) can see and cancel any report running on the server.
- An organization admin (`jasperadmin`) can see every running report, but can cancel only the reports that were started by a user of the same organization or one of its child organizations.
- A regular user can see every running report, but can cancel only the reports that he initiated.

3.3.8 Stopping Running Reports and Jobs

To stop a report that is running and cancel its output, use the PUT method and specify a status of “cancelled” in the body of the request.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/status/
Content-Type	Content
application/xml application/json	Send a <code>status</code> descriptor in either XML or JSON format with the value cancelled. For example: XML: <code><status>cancelled</status></code> JSON: <code>{ "value": "cancelled" }</code>
Options	
accept: application/xml (default) accept: application/json	

Return Value on Success	Typical Return Values on Failure
<p>200 OK – When the report execution was successfully stopped, the server replies with the same status: XML: <status>cancelled</status> JSON: { "value": "cancelled" }</p> <p>204 No Content – When the report specified by the request ID is not running, either because it finished running, failed, or was stopped by another process.</p>	<p>404 Not Found – When the request ID specified in the request does not exist.</p>

3.4 The v2/inputControls Service

The v2/reports service includes methods for reading and setting input controls. The inputControls methods return an XML descriptor by default, but you can optionally specify the JSON format. The example in this section use the JSON format.

3.4.1 Listing Input Control Structure

The following method returns a description of the structure of the input controls for a given report.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/path/to/report/inputControls/
Options	
accept: application/json	
Return Value on Success	Typical Return Values on Failure
<p>200 OK – The content is a JSON object that describes the input control structure. See example below.</p>	<p>404 Not Found – When the specified report URI is not found in the repository.</p>

The body of the response contains the structure of the input controls for the report. This structure contains the information needed by your application to display the input controls to your users and allow them to make a selection. In particular, this includes any cascading structure as a set of dependencies between input controls. The following example shows a response in the JSON format:

```

{
  "inputControl" : [ {
    "id": "Cascading_name_single_select",
    "label": "Cascading name single select",
    "mandatory": "true",
    "readOnly": "false",
    "type": "singleSelect",
    "uri": "repo:/reports/samples/Cascading_multi_select_report_files/
Cascading_name_single_select",
    "visible": "true",
    "masterDependencies": { "controlId": [ "Country_multi_select", "Cascading_state_multi_
select" ] },
    "slaveDependencies": null,
    "validationRules": [ { ... } ]
  } ]
}

```



```

"state": {
  "uri": "/reports/samples/Cascading_multi_select_report_files/
    Cascading_name_single_select",
  "id": "Cascading_name_single_select",
  "value": null,
  "options": [{
    "selected": false,
    "label": "A & U Jaramillo Telecommunications, Inc",
    "value": "A & U Jaramillo Telecommunications, Inc"
  }, ... ]}
},
...
]}

```

The structure includes a set of validation rules for each input control. These rules indicate what type of validation your client should perform on input control values it receives from your users, and if the validation fails, the message to display. Depending on the type of the input control, the following validations are possible:

- `mandatoryValidationRule` – This input is required and your client should ensure the user enters a value.
- `dateTimeFormatValidation` – This input must have a data time format and your client should ensure the user enters a valid date and time.

The following sample shows the structure of these two possible validation rules.

```

"validationRules": [{
  "mandatoryValidationRule" : {
    "errorMessage" : "This field is mandatory so you must enter data."
  },
  "dateTimeFormatValidationRule" : {
    "errorMessage" : "Specify a valid date value.",
    "format" : "yyyy-MM-dd"
  }
}]

```

3.4.2 Listing Input Control Values

The following method returns a description of the possible values of all input controls for the report. Among these choices, it shows which ones are selected.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/path/to/report/inputControls/values/
Options	
accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – The content is a JSON object that describes the input control values and selection.	404 Not Found – When the specified report URI is not found in the repository.

The body of the response contains the structure of the input controls for the report. The following example shows a response in the JSON format:

```
{
  "inputControlState" : [ {
    "uri" : "/reports/samples/.../Country_multi_select",
    "value" : "",
    "options" : {
      "option" : [ {
        "label" : "Canada",
        "selected" : "false",
        "value" : "Canada"
      }, {
        "label" : "Mexico",
        "selected" : "false",
        "value" : "Mexico"
      }, {
        "label" : "USA",
        "selected" : "true",
        "value" : "USA"
      } ]
    }
  },
  ...
]
```



If a selection-type input control has a null value, it is given as ~NULL~. If no selection is made, its value is given as ~NOTHING~.

3.4.3 Setting Input Control Values

The following method updates the state of current input control values, so they are set for the next run of the report.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/path/to/report/inputControls/<ic1>;<ic2>;.../values/	
Content-Type	Content	
application/json	A JSON object that lists the new selected values. See example below.	
Options		
accept: application/json		
Return Value on Success	Typical Return Values on Failure	
200 OK – The content is a JSON object that describes the new selection of input control values.	404 Not Found – When the specified report URI is not found in the repository.	

Assuming the client receives the response given in section 3.4.2, “Listing Input Control Values,” on page 65, it can send the following request body:

```
{
  "Country_multi_select":["Mexico"],
  "Cascading_state_multi_select":["Guerrero", "Sinaloa"]
}
```

When specifying the option for the JSON format, the server’s response is:

```
{
  "inputControlState" : [ {
    "uri" : "/reports/samples/.../Country_multi_select",
    "value" : "",
    "options" : {
      "option" : [ {
        "label" : "Canada",
        "selected" : "false",
        "value" : "Canada"
      }, {
        "label" : "Mexico",
        "selected" : "true",
        "value" : "Mexico"
      }, {
        "label" : "USA",
        "selected" : "false",
        "value" : "USA"
      } ]
    }
  },
  ...
]
```

3.5 The v2/options Service

Report options are sets of input control values that are saved in the repository. A report option is always associated with a report.

3.5.1 Listing Report Options

The following method retrieves a list of report options summaries. The summaries give the name of the report options, but not the input control values that are associated with it.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/path/to/report/options/
Options	
accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – The content is a JSON object that lists the names of the report options for the given report.	404 Not Found – When the specified report URI is not found in the repository.

The body of the response contains the labels of the report options, for example:

```
{
  "reportOptionsSummary": [{
    "uri": "/reports/samples/Options",
    "id": "Options",
    "label": "Options"
  },
  {
    "uri": "/reports/samples/Options_2",
    "id": "Options_2",
    "label": "Options 2"
  }
  ]
}
```

3.5.2 Creating Report Options

The following method creates a new report option for a given report. A report option is defined by a set of values for all of the report’s input controls.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/path/to/report/options?<arguments>	
Argument	Type/Value	Description
label	string	The name to give the new report option.
overwrite?	true / false	If true, any report option that has the same label is replaced. If false or omitted, any report option with the same label will not be replaced.
Content-Type	Content	
application/json	A JSON object that lists the input control selections. See example below.	
Options		
accept: application/json		
Return Value on Success	Typical Return Values on Failure	
200 OK – The content is a JSON object that describes the new selection of input control values.	404 Not Found – When the specified report URI is not found in the repository.	

In this example, we create new options for the sample report named Cascading_multi_select_report:

```
http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/Cascading_multi_select_report/options?label=MyReportOption
```

With the following request body:

```
{
  "Country_multi_select":["Mexico"],
  "Cascading_state_multi_select":["Guerrero", "Sinaloa"]
}
```

When successful, the server responds with a JSON object that describes the new report options, for example:

```
{
  "uri":"/reports/samples/MyReportOption",
  "id":"MyReportOption",
  "label":"MyReportOption"
}
```

3.5.3 Updating Report Options

Use the following method to modify the values in a given report option.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/path/to/report/options/<optionID>/
Content-Type	Content
application/json	A JSON object that lists the input control selections. See example below.
Return Value on Success	Typical Return Values on Failure
200 OK	404 Not Found – When the specified report URI is not found in the repository.

For example, we change the report option we created in section 3.5.2, “Creating Report Options,” on page 68 with the following header:

```
http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/Cascading_multi_select_report/options/MyReportOption
```

And the following request body:

```
{
  "Country_multi_select":["USA"],
  "Cascading_state_multi_select":["CA", "WA"]
}
```

3.5.4 Deleting Report Options

Use the following method to delete a given report option.

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/path/to/report/options/<optionID>/
Return Value on Success	Typical Return Values on Failure
200 OK	404 Not Found – When the specified report URI is not found in the repository.

3.6 The jobsummary Service



The rest/jobsummary service is superseded by [“The v2/jobs Service” on page 73](#).

In order to schedule reports and interact with jobs that are created to run a report at a later time, the REST API provides two services:

- The jobsummary service lists all currently defined jobs on a given report.
- The job service lets you create, modify, and delete a specific job.

The jobsummary service is a read only service. Requests for PUT, POST, and DELETE operations receive the error 405, method not allowed.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest/jobsummary/path/to/report/
Return Value on Success	Typical Return Values on Failure
200 OK – The body contains XML that describes all the active jobs	404 Not Found – When the specified report is not found in the server.

The jobs are described in jobsummary elements such as the following example:

```
<jobs>
  <jobsummary>
    <id>22164</id>
    <label>MyJob</label>
    <nextFireTime>2011-11-11T11:11:11-08:00</nextFireTime>
    <reportUnitURI>/organizations/organization_1/reports/samples/AllAccounts
      </reportUnitURI>
    <state>
      <value>NORMAL</value>
    </state>
    <version>0</version>
  </jobsummary>
  <jobsummary>
    ...
  </jobsummary>
</jobs>
```

The job summary gives the ID of the job that you need to interact with it using the job service. It also gives the next occurrence (“fire time”) of the job, and its status that would indicate any errors.

3.7 The job Service



The rest/job service is superseded by [“The v2/jobs Service” on page 73](#).

The job service lets you view details of a job, edit them, create new jobs, and delete existing ones. For interacting with existing jobs, use the jobID that you obtained from the jobsummary service.

3.7.1 Viewing a Job Definition

The GET method for the job service retrieves the information about a scheduled job.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest/job/<jobID>/
Return Value on Success	Typical Return Values on Failure
200 OK – The body contains XML that describes all the job properties	404 Not Found – When the specified job is not found in the server.

The GET method returns a `job` element that gives the output, scheduling, and parameter details, if any, for the job:

```
<job>
  <baseOutputFilename>AllAccounts</baseOutputFilename>
  <description>Sample job</description>
  <id>22164</id>
  <label>MyJob</label>

  <mailNotification>
    <id>22163</id>
    <messageText></messageText>
    <resultSendType><value>SEND</value></resultSendType>
    <skipEmptyReports>false</skipEmptyReports>
    <subject>Scheduled AllAccounts report</subject>
    <toAddresses>example@example.com</toAddresses>
    <version>2</version>
  </mailNotification>

  <outputFormats>PDF</outputFormats>
  <outputFormats>HTML</outputFormats>
  <outputLocale></outputLocale>
  <reportUnitURI>/reports/samples/AllAccounts</reportUnitURI>

  <repositoryDestination>
    <folderURI>/reports/samples</folderURI>
    <id>22162</id>
    <outputDescription></outputDescription>
    <overwriteFiles>false</overwriteFiles>
    <sequentialFileNames>false</sequentialFileNames>
    <version>0</version>
  </repositoryDestination>

  <simpleTrigger>
    <id>22161</id>
    <startDate>2011-11-11T11:11:11-08:00</startDate>
    <timezone>America/Los_Angeles</timezone>
    <version>0</version>
    <occurrenceCount>1</occurrenceCount>
  </simpleTrigger>
  <version>0</version>
</job>
```

3.7.2 Scheduling a Report

To schedule a report, create its job descriptor and use the PUT method of the job service. Specify the report being scheduled inside the job descriptor. You do not need to specify any job IDs in the descriptor, because the server will assign them.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest/job/	
Content-Type		Content
text/plain		A well-formed XML job descriptor.
Return Value on Success		Typical Return Values on Failure
201 Created – The body contains XML Job descriptor. This is the only case where a descriptor returns for a put request and that id due to the fact that the job id (the job handle) is created in the server.		404 Not Found – When the report specified in the job descriptor is not found in the server.

The output formats are those supported by JasperReports server, as given by the following values:

- ♦ PDF
- ♦ HTML
- ♦ CSV
- ♦ ODS
- ♦ XLS
- ♦ XLS_NOPAG
- ♦ XLSX
- ♦ XLSX_NOPAG
- ♦ DOCX
- ♦ RTF
- ♦ ODT

The recurrence can be defined as follows:

- ♦ No recurrence (single run), for example:

```
<simpleTrigger>
  <startDate>2011-11-11T11:11:11-08:00</startDate>
  <timezone>America/Los_Angeles</timezone>
  <version>0</version>
  <occurrenceCount>1</occurrenceCount>
</simpleTrigger>
```

- ♦ Simple recurrence, for example every day until a given date:

```
<simpleTrigger>
  <endDate>2011-11-11T11:11:11-08:00</endDate>
  <startDate>2012-12-12T12:12:12-08:00</startDate>
  <timezone>America/Los_Angeles</timezone>
  <version>0</version>
  <occurrenceCount>-1</occurrenceCount>
  <recurrenceInterval>1</recurrenceInterval>
  <recurrenceIntervalUnit>
    <value>DAY</value>
  </recurrenceIntervalUnit>
</simpleTrigger>
```


- Calendar recurrence, for example every Tuesday and Thursday in February, April, and June until next year:

```

< <calendarTrigger>
  <endDate>2012-12-12T12:12:12-08:00</endDate>
  <timezone>America/Los_Angeles</timezone>
  <version>0</version>
  <daysType><value>WEEK</value></daysType>
  <hours>0</hours>
  <minutes>0</minutes>
  <monthDays></monthDays>
  <months>2</months>
  <months>4</months>
  <months>6</months>
  <weekDays>3</weekDays>
  <weekDays>5</weekDays>
</calendarTrigger>

```

3.7.3 Editing a Job Definition

To modify an existing job definition, use the GET method to read its job descriptor, modify the descriptor as required, and use the POST method of the job service. The POST method replaces the definition of the job with the given job ID.

Method	URL	
POST	http://<host>:<port>/jasperserver[-proj]/rest/job/<jobID>/	
Content-Type	Content	
text/plain	A well-formed XML job descriptor.	
Return Value on Success	Typical Return Values on Failure	
200 OK	404 Not Found – When the specified job is not found in the server.	

3.7.4 Deleting a Job Definition

Use the DELETE method to delete a job identified by its jobID. Use the jobsummary service to see the job IDs for a given report.

Method	URL	
DELETE	http://<host>:<port>/jasperserver[-proj]/rest/job/<jobID>/	
Return Value on Success	Typical Return Values on Failure	
200 OK	404 Not Found – When the specified job is not found in the server.	

3.8 The v2/jobs Service

The rest_v2/jobs service reimplements the functionality of the jobsummary and job services. In addition, the new service provides an API to scheduler features that were introduced in JasperReports Server 4.7, such as bulk updates, pausing jobs, FTP output and exclusion calendars.

3.8.1 Listing Report Jobs

Use the following method to list jobs, either all jobs managed by the scheduler or the jobs for a specific report:

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs?<argument>	
Argument	Type/Value	Description
reportUnit URI?	/path/to/report	Gives the repository URI of a report to list all jobs. When this argument is omitted, this method returns all jobs for all reports.
Return Value on Success		Typical Return Values on Failure
200 OK – The body contains XML that describes jobs in the scheduler.		404 Not Found – When no job is not found in the server.

The jobs are described in the `jobsummary` element such as the following example:



The `jobsummary` XML element returned by the `rest_v2/jobs` service has a different structure than the element with the same name returned by the `rest/jobsummary` service.

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

<jobs>
  <jobsummary>
    <id>2042</id>
    <label>JUnit_Job_New</label>
    <reportUnitURI>/organizations/organization_1/reports/samples/AllAccounts
      </reportUnitURI>
    <state>
      <nextFireTime>2222-02-04T13:47:00+02:00</nextFireTime>
      <value>NORMAL</value>
    </state>
    <version>1</version>
  </jobsummary>
</jobs>
```

3.8.2 Viewing a Job Definition

The GET method retrieves the information about a scheduled job.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/<jobID>/	
Return Value on Success		Typical Return Values on Failure
200 OK – The body contains XML that describes all the job properties.		404 Not Found – When the specified job is not found in the server.

The GET method returns a `job` element that gives the output, scheduling, and parameter details, if any, for the job.



The job XML element returned by the rest_v2/jobs service has a different structure than the element with the same name returned by the rest/job service.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<job>
  <baseOutputFilename>AllAccounts</baseOutputFilename>
  <repositoryDestination>
    <folderURI>/reports/samples</folderURI>
    <id>2041</id>
    <outputDescription/>
    <overwriteFiles>>false</overwriteFiles>
    <sequentialFileNames>>false</sequentialFileNames>
    <version>0</version>
  </repositoryDestination>
  <description/>
  <id>2042</id>
  <label>MyNewJob</label>
  <mailNotification>
    <bccAddresses/>
    <ccAddresses/>
    <id>2007</id>
    <includingStackTraceWhenJobFails>>false</includingStackTraceWhenJobFails>
    <messageText>Body of message</messageText><
    resultSendType>SEND_ATTACHMENT</resultSendType>
    <skipEmptyReports>>true</skipEmptyReports>
    <skipNotificationWhenJobFails>>false</skipNotificationWhenJobFails>
    <subject>Subject of message</subject>
    <toAddresses><address>name@example.com</address></toAddresses>
    <version>0</version>
  </mailNotification>
  <outputFormats>
    <outputFormat>XLS</outputFormat>
    <outputFormat>CSV</outputFormat>
    <outputFormat>PDF</outputFormat>
    <outputFormat>HTML</outputFormat>
    <outputFormat>DOCX</outputFormat>
  </outputFormats>
  <outputLocale/>
  <reportUnitURI>/reports/samples/AllAccounts</reportUnitURI>
  <simpleTrigger>
    <id>2040</id>
    <startDate>2222-02-04T03:47:00+02:00</startDate>
    <timezone>America/Los_Angeles</timezone>
    <version>0</version>
    <occurrenceCount>1</occurrenceCount>
  </simpleTrigger>
  <version>1</version>
</job>
```

3.8.3 Extended Job Search

The GET method is also used for more advanced job searches. Some field of the jobsummary descriptor can be used directly as parameters, and fields of the job descriptor can also be used as search criteria. You can also control the pagination and sorting order of the reply.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs?<arguments>	
Argument	Type/Value	Description
label	string	The name of the report job.
owner	string	The username of the report job creator; the user who scheduled the report.
reportUnit URI?	/path/to/report	Gives the repository URI of a report to list all jobs. When this argument is omitted, this method returns all jobs for all reports.
example?	JSON jobModel	Searches for jobs that match the JSON jobModel. The jobModel is a fragment of a job descriptor containing one or more fields to be matched.
numberOf Rows	integer	Turns on pagination of the result by specifying the number of jobsummary descriptors per results page.
startIndex	integer	Determines the page number in paginated results by specifying the index of the first jobsummary to be returned.
sortType		Possible values are: NONE, SORTBY_JOBID, SORTBY_JOBNAME, SORTBY_REPORTURI, SORTBY_REPORTNAME, SORTBY_REPORTFOLDER, SORTBY_OWNER, SORTBY_STATUS, SORTBY_LASTRUN, SORTBY_NEXTRUN
isAscending	true / false	Determines the sort order: ascending if true, descending if false or omitted.
Return Value on Success		Typical Return Values on Failure
200 OK – The body contains XML that describes jobs in the scheduler that match the search criteria.		404 Not Found – When the specified report is not found in the server.

The body of the return value is an XML jobs descriptor containing jobsummary descriptors, as shown in section [3.8.1, “Listing Report Jobs,” on page 74](#).

The `example` parameter lets you specify a search on fields in the job descriptor, such as output formats. Some fields may be specified in both the `example` parameter and in a dedicated parameter, for example `label`. In that case, the search specified in the `example` parameter takes precedence.

For example, you can search for all jobs that specify and output format of PDF. The JSON string to specify this field is:

```
{"outputFormat":"PDF"}
```

And the corresponding URI, with proper encoding, is:

```
http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs?example=%7b%22outputFormat%22%3a%22PDF%22%7d
```

3.8.4 Scheduling a Report

To schedule a report, create its job descriptor similar to the one returned by the GET method, and use the PUT method of the V2/jobs service. Specify the report being scheduled inside the job descriptor. You do not need to specify any job IDs in the descriptor, because the server will assign them.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/	
Content-Type		Content
application/xml application/json		A well-formed XML or JSON job descriptor.
Return Value on Success		Typical Return Values on Failure
201 Created – The body contains the XML job descriptor of the newly created job. It is similar to the one that was sent but now contains the jobID for the new job.		404 Not Found – When the report specified in the job descriptor is not found in the server.

3.8.5 Viewing Job Status

The following method returns the current runtime state of a job:

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/<jobID>/state/	
Return Value on Success		Typical Return Values on Failure
200 OK – Body contains the status descriptor.		404 Not Found – When the specified <jobID> does not exist.

3.8.6 Editing a Job Definition

To modify an existing job definition, use the GET method to read its job descriptor, modify the descriptor as required, and use the POST method of the v2/jobs service. The POST method replaces the definition of the job with the given job ID.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/<jobID>/	
Content-Type		Content
application/xml application/json		A well-formed XML or JSON job descriptor.
Return Value on Success		Typical Return Values on Failure
200 OK		404 Not Found – When the specified job is not found in the server.

3.8.7 Updating Jobs in Bulk

The POST method also supports other parameters to perform bulk updates on scheduled jobs.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs?<arguments>	
Argument	Type/Value	Description
id?	jobID string	Can be used multiple times to create a list of jobIDs to update
replace Trigger IgnoreType	true / false	When true, the trigger is replaced from the content being sent and the trigger type is ignored. When false or omitted, the trigger is updated automatically by the scheduler.
Content-Type		Content
application/xml		A well-formed XML jobModel descriptor. The jobModel is a fragment of a job descriptor containing only the fields to be updated. See example below.
Return Value on Success		Typical Return Values on Failure
200 OK – Body empty.		404 Not Found – When the specified job is not found in the server.

For example, the following request will update the job description in several jobs:

POST request: http://localhost:8080/jasperserver-pro/rest_v2/jobs?id=3798&id=3799&id=3800

And the body of the request contains:

```
<jobModel>
  <description>This description updated in bulk</description>
</jobModel>
```

3.8.8 Pausing Jobs

The following method pauses currently scheduled job execution. Pausing keeps the job schedule and all other details but prevents the job from running. It does not delete the job.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/pause/	
Content-Type		Content
application/xml		A well-formed XML jobIdList descriptor that specifies the IDs of the jobs. See example below. If the body of the request is empty, or the list is empty, all jobs in the scheduler will be paused.
Return Value on Success		Typical Return Values on Failure
200 OK		

The following example shows a list of jobs sent in the body of the request.

```
<jobIdList>
  <jobId>1236</jobId>
  <jobId>1237</jobId>
  <jobId>1238</jobId>
  <jobId>1239</jobId>
</jobIdList>
```

3.8.9 Resuming Jobs

Use the following method to resume any or all paused jobs in the scheduler. Resuming a job means that any defined trigger after the time it is resumed will

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/resume/
Content-Type	Content
application/xml	A well-formed XML jobIdList descriptor that specifies the IDs of the jobs. See example below. If the body of the request is empty, or the list is empty, all paused jobs in the scheduler will resume.
Return Value on Success	Typical Return Values on Failure
200 OK	

The XML format of the jobIdList descriptor in the request body is identical to the one used when pausing jobs:

```
<jobIdList>
  <jobId>1236</jobId>
  <jobId>1237</jobId>
</jobIdList>
```

3.8.10 Restarting Failed Jobs

Use the following method to rerun failed jobs in the scheduler. For each job to be restarted, the scheduler creates an immediate single-run copy of job, to replace the one that failed. Therefore, all jobs listed in the request body will run once immediately after issuing this command. The single-run copies have a misfire policy set so that they do not trigger any further failures (MISFIRE_INSTRUCTION_IGNORE_MISFIRE_POLICY). If the single-run copies fail themselves, no further attempts are made automatically.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/restart/
Content-Type	Content
application/xml	A well-formed XML jobIdList descriptor that specifies the IDs of the jobs. See example below.
Return Value on Success	Typical Return Values on Failure
200 OK	

The XML format of the jobIdList descriptor in the request body is identical to the one used when pausing jobs:

```
<jobIdList>
  <jobId>8321</jobId>
  <jobId>8322</jobId>
</jobIdList>
```

3.8.11 Specifying FTP Output

The REST service allows a job to specify output to remote files through FTP (File Transfer Protocol). In addition to the repository location, you can specify an FTP server and path where JasperReports Server will write the output files when the job runs. You also need to provide a username and password to access the FTP server.

To specify these parameters, add the `outputFTPInfo` element to the XML job descriptor, as shown in the following example:

```
<job>
  <reportUnitURI>/reports/samples/AllAccounts</reportUnitURI>
  <label>MyJob</label>
  <description>MyJob description</description>
  <baseOutputFilename>WeeklyAccountsReport</baseOutputFilename>
  <repositoryDestination>
    <folderURI>/reports/samples</folderURI>
    <outputDescription/>
    <overwriteFiles>true</overwriteFiles>
    <sequentialFileNames>false</sequentialFileNames>
    <outputFTPInfo>
      <serverName>ftpserver.example.com</serverName>
      <userName>ftpUser</userName>
      <password>ftpPassword</password>
      <folderPath>/Shared/Users/ftpUser</folderPath>
    </outputFTPInfo>
  </repositoryDestination>
  <outputFormats>
    <outputFormat>XLS</outputFormat>
    <outputFormat>PDF</outputFormat>
  </outputFormats>
  ...
</job>
```

FTP output is always specified in addition to repository output, and the output will be written to both the repository and the FTP location. You cannot specify FTP output alone. The file names to be written are the same ones that are generated by the job output, as specified by the `baseOutputFilename`, sequential pattern if any, and format extensions such as `.pdf`. Similarly, the file overwrite and sequential filename behavior specified for repository output also apply to FTP output.

3.8.12 Calendar Exclusion for the Scheduler

The scheduler allows a job to be defined with a list of excluded days or times when you do not want the job to run. For example, if you have a report scheduled to run every business day, you want to exclude holidays that change every year. The list for excluded days and times is defined as a calendar, and there are various ways to define the calendar.

The scheduler stores any number of exclusion calendars that you can reference by name. When scheduling a report, reference the name of the calendar to exclude, and the scheduler automatically calculates the correct days to trigger the report. The scheduler also allows you to update an exclusion calendar and update all of the report jobs that used it. Therefore, you can update the calendar of excluded holidays every year and not need to modify any report jobs.

3.8.12.1 Listing All Registered Calendar Names

The following method returns the list of all calendar names that were added to the scheduler.

Method	URL
GET	http://<host>:<port>/jasperserver[-proj]/rest_v2/jobs/calendars/
Return Value on Success	Typical Return Values on Failure
200 OK – Body is XML that contains a list of calendar names.	401 Unauthorized

The list of calendar names in the result has the following XML format:

```
<calendarNameList>
  <calendarName>name1</calendarName>
  <calendarName>name2</calendarName>
</calendarNameList>
```

3.8.12.2 Viewing an Exclusion Calendar

The following method takes the name of an exclusion calendar and returns the definition of the calendar:

Method	URL
GET	http://<host>:<port>/jasperserver[-proj]/rest_v2/jobs/calendars/<calendarName>/
Return Value on Success	Typical Return Values on Failure
200 OK – Body is XML that contains the requested calendar.	404 Not Found – When the specified calendar name does not exist.

The calendar descriptor in the result has the following XML format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<reportJobCalendar>
  <baseCalendar>
    <calendarType>base</calendarType>
    <excludeDates/>
    <description>Base calendar description</description>
    <excludeDays/>
    <timeZone>GMT+03:00</timeZone>
  </baseCalendar>
  <calendarType>daily</calendarType>
  <excludeDates/>
  <description>Main calendar description</description>
  <excludeDays/>
  <invertTimeRange>>false</invertTimeRange>
  <rangeEndingCalendar >2012-03-20T14:44:37.353+03:00</rangeEndingCalendar>
  <rangeStartingCalendar>2012-03-20T14:43:37.353+03:00</rangeStartingCalendar>
  <timeZone>GMT+03:00</timeZone>
</reportJobCalendar>
```

3.8.12.3 Deleting an Exclusion Calendar

Use the following method to delete a calendar by name.

Method	URL
DELETE	http://<host>:<port>/jasperserver[-proj]/rest_v2/jobs/calendars/<calendarName>/
Return Value on Success	Typical Return Values on Failure
200 OK	404 Not Found – When the specified calendar name does not exist.

3.8.12.4 Adding or Updating an Exclusion Calendar

This method creates a named exclusion calendar that you can use when scheduling reports. If the calendar already exists, you have the option of replacing it and updating all the jobs that used it.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-proj]/rest_v2/jobs/calendars/<calendarName>?<arguments>	
Argument	Type/Value	Description
replace?	true / false	If true, any calendar existing in the JobStore with the same name is overwritten. When this argument is omitted, it is false by default.
update Triggers?	true / false	Whether or not to update existing triggers that referenced the already existing calendar so that they are based on the new trigger.
Content-Type	Content	
application/xml	A well-formed XML calendar descriptor (see examples below).	
Return Value on Success	Typical Return Values on Failure	
200 OK –	404 Not Found – When the specified calendar name does not exist.	

The following examples show the types of exclusion calendars that you can add to the scheduler:

- Base calendar.

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

<reportJobCalendar>
  <calendarType>base</calendarType>
  <description>Base calendar description</description>
  <timeZone>GMT+03:00</timeZone>
</reportJobCalendar>
```

- Annual calendar – A list of days that you want to exclude every year.

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

<reportJobCalendar>
  <calendarType>annual</calendarType>
  <description>Annual calendar description</description>
  <timeZone>GMT+03:00</timeZone>
```

```

<excludeDays>
  <excludeDay>2012-03-20</excludeDay>
  <excludeDay>2012-03-21</excludeDay>
  <excludeDay>2012-03-22</excludeDay>
</excludeDays>
</reportJobCalendar>

```

- Cron calendar – Defines the days and times to exclude as a cron expression.

```

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

<reportJobCalendar>
  <calendarType>cron</calendarType>
  <description>Cron format description</description>
  <cronExpression>0 30 10-13 ? * WED,FRI</cronExpression>
  <timeZone>GMT+03:00</timeZone>
</reportJobCalendar>

```

- Daily calendar – Defines a time range to exclude every day.

```

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

<reportJobCalendar>
  <calendarType>daily</calendarType>
  <description>Daily calendar description</description>
  <invertTimeRange>false</invertTimeRange>
  <rangeEndingCalendar>2012-03-20T14:44:37.353+03:00</rangeEndingCalendar>
  <rangeStartingCalendar>2012-03-20T14:43:37.353+03:00</rangeStartingCalendar>
  <timeZone>GMT+03:00</timeZone>
</reportJobCalendar>

```

- Holiday calendar – Defines a set of days to exclude that can be updated every year.

```

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

<reportJobCalendar>
  <calendarType>holiday</calendarType>
  <description>Holiday calendar description</description>
  <excludeDays>
    <excludeDay>2012-03-20</excludeDay>
    <excludeDay>2012-03-21</excludeDay>
    <excludeDay>2012-03-22</excludeDay>
  </excludeDays>
  <timeZone>GMT+03:00</timeZone>
</reportJobCalendar>

```

- Weekly calendar – Defines a set of days to be excluded each week.

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

<reportJobCalendar>
  <calendarType>weekly</calendarType>
  <description>test description</description>
  <excludeDaysFlags>
    <excludeDayFlag>false</excludeDayFlag> <!--SUNDAY-->
    <excludeDayFlag>true</excludeDayFlag> <!--MONDAY-->
    <excludeDayFlag>false</excludeDayFlag> <!--TUESDAY-->
    <excludeDayFlag>true</excludeDayFlag> <!--WEDNESDAY-->
    <excludeDayFlag>false</excludeDayFlag> <!--THURSDAY-->
    <excludeDayFlag>true</excludeDayFlag> <!--FRIDAY-->
    <excludeDayFlag>false</excludeDayFlag> <!--SATURDAY-->
  </excludeDaysFlags>
  <timeZone>GMT+03:00</timeZone>
</reportJobCalendar>
```

- Monthly calendar – Defines the dates to exclude every month.

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

<reportJobCalendar>
  <calendarType>monthly</calendarType>
  <description>Monthly calendar description</description>
  <excludeDaysFlags>
    <excludeDayFlag>true</excludeDayFlag> <!--01-->
    <excludeDayFlag>false</excludeDayFlag> <!--02-->
    <excludeDayFlag>true</excludeDayFlag> <!--03-->
    <excludeDayFlag>false</excludeDayFlag> <!--04-->
    <excludeDayFlag>true</excludeDayFlag> <!--05-->
    <excludeDayFlag>false</excludeDayFlag> <!--06-->
    <excludeDayFlag>true</excludeDayFlag> <!--07-->
    <excludeDayFlag>false</excludeDayFlag> <!--08-->
    <excludeDayFlag>true</excludeDayFlag> <!--09-->
    <excludeDayFlag>false</excludeDayFlag> <!--10-->
    <excludeDayFlag>true</excludeDayFlag> <!--11-->
    <excludeDayFlag>false</excludeDayFlag> <!--12-->
    <excludeDayFlag>false</excludeDayFlag> <!--13-->
    <excludeDayFlag>true</excludeDayFlag> <!--14-->
    <excludeDayFlag>false</excludeDayFlag> <!--15-->
    <excludeDayFlag>false</excludeDayFlag> <!--16-->
    <excludeDayFlag>false</excludeDayFlag> <!--17-->
    <excludeDayFlag>false</excludeDayFlag> <!--18-->
    <excludeDayFlag>false</excludeDayFlag> <!--19-->
    <excludeDayFlag>false</excludeDayFlag> <!--20-->
    <excludeDayFlag>false</excludeDayFlag> <!--21-->
    <excludeDayFlag>false</excludeDayFlag> <!--22-->
    <excludeDayFlag>false</excludeDayFlag> <!--23-->
    <excludeDayFlag>false</excludeDayFlag> <!--24-->
    <excludeDayFlag>false</excludeDayFlag> <!--25-->
```

```

    <excludeDayFlag>>false</excludeDayFlag> <!--26-->
    <excludeDayFlag>>false</excludeDayFlag> <!--27-->
    <excludeDayFlag>>false</excludeDayFlag> <!--28-->
    <excludeDayFlag>>false</excludeDayFlag> <!--29-->
    <excludeDayFlag>>false</excludeDayFlag> <!--30-->
    <excludeDayFlag>>false</excludeDayFlag> <!--31-->
</excludeDaysFlags>
<timeZone>GMT+03:00</timeZone>
</reportJobCalendar>

```

3.9 The v2/queryExecutor Service

In addition to running reports, JasperReports Server exposes queries that you can run through the rest_v2/queryExecutor service. In release 5.1, the only resource that supports queries is a Domain.

Method	URL	
GET	http://<host>:<port>/jasperserver-pro/rest_v2/queryExecutor/path/to/Domain/?q=<query>	
Argument	Type/Value	Description
q	Required String	The query string is a special format that references the fields and measures exposed by the Domain. To write this query, you must have knowledge of the Domain schema that is not available through the REST services. See below.
Options		
accept: application/xml (default) accept: application/json Accept-Language: <locale>, <relativeQualityFactor>; for example en_US, q=0.8;		
Return Value on Success		Typical Return Values on Failure
200 OK – The body contains the data that is the result of the query. See the format of the data below.		404 Not Found – When the specified Domain does not exist.

If the query is too large to fit in the argument in the URL, use the POST method to send it as request content:

Method	URL	
POST	http://<host>:<port>/jasperserver-pro/rest_v2/queryExecutor/path/to/Domain/	
Content-Type	Content	
application/xml	The query string is a special format that references the fields and measures exposed by the Domain. To write this query, you must have knowledge of the Domain schema that is not available through the REST services. See below.	
Options		
accept: application/xml (default) accept: application/json Accept-Language: <locale>, <relativeQualityFactor>; for example en_US, q=0.8;		
Return Value on Success		Typical Return Values on Failure
200 OK – The body contains the data that is the result of the query. See the format of the data below.		404 Not Found – When the specified Domain does not exist.

The following example show the format of a query in XML:

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

And the following sample shows the result of query. In order to optimize the size of the response, rows are presented as sets of values without the column names repeated for each row. The column IDs appear at the top of the result, as shown in the following example. As with the query, the result requires knowledge of the Domain schema to identify the human-readable column names.

```
<queryResult>
  <names>
    <name>expense_join_account.ej_account_account_description</name>
    <name>expense_join_account.ej_expense_fact_account_id</name>
    <name>expense_join_account.ej_account_account_parent</name>
    <name>expense_join_account.ej_account_account_rollup</name>
    <name>expense_join_account.ej_account_account_type</name>
    <name>expense_join_account.ej_account_Custom_Members</name>
    <name>expense_join.ej_expense_fact_amount</name>
    <name>expense_join_store.ej_store_store_type</name>
    <name>expense_join_store.ej_store_store_street_address</name>
    <name>expense_join_store.ej_store_store_city</name>
    <name>expense_join_store.ej_store_store_state</name>
    <name>expense_join_store.ej_store_store_postal_code</name>
    <name>expense_join_store.sample_time</name>
  </names>
  <values>
    <row>
      <value xsi:type="xs:string">Marketing</value>
      <value xsi:type="xs:int">4300</value>
      <value xsi:type="xs:int">4000</value>
      <value xsi:type="xs:string">+</value>
      <value xsi:type="xs:string">Expense</value>
      <value xsi:nil="true"/>
      <value xsi:type="xs:double">1884.0000</value>
      <value xsi:type="xs:dateTime">1997-01-01T04:05:06+02:00</value>
      <value xsi:type="xs:string">HeadQuarters</value>
      <value xsi:type="xs:string">1 Alameda Way</value>
      <value xsi:type="xs:string">Alameda</value>
      <value xsi:type="xs:string">CA</value>
      <value xsi:type="xs:int">94502</value>
      <value xsi:type="xs:string">USA</value>
    </row>
  </values>
</queryResult>
```

```
    <value xsi:type="xs:time">04:05:06+02:00</value>
  </row>
  ...
</values>
</queryResult>
```



Both date-only and timestamp fields are given in the ISO date-time format such as 1997-01-01T04:05:06+02:00.

For database columns that store a time and date that includes a time zone, such as “timestamp with time zone” in PostgreSQL, the result is not guaranteed to be in the same timezone as stored in the database. These dates and times are converted to the server’s time zone.

CHAPTER 4 REST - WEB SERVICES FOR ADMINISTRATION

Only administrative users may access the REST services for administration. For authentication using the REST web services, see section 1.2, “REST Authentication,” on page 12.

The RESTful administration services gives responses that contain the same XML data structure that are used in the SOAP administration web service. These data structures are documented in section 1.6, “Syntax of resourceDescriptor,” on page 17, with reference material in Appendix A, “ResourceDescriptor API Constants,” on page 155.

This chapter includes the following sections:

- **The organization Service**
- **The user Service**
- **The attribute Service**
- **The role Service**
- **The v2/organizations Service**
- **The v2/users Service**
- **The v2/attributes Service**
- **The v2/roles Service**

4.1 The organization Service



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.

In commercial editions of JasperReports Server, the organization service lets you view, create, modify, and delete organizations (also known as tenants).



The rest/organization service is replaced by the rest_v2/organizations service. Jaspersoft recommends using the latest rest_v2 services.

4.1.1 Viewing an Organization

The GET method retrieves information about an organization and optionally its child organizations. To specify an organization, use its ID, not its path.

Method	URL	
GET	http://<host>:<port>/jasperserver-pro/rest/organization/organizationID?<arguments>	
Argument	Type/Value	Description
listSubOrgs?	Boolean	When this argument is omitted or is false, only the specified organization is returned. When true only the suborganizations are returned.
Return Value on Success		Typical Return Values on Failure
200 OK – The content is a descriptor for the organization or its suborganizations.		404 Not Found – When the specified organization ID is not found in the server.

When the `listSubOrgs` argument is omitted or false, the GET method returns a single `tenant` descriptor for the given organization:

```
<tenant>
  <alias>organization_1</alias>
  <id>organization_1</id>
  <parentId>organizations</parentId>
  <tenantDesc> </tenantDesc>
  <tenantFolderUri>/organizations/organization_1</tenantFolderUri>
  <tenantName>Organization</tenantName>
  <tenantNote> </tenantNote>
  <tenantUri>/organization_1</tenantUri>
  <theme>default</theme>
</tenant>
```



The `tenantFolderURI` is always relative to the user ID that authenticated the request. In these two examples, the user ID is `superuser`.

When the `listSubOrgs` argument is true, the GET method returns a list of tenant descriptors. If the given organization has no suborganizations, the list is empty.

```
<tenantsList>
  <tenant>
    <alias>SubOrganization</alias>
    <id>SubOrganization</id>
    <parentId>organization_1</parentId>
    <tenantDesc>My SubOrganization</tenantDesc>
    <tenantFolderUri>/organizations/organization_1/organizations/SubOrganization
      </tenantFolderUri>
    <tenantName>SubOrganization</tenantName>
    <tenantUri>/organization_1/SubOrganization</tenantUri>
    <theme>default</theme>
  </tenant>
</tenantsList>
```

4.1.2 Creating an Organization

Use the PUT method of the organization service to create a new organization.

Method	URL	
PUT	http://<host>:<port>/jasperserver-pro/rest/organization/	
Content-Type	Content	
text/plain	A well-formed <code>tenant</code> descriptor that accurately describes the desired organization.	
Return Value on Success		Typical Return Values on Failure
201 Created		404 Not Found – When the parent organization ID in the descriptor is not found in the server.

4.1.3 Modifying Organization Properties

Use the POST method of the organization service to update the properties of an existing organization.

Method	URL	
POST	http://<host>:<port>/jasperserver-pro/rest/organization/organizationID/	
Content-Type	Content	
text/plain	A well-formed <code>tenant</code> descriptor with updated property values for the desired organization.	
Return Value on Success		Typical Return Values on Failure
200 OK		404 Not Found – When the organization ID is not found in the server.

As with organizations managed in the user interface, only certain fields may be modified in the tenant descriptor:

- `alias` – Can be used for logging in, but must be unique among all organization aliases.
- `tenantDesc` – Description of the organization, visible only to administrators.
- `tenantName` – Display name of the organization, appearing to users on the organization's root folder.
- `theme` – The user interface theme that is active for all organization users.

4.1.4 Deleting an Organization

Use the DELETE method of the organization service to remove an existing organization.

Method	URL	
DELETE	http://<host>:<port>/jasperserver-pro/rest/organization/organizationID/	
Return Value on Success		Typical Return Values on Failure
200 OK		404 Not Found – When the organization ID is not found in the server.

Deleting an organization removes all of its users, roles, and all of its suborganizations recursively.

4.2 The user Service

The GET method for the user service returns descriptors for all users that match the search string. In commercial editions, the scope of the search is the administrator’s organization and all suborganizations. In the community project, there are no organizations, and the scope is all users defined in the server. If no search string is specified, all users are returned. If no users match the search string, the method returns an empty list.

Method	URL
GET	http://<host>:<port>/jasperserver[-proj]/rest/user/<searchString>
Return Value on Success	Typical Return Values on Failure
200 OK – The content is a descriptor for each of the users that match the search.	

The following example shows the descriptors for users that match the search string “joe”:

```
<users>
  <user>
    <enabled>true</enabled>
    <externallyDefined>>false</externallyDefined>
    <fullName>Joe User</fullName>
    <previousPasswordChangeTime>2011-11-29T10:18:38.062-08:00
      </previousPasswordChangeTime>
    <roles>
      <externallyDefined>>false</externallyDefined>
      <roleName>ROLE_USER</roleName>
    </roles>
    <tenantId>organization_1</tenantId>
    <username>joeuser</username>
  </user>
  <user>
    <emailAddress></emailAddress>
    <enabled>true</enabled>
    <externallyDefined>>false</externallyDefined>
    <fullName>joeuser</fullName>
    <previousPasswordChangeTime>2011-11-29T15:52:18.407-08:00
      </previousPasswordChangeTime>
    <roles>
      <externallyDefined>>false</externallyDefined>
      <roleName>ROLE_USER</roleName>
    </roles>
    <tenantId>SubOrganization</tenantId>
    <username>joeuser</username>
  </user>
</users>
```

The descriptor above is from a commercial edition, and each user has a `tenantId` element to indicate which organization the user belongs to. The community project does not have organizations and thus does not specify the `tenantId` element.



The `externallyDefined` property is true when the user is authenticated by a 3rd party such as an LDAP directory or single sign-on mechanism. For more information, see the *JasperReports Server Authentication Cookbook*.

4.2.1 Creating a User

Use the PUT method of the user service to create a new user. In commercial editions, specify the user's organization in the `tenantId` element of the `user` descriptor.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest/user/	
Content-Type	Content	
text/plain	A well-formed <code>user</code> descriptor that describes the properties of the desired user. You can set the following properties on the user account: <ul style="list-style-type: none"> ♦ <code>username</code> – Required. ♦ <code>password</code> – Required. ♦ <code>fullName</code> – Required. ♦ <code>tenantId</code> – Required on commercial editions with multiple organizations. ♦ <code>enabled</code> – Optional. ♦ <code>emailAddress</code> – Optional. ♦ <code>roles</code> – Optional list of roles. The <code>ROLE_USER</code> is assigned automatically. All other properties seen in user descriptors are generated by the server and should not be set.	
Return Value on Success		Typical Return Values on Failure
201 Created		404 Not Found – When the organization ID in the descriptor is not found in the server.

The following example shows a user descriptor for creating a user:

```
<user>
  <username>Alice</username>
  <password>myPassword</password>
  <fullName>Alice Adams</fullName>
  <tenantId>organization_1</tenantId>
  <emailAddress>alice@example.com</emailAddress>
  <roles>
    <roleName>ROLE_DEMO</roleName>
  </roles>
</user>
```

4.2.2 Editing a User

Use the POST method of the user service to update the properties of an existing user.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest/user/<userID>/	
Content-Type	Content	
text/plain	A well-formed <code>user</code> descriptor that describes the properties of the desired user.	
Return Value on Success		Typical Return Values on Failure
200 OK		404 Not Found – When the user ID is not found in the server.

4.2.3 Deleting a User

Use the DELETE method of the user service to remove an existing user.

Method	URL
DELETE	http://<host>:<port>/jasperserver[-proj]/rest/user/<userID>/
Return Value on Success	Typical Return Values on Failure
200 OK	404 Not Found – When the specified user ID is not found in the server.

4.3 The attribute Service

The attribute service lets you view and update profile attributes, which are custom properties associated with a user. This service does not delete attributes in this release.

Method	URL
GET	http://<host>:<port>/jasperserver[-proj]/rest/attribute/<userID>/
Return Value on Success	Typical Return Values on Failure
200 OK – The content is a descriptor for the user attributes.	404 Not Found – When the specified user ID is not found in the server.

The following example show the user attributes specified in an entityResource element:

```
<entityResource>
  <Item xsi:type="profileAttributeImpl"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <attrName>State</attrName>
    <attrValue>CA</attrValue>
  </Item>
  <Item xsi:type="profileAttributeImpl"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <attrName>Cities</attrName>
    <attrValue>San Francisco, Oakland, San Jose</attrValue>
  </Item>
</entityResource>
```

Use the PUT or POST methods of the attribute service to add attributes to a user. For this service, these methods are synonyms.

Method	URL
PUT or POST	http://<host>:<port>/jasperserver[-proj]/rest/attribute/<userID>/
Content-Type	Content
text/plain	A well-formed descriptor that contains the attributes to add to the given user.
Return Value on Success	Typical Return Values on Failure
201 Created	404 Not Found – When the user ID is not found in the server.

The DELETE on the attribute service is not implemented in this release.

4.4 The role Service

The role service allows administrators to view, create, edit, and delete role definitions. However, the role service does not define role membership. To add users to a role, edit the user's properties, as described in 4.2.2, "Editing a User," on page 93.

The GET method of the role service returns descriptors for all roles that match the search string. In commercial editions, the scope of the search is the administrator's organization and all suborganizations. In the community project, there are no organizations, and the scope is all roles defined in the server. If no search string is specified, all roles are returned. If no roles match the search string, the method returns an empty list.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest/role/<searchString>/
Return Value on Success	Typical Return Values on Failure
200 OK – The content is a descriptor for each of the roles that match the search.	

In commercial editions, roles defined in an organization specify its ID in the `tenantID` element. System roles that appear in every organization are listed without a `tenantID` property. The community project does not have organizations and thus does not specify the `tenantId` element on any roles.

```
<roles>
  <role>
    <externallyDefined>false</externallyDefined>
    <roleName>ROLE_ADMINISTRATOR</roleName>
  </role>
  <role>
    <externallyDefined>false</externallyDefined>
    <roleName>ROLE_ANONYMOUS</roleName>
  </role>
  <role>
    <externallyDefined>false</externallyDefined>
    <roleName>ROLE_DEMO</roleName>
  </role>
  <role>
    <externallyDefined>false</externallyDefined>
    <roleName>ROLE_SUPERMART_MANAGER</roleName>
  </role>
  <role>
    <externallyDefined>false</externallyDefined>
    <roleName>ROLE_USER</roleName>
  </role>
  <role>
    <externallyDefined>false</externallyDefined>
    <roleName>ROLE_SAMPLE</roleName>
    <tenantId>organization_1</tenantId>
  </role>
</roles>
```

4.4.1 Creating a New Role

Use the PUT method of the role service to create a new role. In commercial editions, specify the role's organization in the `tenantID` element of the `role` descriptor.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-proj]/rest/role/	
Content-Type	Content	
text/plain	A well-formed <code>role</code> descriptor that describes the properties of the desired role.	
Return Value on Success	Typical Return Values on Failure	
201 Created	404 Not Found – When the organization ID in the descriptor is not found in the server.	

4.4.2 Editing a Role

Use the POST method of the role service to update the properties of an existing role.

Method	URL	
POST	http://<host>:<port>/jasperserver[-proj]/rest/role/<roleName>/	
Content-Type	Content	
text/plain	A well-formed <code>role</code> descriptor that describes the properties of the desired role.	
Return Value on Success	Typical Return Values on Failure	
200 OK	404 Not Found – When the <code>roleName</code> is not found in the server.	

4.4.3 Deleting a Role

Use the DELETE method of the role service to remove an existing role.

Method	URL	
DELETE	http://<host>:<port>/jasperserver[-proj]/rest/role/<roleName>/	
Return Value on Success	Typical Return Values on Failure	
200 OK	404 Not Found – When the specified role is not found in the server.	

4.5 The v2/organizations Service



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.

In commercial editions of JasperReports Server 5.1 and later, the `rest_v2/organizations` service replaces the `rest/organization` service. It provides similar methods that allow you to list, view, create, modify, and delete organizations (also known as tenants). New functionality allows you to search for organizations by name and retrieve hierarchies of organizations.

Because the organization ID is used in the URL, this service can operate only on organizations whose ID is less than 100 characters long and does not contain spaces or special symbols. As with resource IDs, the organization ID is permanent and cannot be modified for the life of the organization.

4.5.1 Searching for Organizations

The GET method without any organization ID searches for organizations by ID, alias, or display name. If no search is specified, it returns a list of all organizations. Searches and listings start from but do not include the logged-in user's organization or the specified base (`rootTenantId`).

Method	URL	
GET	http://<host>:<port>/jasperserver-pro/rest_v2/organizations?<arguments>	
Argument	Type	Description
q	Optional String	Specify a string or substring to match the organization ID, alias, or name of any organization. The search is not case sensitive. Only the matching organizations are returned in the results, regardless of their hierarchy.
includeParents	Optional Boolean	When used with a search, the result will include the parent hierarchy of each matching organization. When not specified, this argument is false by default.
rootTenantId	Optional String	Specifies an organization ID as a base for searching and listing child organizations. The base is not included in the results. Regardless of this base, the <code>tenantFolderURI</code> values in the result are always relative to the logged-in user's organization. When not specified, the default base is the logged-in user's organization.
Options		
accept: application/xml (default) accept: application/json		
Return Value on Success		Typical Return Values on Failure
200 OK – The content is a set of descriptors for all organizations in the result. 204 No Content – The search did not return any organizations.		

The following example shows a search for an organization and its parent hierarchy:

```
GET http://localhost:8080/jasperserver-pro/rest_v2/organizations?q=acc&includeParents=true
```

This request has the following response, as viewed by superuser at the root of the organization hierarchy:

```
<organizations>
  <organization>
    <alias>Finance</alias>
    <id>Finance</id>
    <parentId>organizations</parentId>
    <tenantDesc></tenantDesc>
    <tenantFolderUri>/organizations/Finance</tenantFolderUri>
    <tenantName>Finance</tenantName>
    <tenantUri>/Finance</tenantUri>
    <theme>default</theme>
  </organization>
```

```

<organization>
  <alias>Accounts</alias>
  <id>Accounts</id>
  <parentId>Finance</parentId>
  <tenantDesc></tenantDesc>
  <tenantFolderUri>/organizations/Finance/organizations/Accounts</tenantFolderUri>
  <tenantName>Accounts</tenantName>
  <tenantUri>/Finance/Accounts</tenantUri>
  <theme>default</theme>
</organization>
</organizations>

```

4.5.2 Viewing an Organization

The GET method with an organization ID retrieves a single descriptor containing the list of properties for the organization. When you specify an organization, use its unique ID, not its path.

Method	URL
GET	http://<host>:<port>/jasperserver-pro/rest_v2/organizations/organizationID
Options	
accept: application/xml (default) accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – The content is the descriptor for the given organization.	404 Not Found – When the ID does not match any organization. The content includes an error message. 403 Forbidden – When the logged-in user does not have permission to view the given organization

The organization descriptor is identical to the one returned when searching or listing organization, but only a single descriptor is ever returned. The following example shows the descriptor in JSON format:

```

{
  "id": "Finance",
  "alias": "Finance",
  "parentId": "organizations",
  "tenantName": "Finance",
  "tenantDesc": " ",
  "tenantNote": null,
  "tenantUri": "/Finance",
  "tenantFolderUri": "/organizations/Finance",
  "theme": "default"
}

```

4.5.3 Creating an Organization

To create an organization, put all information in an organization descriptor, and include it in a POST request to the rest_v2/organizations service, with no ID specified in the URL. The organization is created in the organization specified by the parentId value of the descriptor.

Method	URL	
POST	http://<host>:<port>/jasperserver-pro/rest_v2/organizations?<argument>	
Argument	Type	Description
create Default Users	Optional Boolean	Set this argument to false to suppress the creation of default users (joeuser, jasperadmin) in the new organization. When not specified, the default behavior is true and organizations are created with the standard default users.
Content-Type		Content
application/xml application/json		A partial or complete organization descriptor that includes the desired properties for the organization.
Return Value on Success		Typical Return Values on Failure
201 Created – The organization was successfully created using the values in the descriptor or default values if missing.		404 Not Found – When the ID of the parent organization cannot be resolved. 400 Bad Request – When the ID or alias of the new organization is not unique on the server, or when the ID in the description contains illegal symbols. The following symbols are not allowed: id and alias: ~!+-#\$\$%^ tenantName: &*?<>/\

The descriptor sent in the request should contain all the properties you want to set on the new organization. Specify the `parentId` value to set the parent of the organization, not the `tenantUri` or `tenantFolderUri` properties. The following example shows the descriptor in JSON format:

```
{
  "id": "Audit",
  "alias": "Audit",
  "parentId": "Finance",
  "tenantName": "Audit",
  "tenantDesc": "Audit Department of Finance",
  "theme": "default"
}
```

However, all properties have defaults or can be determined based on the alias value. The minimal descriptor necessary to create an organization is simply the alias property. In this case, the organization is created as a child of the logged-in user's home organization. For example, if `superuser` posts the following descriptor, the server creates an organization with the name, ID, and alias of "HR" as a child of the root organization:

```
{
  "alias": "HR"
}
```

4.5.4 Modifying Organization Properties

To modify the properties of an organization, use the PUT method and specify the organization ID in the URL. The request must include an organization descriptor with the values you want to change. You cannot change the ID of an organization, only its name (used for display) and its alias (used for logging in).

Method	URL
PUT	http://<host>:<port>/jasperserver-pro/rest_v2/organizations/organizationID/
Content-Type	Content
application/xml application/json	A partial organization descriptor that includes the properties to change. Do not specify the following properties: <ul style="list-style-type: none"> ♦ id – The organization ID is permanent and can never be modified. ♦ parentId – Organizations cannot change parents. ♦ tenantUri – Organizations cannot change the organization hierarchy. ♦ tenantFolderUri – The organization folder is automatically based on its parent, which cannot be changed.
Return Value on Success	Typical Return Values on Failure
200 OK – The organization was successfully updated.	400 Bad Request – When some dependent resources cannot be resolved.

The following example shows a descriptor sent to update the name and description of an organization:

```
{
  "tenantName": "Audit Dept",
  "tenantDesc": "Audit Department of Finance Division"
}
```

4.5.5 Setting the Theme of an Organization

A theme determines how the JasperReports Server interface appears to users. Administrator can create and set different themes for each organization. To set a theme through web services, use the PUT method of the REST v2/organizations service to modify the corresponding property of the desired organization.

For example:

PUT http://localhost:8080/jasperserver-pro/rest_v2/organizations/Audit

```
{
  "theme": "jasper_dark"
}
```

For more information about themes, see the *JasperReports Server Administrator Guide*.

4.5.6 Deleting an Organization

To delete an organization, use the DELETE method and specify the organization ID in the URL. When deleting an organization, all of its resources in the repository, all of its sub-organizations, all of its users, and all of its roles are permanently deleted.

Method	URL
DELETE	http://<host>:<port>/jasperserver-pro/rest_v2/organizations/organizationID/
Return Value on Success	Typical Return Values on Failure
204 No Content – The organization was successfully deleted.	400 Bad Request – When attempting to delete the organization of the logged-in user. 404 Not Found – When the ID of the organization cannot be resolved.

4.6 The v2/users Service

The `rest_v2/users` service replaces the `rest/user` service. It provides similar methods that allow you to list, view, create, modify, and delete user accounts, including setting role membership. The new service provides improved search functionality, such as organization-based searches in commercial editions licensed to use organizations. Every method has two URL forms, one with an organization ID and one without.

Because the user ID and organization ID are used in the URL, this service can operate only on users and organizations whose ID is less than 100 characters long and does not contain spaces or special symbols. As with resource IDs, the user ID is permanent and cannot be modified for the life of the user account.

4.6.1 Searching for Users

The GET method without any user ID searches for and lists user accounts. It has options to search for users by name or by role. If no search is specified, it returns all users. The method has two forms:

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL without an organization ID.
- In commercial editions with organizations, use the first URL to list all users starting from the logged-in user's organization (root for the system admin), and use the second URL to list all users in a specified organization.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/users?<arguments> http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users?<arguments>	
Argument	Type	Description
search	Optional String	Specify a string or substring to match the user ID or full name of any user. The search is not case sensitive.
requiredRole	Optional String	Specify a role name to list only users with this role. Repeat this argument to filter with multiple roles. In commercial editions with multiple organizations, specify roles as <roleName>%7C<orgID> (%7C is the character).
hasAll Required Roles	Optional Boolean	When set to false with multiple requiredRole arguments, users will match if they have any of the given roles (OR operation). When true or not specified, users must match all of the given roles (AND operation).
include SubOrgs	Optional Boolean	Limits the scope of the search or list in commercial editions with multiple organizations. When set to false, the first URL form is limited to the logged-in user's organization, and the second URL form is limited to the organization specified in the URL. When true or not specified, the scope includes the hierarchy of all child organizations.
Options		
accept: application/xml (default)		
accept: application/json		
Return Value on Success		Typical Return Values on Failure
200 OK – The content is a set of descriptors for all users in the result.		404 Not Found – When the organization ID does not match any organization. The content includes an error message.
204 No Content – The search did not return any users.		

The following example shows the first form of the URL on a community edition server:

```
GET http://localhost:8080/jasperserver/rest_v2/users?search=j
```

The response is a set of summary descriptors for all users containing the string “j”:

```
<users>
  <user>
    <externallyDefined>false</externallyDefined>
    <fullName>jasperadmin User</fullName>
    <username>jasperadmin</username>
  </user>
  <user>
    <externallyDefined>false</externallyDefined>
    <fullName>Joe User</fullName>
    <username>joeuser</username>
  </user>
</users>
```

The next example shows the second form of the URL on a commercial edition server with multiple organizations:

GET http://localhost:8080/jasperserver/rest_v2/organizations/Finance/users

On servers with multiple organizations, the summary user descriptors include the organization (tenant) ID:

```
<users>
  <user>
    <externallyDefined>false</externallyDefined>
    <fullName>jasperadmin</fullName>
    <tenantId>Finance</tenantId>
    <username>jasperadmin</username>
  </user>
  <user>
    <externallyDefined>false</externallyDefined>
    <fullName>jasperadmin</fullName>
    <tenantId>Audit</tenantId>
    <username>jasperadmin</username>
  </user>
  <user>
    <externallyDefined>false</externallyDefined>
    <fullName>joeuser</fullName>
    <tenantId>Finance</tenantId>
    <username>joeuser</username>
  </user>
  <user>
    <externallyDefined>false</externallyDefined>
    <fullName>joeuser</fullName>
    <tenantId>Audit</tenantId>
    <username>joeuser</username>
  </user>
</users>
```

4.6.2 Viewing a User

The GET method with a user ID (username) retrieves a single descriptor containing the full list of user properties and roles.

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL.

- In commercial editions with organizations, use the second URL to specify the user's organization. When specifying the organization, use its unique ID, not its path. When logged in as the system admin (`superuser`), use the first URL to specify users of the root organization.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID
Options	
accept: application/xml (default) accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – The content is the descriptor for the given user.	404 Not Found – When the user ID or organization ID does not match any user or organization. The content includes an error message.

The full user descriptor includes detailed information about the user account, including any roles. The following example shows the descriptor in XML format:

GET `http://localhost:8080/jasperserver/rest_v2/users/joeuser`

```
<user>
  <enabled>true</enabled>
  <externallyDefined>false</externallyDefined>
  <fullName>Joe User</fullName>
  <previousPasswordChangeTime>2013-04-19T18:53:07.602-07:00
</previousPasswordChangeTime>
  <roles>
    <role>
      <externallyDefined>false</externallyDefined>
      <name>ROLE_USER</name>
    </role>
  </roles>
  <username>joeuser</username>
</user>
```

In servers with multiple organizations, the full descriptor includes the organization (tenant) ID. The following example shows the descriptor in JSON format:

GET `http://localhost:8080/jasperserver/rest_v2/organizations/Finance/users/joeuser`

```
{
  "fullName": "joeuser",
  "emailAddress": "",
  "externallyDefined": false,
  "enabled": true,
  "previousPasswordChangeTime": 1366429181984,
  "tenantId": "Finance",
  "username": "joeuser",
  "roles": [
    { "name": "ROLE_USER", "externallyDefined": false }
  ]
}
```

4.6.3 Creating a User

To create a user account, put all required information in a user descriptor, and include it in a PUT request to the `rest_v2/users` service, with the intended user ID (username) specified in the URL.

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL.
- In commercial editions with organizations, use the second URL to specify the user’s organization. When specifying the organization, use its unique ID, not its path. When logged in as the system admin (`superuser`), use the first URL to create users in the root organization.

To create a user, the user ID in the URL must be unique on the server or in the organization. If the user ID already exists, that user account will be modified, as described in section 4.6.4, “[Modifying User Properties](#),” on page 104.

Method	URL
PUT	<a href="http://<host>:<port>/jasperserver[-proj]/rest_v2/users/userID">http://<host>:<port>/jasperserver[-proj]/rest_v2/users/userID <a href="http://<host>:<port>/jasperserver[-proj]/rest_v2/organizations/orgID/users/userID">http://<host>:<port>/jasperserver[-proj]/rest_v2/organizations/orgID/users/userID
Content-Type	Content
application/xml application/json	A user descriptor that includes at least the <code>fullName</code> and <code>password</code> for the user. The role <code>ROLE_USER</code> is automatically assigned to all users, so it does not need to be specified. Do not specify the following properties: <ul style="list-style-type: none"> • <code>username</code> – Specified in the URL and cannot be modified in the descriptor. • <code>tenantID</code> – Specified in the URL and cannot be modified in the descriptor. • <code>externallyDefined</code> – Computed automatically by the server. • <code>previousPasswordChangeTime</code> – Computed automatically by the server.
Return Value on Success	Typical Return Values on Failure
201 Created – The user was successfully created using the values in the descriptor. The response contains the full descriptor of the new user.	404 Not Found – When the organization ID cannot be resolved.

The descriptor sent in the request should contain all the properties you want to set on the new user, except for the username that is specified in the URL. To set roles on the user, specify them as a list of roles. The following example shows the descriptor in JSON format:

```

{
  "fullName": "Joe User",
  "emailAddress": "juser@example.com",
  "enabled": false,
  "password": "mySecretPassword",
  "roles": [
    { "name": "ROLE_MANAGER" } ]
}
    
```

4.6.4 Modifying User Properties

To modify the properties of a user account, put all desired information in a user descriptor, and include it in a PUT request to the `rest_v2/users` service, with the existing user ID (username) specified in the URL.

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL.
- In commercial editions with organizations, use the second URL to specify the user’s organization. When specifying the organization, use its unique ID, not its path. When logged in as the system admin (`superuser`), use the first URL to modify users of the root organization.

To modify a user, the user ID in the URL must already exist on the server or in the organization. If the user ID doesn’t exist, a user account will be created, as described in section 4.6.3, “[Creating a User](#),” on page 104.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID
Content-Type	Content
application/xml application/json	A user descriptor that includes the properties you want to change. Do not specify the following properties: <ul style="list-style-type: none"> • username – Specified in the URL and cannot be modified in the descriptor. • tenantID – Specified in the URL and cannot be modified in the descriptor. • externallyDefined – Computed automatically by the server. • previousPasswordChangeTime – Computed automatically by the server.
Return Value on Success	Typical Return Values on Failure
200 OK – The user properties were successfully updated.	404 Not Found – When the organization ID cannot be resolved.

To add a role to the user, specify the entire list of roles with the desired role added. To remove a role from a user, specify the entire list of roles with the desired role removed. The following example shows the descriptor in JSON format:

```
{
  "enabled": true,
  "password": "newPassword",
  "roles": [
    { "name": "ROLE_USER" }
    { "name": "ROLE_STOREMANAGER" }
  ]
}
```

4.6.5 Deleting a User

To delete a user, send the DELETE method and specify the user ID in the URL.

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL.
- In commercial editions with organizations, use the second URL to specify the user's organization. When specifying the organization, use its unique ID, not its path. When logged in as the system admin (*superuser*), use the first URL to delete users of the root organization.

When this method is successful, the user is permanently deleted.

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID
Return Value on Success	Typical Return Values on Failure
204 No Content – The user was successfully deleted.	404 Not Found – When the ID of the organization cannot be resolved.

4.7 The v2/attributes Service

Attributes, also called profile attributes, are name-value pairs associated with a user. Certain advanced features such as Domain security and OLAP access grants use profile attributes in addition to roles to grant certain permissions. Unlike roles, attributes are not pre-defined, and thus any attribute name can be assigned any value at any time.

The `rest_v2/attributes` service replaces the `rest/attribute` service. It provides methods for reading, writing, and deleting attributes on any given user account. All attribute operations apply to a single specific user; there are no operations for reading or searching attributes from multiple users.

Because the user ID and organization ID are used in the URL, this service can operate only on users and organizations whose ID is less than 100 characters long and does not contain spaces or special symbols. In addition, both attribute names and attribute values being written with this service are limited to 255 characters and may not be empty (null) or contain only whitespace characters.

4.7.1 Viewing User Attributes

The GET method of the attributes service retrieves the list of attributes, if any, defined for the user.

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL.
- In commercial editions with organizations, you must use the second URL to specify the user’s organization. When specifying the organization, use its unique ID, not its path. When logged in as the system admin (`superuser`), use the first URL to specify users of the root organization.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID/attributes?<arguments> http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID/attributes? <arguments>	
Argument	Type	Description
name	Optional String	Specify an attribute name to list the value of that specific attribute. Repeat this argument to view multiple attributes. When this argument is omitted, all attributes and their values are returned for the given user.
Options		
accept: application/xml (default) accept: application/json		
Return Value on Success		Typical Return Values on Failure
200 OK – The content is the list of attributes for the given user. 204 No Content – The search did not return any attributes or the user has no attributes.		404 Not Found – When the user ID or organization ID does not match any user or organization. The content includes an error message.

The list of attributes includes the name and value of each attribute. Each attribute may only have one value, however that value may contain a comma-separated list that is interpreted by the server as being multi-valued. The following example shows attributes in JSON format:

GET http://localhost:8080/jasperserver/rest_v2/users/joeuser/attributes

```
{
  "attribute": [
    {
      "name": "Attr1",
      "value": "Value1a, Value1b, Value1c"
    },
    ...
    {
      "name": "AttrN",
      "value": "ValueN"
    }
  ]
}
```

An alternative syntax exists to read a single attribute by specifying its name in the URL:

Method	URL
GET	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID/attributes/attrName">http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID/attributes/attrName <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID/attributes/attrName">http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID/attributes/attrName
Options	
accept: application/xml (default) accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – The content is a single attributes for the given user.	404 Not Found – When the user ID or organization ID does not match any user or organization. The content includes an error message.

The response is a single attribute name-value pair. The following example shows an attribute in JSON format:

GET http://localhost:8080/jasperserver/rest_v2/users/joeuser/attributes/Attr2

```
{
  "name": "Attr2",
  "value": "Value2"
}
```

4.7.2 Setting User Attributes

The PUT method of the attributes service adds or replaces attributes on the specified user.

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL.
- In commercial editions with organizations, you must use the second URL to specify the user's organization. When specifying the organization, use its unique ID, not its path. When logged in as the system admin (*superuser*), use the first URL to specify users of the root organization.

There are two syntaxes, the following one is for adding or replacing all attributes

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID/attributes http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID/attributes
Content-Type	Content
application/xml application/json	An attribute descriptor that includes the new list of attributes. All previously defined attributes are replaced by this new list.
Return Value on Success	Typical Return Values on Failure
201 Created – When the attributes were successfully created for the user. 200 OK – When the attributes were successfully updated.	404 Not Found – When the user ID or organization ID does not match any user or organization. The content includes an error message. 400 Bad Request – When an attribute name or value is null, blank, or too long. If one attribute causes an error, the operation stops and returns an error, but attributes that were already processed remain in their new state.

The list of attributes defines the name and value of each attribute. Each attribute may only have one value, however, that value may contain a comma separated list that is interpreted by the server as being multi-valued.

```

{
  "attribute": [
    {
      "name": "Attr1",
      "value": "newValue1"
    },
    {
      "name": "Attr2",
      "value": "newValue2a, newValue2b"
    }
  ]
}
    
```

The second syntax of the PUT attributes method is for adding or replacing individual attributes.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID/attributes/attrName http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID/attributes/ attrName
Content-Type	Content
application/xml application/json	A single attribute name-value pair. The attribute name must match the <i>attrName</i> exactly as it appears in the URL. If this attribute name already exists on the specified user, this attribute's value is updated. If the attribute does not exist, it is added to the user's list of attributes.
Return Value on Success	Typical Return Values on Failure
201 Created – When the attribute was successfully created for the user. 200 OK – When the attribute was successfully updated.	404 Not Found – When the user ID or organization ID does not match any user or organization. The content includes an error message.

The content in the request is a single attribute, for example:

PUT http://localhost:8080/jasperserver/rest_v2/users/joeuser/attributes/Attr3

```
{
  "name": "Attr3",
  "value": "NewValue3"
}
```

4.7.3 Deleting User Attributes

The DELETE method of the attributes service removes attributes from the specified user. When attributes are removed, both the name and the value of the attribute are removed, not only the value.

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL.
- In commercial editions with organizations, you must use the second URL to specify the user's organization. When specifying the organization, use its unique ID, not its path. When logged in as the system admin (*superuser*), use the first URL to specify users of the root organization.

There are two syntaxes; the following one is for deleting multiple attributes or all attributes at once.

Method	URL	
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID/attributes?<arguments> http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID/attributes?<arguments>	
Argument	Type	Description
name	Optional String	Specify an attribute name to remove that attribute from the user. Repeat this argument to delete multiple attributes. When this argument is omitted, all attributes are deleted from the given user.
Return Value on Success		Typical Return Values on Failure
204 No Content – The attributes were successfully removed from the user.		404 Not Found – When the user ID or organization ID does not match any user or organization. The content includes an error message. 400 Bad Request – When an attribute name is null, blank, or too long. If one attribute causes an error, the operation stops and returns an error, but attributes that were already deleted remain deleted.

The second syntax deletes a single attribute named in the URL from the specified user.

Method	URL	
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID/attributes/attrName http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID/attributes/attrName	
Return Value on Success		Typical Return Values on Failure
204 No Content – The attribute was successfully removed from the user.		404 Not Found – When the user ID or organization ID does not match any user or organization. The content includes an error message. 400 Bad Request – When an attribute name is null, blank, or too long.

4.8 The v2/roles Service

The `rest_v2/roles` service replaces the `rest/role` service. It provides similar methods that allow you to list, view, create, modify, and delete roles. The new service provides improved search functionality, including user-based role searches. Every method has two URL forms, one with an organization ID and one without.

Because the role ID and organization ID are used in the URL, this service can operate only on roles and organizations whose ID is less than 100 characters long and does not contain spaces or special symbols. Unlike resource IDs, the role ID is the role name and can be modified.

4.8.1 Searching for Roles

The GET method without any role ID searches for and lists role definitions. It has options to search for roles by name or by user that belong to the role. If no search is specified, it returns all roles. The method has two forms:

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL without an organization ID.
- In commercial editions with organizations, use the first URL to search or list all roles starting from the logged-in user's organization (root for the system admin), and use the second URL to search or list all roles in a specified organization.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/roles?<arguments> http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/roles?<arguments>	
Argument	Type	Description
search	Optional String	Specify a string or substring to match the role ID of any role. The search is not case sensitive.
user	Optional String	Specify a username (ID) to list the roles to which this user belongs. Repeat this argument to list all roles of multiple users. In commercial editions with multiple organizations, specify users as <userID>%7C<orgID> (%7C is the character).
hasAllUsers	Optional Boolean	When set to true with multiple user arguments, this method returns only the roles to which all specified users belong (intersection of users' roles). When false or not specified, all roles of all users are found (union of users' roles).
includeSubOrgs	Optional Boolean	Limits the scope of the search or list in commercial editions with multiple tenants. When set to false, the first URL form is limited to the logged-in user's organization, and the second URL form is limited to the organization specified in the URL. When true or not specified, the scope includes the hierarchy of all child organizations.
Options		
accept: application/xml (default) accept: application/json		
Return Value on Success		Typical Return Values on Failure
200 OK – The content is a set of descriptors for all roles in the result. 204 No Content – The search did not return any roles.		404 Not Found – When the organization ID does not match any organization. The content includes an error message.

The following example shows the first form URL on a commercial edition server with multiple organizations:

GET http://localhost:8080/jasperserver/rest_v2/roles

This method returns the set of all default system and root roles defined on a server with the sample data (no organization roles have been defined yet):

```
<roles>
  <role>
    <externallyDefined>false</externallyDefined>
    <name>ROLE_ADMINISTRATOR</name>
  </role>
  <role>
    <externallyDefined>false</externallyDefined>
    <name>ROLE_ANONYMOUS</name>
  </role>
  <role>
    <externallyDefined>false</externallyDefined>
    <name>ROLE_DEMO</name>
  </role>
  <role>
    <externallyDefined>false</externallyDefined>
    <name>ROLE_PORTLET</name>
  </role>
  <role>
    <externallyDefined>false</externallyDefined>
    <name>ROLE_SUPERMART_MANAGER</name>
  </role>
  <role>
    <externallyDefined>false</externallyDefined>
    <name>ROLE_SUPERUSER</name>
  </role>
  <role>
    <externallyDefined>false</externallyDefined>
    <name>ROLE_USER</name>
  </role>
</roles>
```



The `externallyDefined` property is true when the role is synchronized from a 3rd party such as an LDAP directory or single sign-on mechanism. For more information, see the *JasperReports Server Authentication Cookbook*.

4.8.2 Viewing a Role

The GET method with a role ID retrieves a single role descriptor containing the role properties.

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL.
- In commercial editions with organizations, use the second URL to specify the role's organization. When specifying the organization, use its unique ID, not its path. When logged in as the system admin (`superuser`), use the first URL to specify roles of the root organization.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/roles/roleID http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/roles/roleID
Options	
accept: application/xml (default) accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – The content is the descriptor for the given role.	404 Not Found – When the role ID or organization ID does not match any role or organization. The content includes an error message.

After adding roles to an organization, the following example shows the simple role descriptor for an organization role in JSON format:

GET http://localhost:8080/jasperserver-pro/rest_v2/organizations/Finance/roles/ROLE_MANAGER

```
{
  "name": "ROLE_MANAGER",
  "externallyDefined": false,
  "tenantId": "Finance"
}
```

4.8.3 Creating a Role

To create a role, send the PUT request to the rest_v2/roles service with the intended role ID (name) specified in the URL.

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL.
- In commercial editions with organizations, use the second URL to specify the user’s organization. When specifying the organization, use its unique ID, not its path. When logged in as the system admin (*superuser*), use the first URL to create roles in the root organization.

Roles do not have any properties to specify other than the role ID, but the request must include a descriptor that can be empty.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/roles/roleID http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/roles/roleID
Content-Type	Content
application/xml application/json	An empty role descriptor, either <role></role> or {}. Do <i>not</i> specify the following properties: <ul style="list-style-type: none"> ♦ name – Specified in the URL and should not be modified in the descriptor. ♦ tenantID – Specified in the URL and cannot be modified in the descriptor. ♦ externallyDefined – Computed automatically by the server.
Return Value on Success	Typical Return Values on Failure
201 Created – The role was successfully created. The response contains the full descriptor of the new role.	404 Not Found – When the organization ID cannot be resolved.

4.8.4 Modifying a Role

To change the name of a role, send a PUT request to the rest_v2/roles service and specify the new name in the role descriptor.

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL.

- In commercial editions with organizations, use the second URL to specify the user's organization. When specifying the organization, use its unique ID, not its path. When logged in as the system admin (*superuser*), use the first URL to modify roles in the root organization.

The only property of a role that you can modify is the role's name. After the update, all members of the role are members of the new role name, and all permissions associated with the old role name are updated to the new role name.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/roles/roleID http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/roles/roleID
Content-Type	Content
application/xml application/json	A role descriptor containing a single property: <ul style="list-style-type: none"> • <code>name</code> – The new name for the role.
Return Value on Success	Typical Return Values on Failure
200 OK – The role was successfully updated. The response contains the full descriptor of the updated role.	404 Not Found – When the organization ID cannot be resolved.

4.8.5 Setting Role Membership

To assign role membership to a user, set the roles property on the user account with the PUT method of the `rest_v2/users` service. For details, see section 4.6.4, “[Modifying User Properties](#),” on page 104.

4.8.6 Deleting a Role

To delete a role, send the DELETE method and specify the role ID (name) in the URL.

- In the community edition of the server, or commercial editions without organizations, use the first form of the URL.
- In commercial editions with organizations, use the second URL to specify the user's organization. When specifying the organization, use its unique ID, not its path. When logged in as the system admin (*superuser*), use the first URL to delete roles of the root organization.

When this method is successful, the role is permanently deleted.

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/roles/roleID http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/roles/roleID
Return Value on Success	Typical Return Values on Failure
204 No Content – The role was successfully deleted.	404 Not Found – When the ID of the organization cannot be resolved.

CHAPTER 5 SOAP - REPOSITORY WEB SERVICE

The repository web service is comprised of seven methods: `list`, `get`, `put`, `move`, `copy`, `delete`, and `runReport`.

You can retrieve the WSDL (Web Services Description Language) document that describes the repository service by invoking the URL of the service and appending the string `?wsdl`. For example:

```
http://localhost:8080/jasperserver-pro/services/repository?wsdl
```

This chapter contains the following sections:

- **Request and Operation Result**
- **List Operation**
- **Get Operation**
- **Put Operation**
- **Delete Operation**
- **Move Operation**
- **Copy Operation**
- **runReport Operation**
- **Errors**
- **Implementation Suggestions**

5.1 Request and Operation Result

The repository web services operation takes a single input parameter of type `String`. This XML document represents the request. The following shows its DTD:

```
<!ELEMENT request (argument*, resourceDescriptor?)>
<!ATTLIST request
  operationName (get | list | put | runReport) "list"
  locale #IMPLIED
>
<!ELEMENT argument (#PCDATA)>
<!ATTLIST argument
  name CDATA #REQUIRED
>
```

A request is a very simple document that contains:

- The operation to execute (`list`, `get`, `put`, `delete`, or `runReport`).
- A set of optional arguments. Each argument is a pair of a key and a value that is used to achieve very particular results; arguments are only used rarely.
- A resource descriptor.

The operation name is redundant, since the operation to execute is intrinsic in the invoked service. However, including the name can clarify the request document.

The services act on a single resource at time. The resource that is the subject of the request is described by a `resourceDescriptor`.

To get error messages in a particular locale supported by the server, specify the locale code with the `locale` attribute. Locale codes are in the form `<language code>[_<country>[_<variant>]]`. Valid examples include `en_US`, `it_IT`, `fr_FR`, `de_DE`, `ja_JP`, and `es_ES`. For a list of Java-compliant locales, refer to Sun's Java web site.

The following sample request lists the repository root:

```
<?xml version="1.0" encoding="UTF-8"?>
<request operationName="list" locale="en">
  <resourceDescriptor name="" wsType="folder" uriString="/">
    <label>null</label>
  </resourceDescriptor>
</request>
```

Executing a service produces the `operationResult` in the form of a new XML document.

The DTD is very simple:

```
<!ELEMENT operationResult (code, message?, resourceDescriptor*)>
<!ATTLIST operationResult
  version NMTOKEN #REQUIRED
>
<!ELEMENT code (#PCDATA)>
<!ELEMENT message (#PCDATA)>
```

The operation result contains a return code, an optional return message, and zero or more resource descriptors. A return code other than 0 indicates an error, which is normally described in the message tag.

The operation result always includes the `version` attribute: it can be used to detect the server version. For example, you can list the repository root and read the version set by the server in the response. In this case, we aren't interested in the root folder's content. We just want the version information from the response object itself.

The operation result of such a request is:

```
<operationResult version="1.2.1">
  <returnCode>0</returnCode>
  ...
  several resource descriptors...
  ...
</operationResult>
```

5.2 List Operation

This service lists the contents of the specified folder or report unit. The following sample request lists the contents of the /ContentFiles folder in the repository:

```
<request operationName="list" locale="en">
  <resourceDescriptor name="" wsType="folder" uriString="/ContentFiles" isNew=
    "false">
    <label>null</label>
  </resourceDescriptor>
</request>
```

Sample response:

```
<operationResult version="1.2.0">
  <returnCode>0</returnCode>
  <resourceDescriptor name="html" wsType="folder" uriString="/ContentFiles/html"
    isNew="false">
    <label>html</label>
    <resourceProperty name="PROP_RESOURCE_TYPE">
      <value>com.jaspersoft.jasperserver.api.metadata.common.domain.Folder</value>
    </resourceProperty>
    <resourceProperty name="PROP_PARENT_FOLDER">
      <value>/ContentFiles</value>
    </resourceProperty>
    <resourceProperty name="PROP_VERSION">
      <value>0</value>
    </resourceProperty>
  </resourceDescriptor>
  <resourceDescriptor name="pdf" wsType="folder" uriString="/ContentFiles/pdf"
    isNew="false">
    <label>pdf</label>
    <resourceProperty name="PROP_RESOURCE_TYPE">
      <value>com.jaspersoft.jasperserver.api.metadata.common.domain.Folder</value>
    </resourceProperty>
    <resourceProperty name="PROP_PARENT_FOLDER">
      <value>/ContentFiles</value>
    </resourceProperty>
    <resourceProperty name="PROP_VERSION">
      <value>0</value>
    </resourceProperty>
  </resourceDescriptor>
  <resourceDescriptor name="xls" wsType="folder" uriString="/ContentFiles/xls"
    isNew="false">
    <label>xls</label>
    <resourceProperty name="PROP_RESOURCE_TYPE">
      <value>com.jaspersoft.jasperserver.api.metadata.common.domain.Folder</value>
    </resourceProperty>
    <resourceProperty name="PROP_PARENT_FOLDER">
      <value>/ContentFiles</value>
    </resourceProperty>
```

```
<resourceProperty name="PROP_VERSION">
  <value>0</value>
</resourceProperty>
</resourceDescriptor>
</operationResult>
```

When it lists a folder, the repository web service returns a set of resource descriptors: one for each resource that resides in the specified folder. Use / as the URI of the root folder.

Similarly, when it lists a report unit, the repository web service returns a set of resource descriptors that contain (at a minimum) the main JRXML source file. Since a resource in a report unit can be either a local resource or a reference to another repository resource, you should keep a few details in mind:

- If a report unit data source is not defined locally, its `wsType` is set to `datasource`, which does not indicate the exact nature of the resource. Its `type` should simply be `reference`, but since the data source used by the report unit is a special child resource, it's easy to recognize. The URI of the referenced resource is available in the `PROP_REFERENCE_URI` property.
- The main JRXML resource's `wsType` is always set to `jrxml`, even if it's a reference to an external JRXML resource. By looking at the `PROP_IS_REFERENCE` and `PROP_REFERENCE_URI` properties, you can determine where the resource is actually stored. The `PROP_RU_IS_MAIN_REPORT` property identifies the main JRXML source file of the report unit, even if the order of its children is altered.
- The purpose of listing a report unit is to get the list of the resources contained in the report unit. To retrieve the entire report unit (report unit resource as well as its children) at the same time, use the `get` service.

The following Java sample illustrates `wsclient` as an instance of `com.jaspersoft.jasperserver.irplugin.wsclient.WSClient`:

```
ResourceDescriptor rd = new ResourceDescriptor();
rd.setWsType( ResourceDescriptor.TYPE_FOLDER );
rd.setUriString("/");
List lst = wsclient.list(rd);
```

PHP sample:

```
$result = ws_list("/");
if (get_class($result) == 'SOAP_Fault')
{
  $errorMessage = $result->getFault()->faultstring;
}
else
{
  $folders = getResourceDescriptors($result);
}
```

This PHP sample uses the `client.php` file found in the PHP sample provided with JasperReports Server. This file defines the most important constants you may find useful when integrating with the JasperReports Server web services, as well as useful functions that wrap the `list`, `get`, and the `runReport` operations.

The `list` operation also provides a shortcut to get the list of all resources of a given type in the repository, for example all the reports. This use of the `list` operation has the following syntax:

```
<request operationName="list">
  <argument name="LIST_RESOURCES"/>
  <argument name="RESOURCE_TYPE">reportUnit</argument>
  <argument name="PARENT_DIRECTORY">/reports</argument>
</request>
```

```

or
<request operationName="list">
  <argument name="LIST_RESOURCES"/>
  <argument name="RESOURCE_TYPE">reportUnit</argument>
  <argument name="START_FROM_DIRECTORY">/reports</argument>
</request>

```

No value is needed for the `LIST_RESOURCES` argument. The value of the `RESOURCE_TYPE` argument can be any value of `wsType` except `folder`. The `PARENT_DIRECTORY` argument is the name of folder in which you want to look for resources. If you want to look for the resources in a branch of the repository, use the `START_FROM_DIRECTORY` argument.



Using `LIST_RESOURCES` is the only case in which a request doesn't require a resource descriptor.

Several Java methods in `com.jaspersoft.jasperserver.irplugin.wsclient.WSClient` use `LIST_RESOURCES`:

- `list(String xmlRequest)` - Sends any custom request, including one using `LIST_RESOURCES` as shown above.
- `listResources(String type)` - Lists all resources of the given type in the repository visible to the logged in user.
- `listResourcesInFolder(String type, String parentFolder)` - Lists resources of the given type in the folder.
- `listResourcesUnderFolder(String type, String ancestorFolder)` - Lists resources of the given type in the folder and the entire tree beneath that folder.

5.3 Get Operation

The `get` operation is used to obtain information about a resource. In the case of file resources, such as images, fonts, JRXML files, and JAR files, the resource file is attached to the response message.

This method's behavior differs according to the type of object specified:

- Generally, a simple resource descriptor is returned.
- If you get a resource file, the file content is attached to the response; if you do not want the server to attach files to the response, set the request's `NO_ATTACHMENT` argument to `true`.
- If you get a report unit, all the related resources are added as child resource descriptors to the report unit descriptor.
- To get an input control that is based on a query, you must set the `IC_GET_QUERY_DATA` argument to the valid URI of a datasource for the control, or you can handle the `NONE` condition as in the sample code `java-webapp-sample`.

If you set the datasource in the input control, that datasource is used to execute the query and populate the resource descriptor. This can be useful when the input control must be rendered (for example, on a web page) in order to capture a value to pass when executing a report.

- You can use parameters in the input control to select query values, including the datasource. See the examples starting on [page 127](#).

The following sample request gets a file resource:

```

<request operationName="get" locale="en">
  <resourceDescriptor name="JRLogo" wsType="img" uriString="/images/JRLogo"
  isNew="false">
    <label>JR logo</label>
    <description>JR logo</description>
  </resourceDescriptor>
</request>

```

The service only uses the `uriString` to identify the resource to get and check for access permissions. This means that other information present in the resource description (such as resource properties, label, and description) are not actually used or required.

If a file is attached to the response, the returned resource descriptor has the `PROP_HAS_DATA` property set to `true`. By default, the attachments format is MIME. You can use DIME attachments by specifying the `USE_DIME_ATTACHMENTS` argument in the request.

A `get` call always returns a resource descriptor. If the specified resource is not found, or the specified user cannot access it, an error with code 2 is returned.

Java sample:

```
String imgUri = "/images/JRLogo";
ResourceDescriptor rdis = new ResourceDescriptor();
rdis.setParentFolder("/images");
rdis.setUriString(imgUri);

ResourceDescriptor result = wsclient.get(rdis, null);
```

PHP sample:

```
$result = ws_get($someInputControlUri, array( IC_GET_QUERY_DATA =>
$someDatasourceUri ) );
```

The resource descriptor of an input control that includes data obtained by setting the `IC_GET_QUERY_DATA` argument to `true` would be similar to the following XML:

```
<resourceDescriptor name="TEST_LIST" wsType="inputControl" uriString=
"/MyInputControls/TEST_LIST" isNew="false">
  <label>My test list</label>
  <description>My test list</description>
  <resourceProperty name="PROP_RESOURCE_TYPE">
    <value>com.jaspersoft.jasperserver.api.metadata.common.domain.InputControl</
    value>
  </resourceProperty>
  <resourceProperty name="PROP_PARENT_FOLDER">
    <value>/MyInputControls</value>
  </resourceProperty>
  <resourceProperty name="PROP_VERSION">
    <value>6</value>
  </resourceProperty>
  <resourceProperty name="PROP_HAS_DATA">
    <value>>false</value>
  </resourceProperty>
  <resourceProperty name="PROP_IS_REFERENCE">
    <value>>false</value>
  </resourceProperty>
  <resourceProperty name="PROP_INPUTCONTROL_IS_MANDATORY">
    <value>>true</value>
  </resourceProperty>
  <resourceProperty name="PROP_INPUTCONTROL_IS_READONLY">
    <value>>false</value>
  </resourceProperty>
  <resourceProperty name="PROP_INPUTCONTROL_TYPE">
    <value>7</value>
  </resourceProperty>
```



```
<resourceProperty name="PROP_QUERY_VALUE_COLUMN">
  <value>name</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY_VISIBLE_COLUMNS">
  <resourceProperty name="PROP_QUERY_VISIBLE_COLUMN_NAME">
    <value>name</value>
  </resourceProperty>
  <resourceProperty name="PROP_QUERY_VISIBLE_COLUMN_NAME">
    <value>phone_office</value>
  </resourceProperty>
  <resourceProperty name="PROP_QUERY_VISIBLE_COLUMN_NAME">
    <value>billing_address_city</value>
  </resourceProperty>
</resourceProperty>
<resourceProperty name="PROP_QUERY_DATA">
  <resourceProperty name="PROP_QUERY_DATA_ROW">
    <value>A & L Powers Engineering, Inc</value>
    <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
      <value>A & L Powers Engineering, Inc</value>
    </resourceProperty>
    <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
      <value>738-555-3283</value>
    </resourceProperty>
    <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
      <value>Haney</value>
    </resourceProperty>
  </resourceProperty>
  <resourceProperty name="PROP_QUERY_DATA_ROW">
    <value>A & U Jaramillo Telecommunications, Inc</value>
    <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
      <value>A & U Jaramillo Telecommunications, Inc</value>
    </resourceProperty>
    <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
      <value>564-555-6913</value>
    </resourceProperty>
    <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
      <value>Walla Walla</value>
    </resourceProperty>
  </resourceProperty>
  <resourceProperty name="PROP_QUERY_DATA_ROW">
    <value>A & U Stalker Telecommunications, Inc</value>
    <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
      <value>A & U Stalker Telecommunications, Inc</value>
    </resourceProperty>
    <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
      <value>323-555-1226</value>
    </resourceProperty>
    <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
      <value>Mill Valley</value>
    </resourceProperty>
  </resourceProperty>
</resourceProperty>
```

```
<resourceProperty name="PROP_QUERY_DATA_ROW">
  <value>A & X Caravello Engineering, Inc</value>
  <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
    <value>A & X Caravello Engineering, Inc</value>
  </resourceProperty>
  <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
    <value>958-555-5890</value>
  </resourceProperty>
  <resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
    <value>Tlaxiaco</value>
  </resourceProperty>
</resourceProperty>
</resourceProperty>
<resourceDescriptor name="query" wsType="query" uriString="/MyInputControls/
TEST_LIST_files/query" isNew="false">
  <label>query</label>
<description>query</description>
<resourceProperty name="PROP_RESOURCE_TYPE">
  <value>com.jaspersoft.jasperserver.api.metadata.common.domain.Query</value>
</resourceProperty>
<resourceProperty name="PROP_PARENT_FOLDER">
  <value>/MyInputControls/TEST_LIST_files</value>
</resourceProperty>
<resourceProperty name="PROP_VERSION">
  <value>1</value>
</resourceProperty>
<resourceProperty name="PROP_HAS_DATA">
  <value>>false</value>
</resourceProperty>
<resourceProperty name="PROP_IS_REFERENCE">
  <value>>false</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY">
  <value>SELECT name, phone_office, billing_address_city FROM accounts order by
  name</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY_LANGUAGE">
  <value>sql</value>
</resourceProperty>
  <resourceDescriptor name="" wsType="datasource" uriString="" isNew="false">
    <label>null</label>
    <resourceProperty name="PROP_REFERENCE_URI">
      <value>/datasources/JServerJdbcDS</value>
    </resourceProperty>
    <resourceProperty name="PROP_IS_REFERENCE">
      <value>>true</value>
    </resourceProperty>
  </resourceDescriptor>
</resourceDescriptor>
</resourceDescriptor>
```

The query result is a set of rows that represents the full set of possible values for the input control. For each row, the repository web service returns the value that `runReport` expects for that particular option in the input control. Each row also includes the column values that should be displayed in the input control when prompting users.

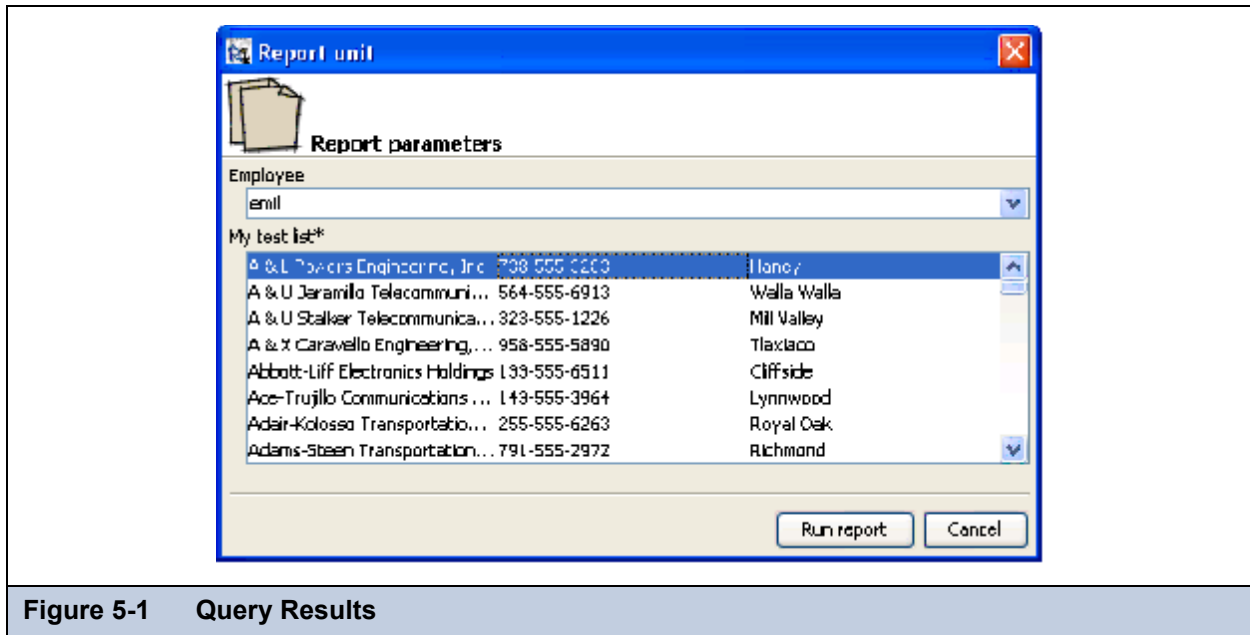


Figure 5-1 Query Results

This figure shows input controls based on queries, as they are rendered by the iReport Designer plugin for JasperReports Server. When the web services run report units, the rendering of input controls is left to the client application. The best way to proceed is:

- Get the report unit.
- Check for query-based input controls by looking at the `PROP_INPUTCONTROL_TYPE` resource property or each child resource descriptor where `wsType` is equal to `inputControl`.
- Get each query-based input control by setting the `IC_GET_QUERY_DATA` argument to true.
- Render the input controls (if used in the report unit), being mindful of the input control properties (such as read only and mandatory).
- Call the `runReport` service and pass the user-selected values.

The rows are stored in the `PROP_QUERY_DATA` resource property: for each row, a child resource property named `PROP_QUERY_DATA_ROW` contains the value and a set of children that contain the column values; these last resource properties are named `PROP_QUERY_DATA_ROW_COLUMN`.

The following schema may elucidate the whole data structure:

```

PROP_QUERY_DATA
(
  PROP_QUERY_DATA_ROW, value
  (
    PROP_QUERY_DATA_ROW_COLUMN, value
    PROP_QUERY_DATA_ROW_COLUMN, value
    ...
  )
  PROP_QUERY_DATA_ROW, value
  (
    PROP_QUERY_DATA_ROW_COLUMN, value
    PROP_QUERY_DATA_ROW_COLUMN, value
    ...
  )
)

```

```

PROP_QUERY_DATA_ROW, value
(
PROP_QUERY_DATA_ROW_COLUMN, value
PROP_QUERY_DATA_ROW_COLUMN, value
...
)
)

```

In Java, to simplify response processing, the `resourceDescriptor` class provides the `getQueryData()` method that returns a list of `InputControlQueryDataRow`, which is a convenient class containing all the row information (row and column values).

5.4 Put Operation

The `put` operation adds new resources to the repository or modifies existing ones. Whether the service adds or modifies a resource depends on whether the request's `isNew` resource descriptor attribute is set to `true`. The parent URI of the new resource must exist, and can be the repository root (`/`). When modifying a resource, you must provide the whole resource descriptor; the changes do not impact child resources.



You cannot use the `put` web service to create report options.

In the web interface, report options are created when users specify values for a report's input controls or filters, and then choose to save those settings. A new instance of the report appears as a child of the report itself. Users click the report instance to run the report using the saved values.

The following XML code creates a folder called `test` inside the `/reports/samples` folder:

```

<request operationName="put" locale="en">
  <resourceDescriptor name="test" wsType="folder" uriString="/reports/samples/test"
    isNew="true">
    <label>Test</label>
    <description>This is a test</description>
    <resourceProperty name="PROP_PARENT_FOLDER">
      <value>/reports/samples</value>
    </resourceProperty>
  </resourceDescriptor>
</request>

```

When adding a file resource, the data must be added as an attachment to the SOAP request, and the `PROP_HAS_DATA` property must be set to `true`. When modifying a file resource, you only need to attach the file if it must be replaced; otherwise `PROP_HAS_DATA` can be set to `FALSE`. In this case, the properties you provide are changed (for example, the label and the description).

The following Java sample creates a new image resource in the repository using the sample classes provided with JasperReports Server:

```

ResourceDescriptor rdis = new ResourceDescriptor();
rdis.setResourceType(ResourceDescriptor.TYPE_IMAGE);
rdis.setName("testImageName");
rdis.setLabel("TestImageLabel");
rdis.setDescription("Test Image Description");
rdis.setParentFolder("/images");

```

```

rdis.setUriString(rdis.getParentFolder() + "/" + rdis.getName());
rdis.setWsType(ResourceDescriptor.TYPE_IMAGE);

File img = new File("/some/file/logo.jpg");

rdis.setHasData(true);
rdis.setIsNew(true);

ResourceDescriptor result = wsclient.addOrModifyResource(rdis, img);

```

Working with report units is a bit more complicated. When creating a new report unit, the request must contain a child JRXML resource descriptor where the `PROP_RU_IS_MAIN_REPORT` property is set to `true`. This resource becomes the main JRXML of the report unit. If it is defined locally to the report, the file must be attached to the SOAP request (in this case, the parent URI for report unit's children is not relevant, and can be set to something like `<report unit parent uri>/<report unit name>_files`).

If the report unit's main JRXML already resides in the repository, the descriptor is still defined as a JRXML resource (that is, the `wsType` property must be set to `jrxml`), and the `PROP_FILERESOURCE_REFERENCE_URI` property must be set to the URI of the correct JRXML resource in the repository.

A second child resource is recognized during creation: a data source descriptor of the data source that the server will use to run the report. This resource is optional, and can be defined either locally to the report unit or as a reference to another resource in the repository:

- When the data source is defined locally, the resource's `wsType` must be a valid data source type, such as `jdbc`, `jndi`, or `bean`.
- If the data source is defined elsewhere in the repository, its `wsType` must be set to `datasource`, which indicates an undefined resource that can be used as a data source, and its `PROP_FILERESOURCE_IS_REFERENCE` property must be set to `true`. The resource's actual URI must be set using the `PROP_FILERESOURCE_REFERENCE_URI` property.

Other resources such as input controls and subreports, must be added separately using the `put` operation to modify the report unit.

Creating, modifying, and removing resources in a report unit is similar to working with resources in a folder. The main difference is that you must set the request's `MODIFY_REPORTUNIT_URI` argument to the URI of the report unit you want to modify. You cannot remove the JRXML resource flagged as main JRXML, but can replace or modify it. The repository web service doesn't allow you to add more than a single data source to the report unit; the report unit is always run against this data source.



When creating reports with parameters, note that the corresponding input controls must be added using a subsequent web service request; you cannot create the input controls in the same web service request that created the report.

5.5 Delete Operation

This operation deletes resources from the repository. If the specified resource is located in a report unit, you must set the request's `MODIFY_REPORTUNIT_URI` argument to the URI of the report unit you want to modify.

If you are deleting a folder, all its content is removed recursively. There is no way to recover a deleted resource or folder, so use caution when calling this service.

The following sample request deletes a resource from a report unit:

```
<request operationName="delete" locale="en">
<argument name="MODIFY_REPORTUNIT_URI">/reports/JD_New_report</argument>
<resourceDescriptor name="test_img" wsType="img" uriString="/reports/
JD_New_report_files/test_img">
<label>test image</label>
<description>test image</description>
</resourceDescriptor>
</request>
```

5.6 Move Operation

This operation moves a repository folder or resource to a different folder in the repository. The operation exposes the API repository service `moveResource` and `moveFolder` methods.

The operation expects (as part of the request) a resource descriptor that identifies the resource or folder to be moved. The new location of the resource or folder must be provided as the value of the `DESTINATION_URI` request argument. The destination URI must resolve to an existing repository folder.

The following request moves the report unit located at `/Reports/NewReport` to `/MyReports`:

```
<request operationName="move" locale="en"
<argument name="DESTINATION_URI">/MyReports</argument>
<resourceDescriptor name="NewReport" wsType="reportUnit" uriString="/Reports/
NewReport">
</resourceDescriptor>
</request>
```

5.7 Copy Operation

This operation creates a copy of an existing resource or folder. The operation exposes the repository service `copyResource` and `copyFolder` API methods.

The resource or folder to be copied is sent as the resource descriptor of the request; the caller does not need to provide the full resource information; just the information required to locate the resource is required.

The full location of the copy must be provided as the value of the `DESTINATION_URI` request argument. If this location already exists in the repository at the moment the operation is called, the server automatically changes the name part of the destination URI and saves the resource or folder copy at the new URI.

The copy operation response includes a descriptor for the saved resource or folder copy. The response descriptor is particularly useful in determining whether the copy has been created at the specified destination URI or at a different/generated URI.

When a folder is being copied, all its subfolders and contained resources are copied recursively.

The following request copies the report unit located at `/Reports/NewReport` to `/MyReports/NewReportCopy`:

```
<request operationName="copy" locale="en"
<argument name="DESTINATION_URI">/MyReports/NewReportCopy</argument>
<resourceDescriptor name="NewReport" wsType="reportUnit" uriString="/Reports/NewReport">
</resourceDescriptor>
</request>
```

5.8 runReport Operation

This operation executes a report on the server then returns the report's results in the specified format. The client application is responsible for prompting users for values to pass to any input controls referenced by the report, as shown in the following sample request XML:

```
<request operationName="runReport" locale="en">
<argument name="RUN_OUTPUT_FORMAT">JRPRINT</argument>
<resourceDescriptor name="" wsType=""
uriString="/reports/samples/EmployeeAccounts"
isNew="false">
<label>null</label>
<parameter name="EmployeeID">emil_id</parameter>
<parameter name="TEST_LIST" isListItem="true">A & L Powers Engineering, Inc</
parameter>
<parameter name="TEST_LIST" isListItem="true">A & U Jaramillo Telecom, Inc</
parameter>
<parameter name="TEST_LIST" isListItem="true">A & U Stalker Telecom, Inc</parameter>
</resourceDescriptor>
</request>
```

This example shows a parameter tag:

```
<!ELEMENT parameter (#PCDATA)>

<!ATTLIST parameter
name CDATA #REQUIRED
isListItem ( true | false ) false
```

In the example, name is the input control to set. If the input control is of type multi-select, the list of selected values is composed of a set of parameter tags that have the same names and have the isListItem attribute set to true, indicating that the parameter is part of a list.

The next example shows the getInputControlValues call for a cascading multi-select input control:

1. The IC_GET_QUERY_DATA argument gets the data from the data source.
2. The RU_REF_URI argument points to the report in which the input control is used.
3. Parameter tags under resourceDescriptor supply the parameters for the input control. The parameters' specifics are derived from the ReportUnit resource properties ([page 156](#)).

```
ResourceDescriptor rd = new ResourceDescriptor();
rd.setUriString("/reports/samples/Cascading_multi_select_report_files/
Cascading_state_multi_select");
rd.setResourceProperty(rd.PROP_QUERY_DATA, null);
ListItem li1 = new ListItem("Country_multi_select", "USA");
li1.setIsListItem(true);
rd.getParameters().add(li1);
ListItem li2 = new ListItem("Country_multi_select", "Mexico");
li2.setIsListItem(true);
rd.getParameters().add(li2);
java.util.List args = new java.util.ArrayList();
args.add(new Argument(Argument.IC_GET_QUERY_DATA, ""));
args.add(new Argument(Argument.RU_REF_URI,
"/reports/samples/Cascading_multi_select_report"));
ResourceDescriptor rd2 = wsclnt.get(rd, null, args);
```

```

if (rd2.getQueryData() != null) {
    List l = (List) rd2.getQueryData();
    for (Object dr : l) {
        InputControlQueryDataRow icdr = (InputControlQueryDataRow) dr;
        for (Object cv : icdr.getColumnValues()) {
            System.out.print(cv + " | ");
        }
        System.out.println();
    }
}

```



Note the following conventions for parameter values:

- ♦ All parameter values are treated as strings; only number, string, and date/time values are allowed.
- ♦ Numbers cannot include punctuation for the digit grouping symbol (thousands separator) and must use a period (.) as the decimal separator (if the relative parameter is not an integer).
- ♦ Dates and date/times must be represented as the number of milliseconds since January 1, 1970, 00:00:00 GMT.

5.8.1 Report Output

The files produced are attached to the response. Specify the final format of the report by setting the request `RUN_OUTPUT_FORMAT` argument. Its possible values are: PDF, JRPRINT, HTML, XLS, XML, CSV and RTF. The default is PDF.

If the final format is set to JRPRINT, the data attached to the response contains a serialized instance of a JasperPrint object. This is the best format to use for clients in Java environments, because it provides the Java client with access to all of JasperReports' export options, and only relies on the server to fill the report.

The following Java code shows how to access the serialized object and get the JasperPrint object:

```

FileContent content = null;
if (attachments != null && !attachments.isEmpty()) {

    content = (FileContent) (attachments.values().toArray()[0]);
}

if (content == null) {
    throw new Exception("No JasperPrint");
}

InputStream is = new ByteArrayInputStream(content.getData());

JasperPrint print = (JasperPrint) JRLoader.loadObject(is);

```

If the specified output format is HTML, the URI for images can be set using the `RUN_OUTPUT_IMAGES_URI` argument: the default value is `images/`. If images are found, they are attached to the response.

If only a single page should be printed, use the `RUN_OUTPUT_PAGE` argument, which must contain the index of page to fill.

5.8.2 Report Locales

Reports that have resource bundles for localization can be generated in a specific languages when the locale is passed using the `REPORT_LOCALE` built-in report parameter. If this parameter is not specified in the web service request, the report locale defaults to the request's locale. If no locale was specified for the request, the report is generated in the server's default locale.

The following XML shows a request to run a report in the Italian locale, which is passed as the value of the `REPORT_LOCALE` built-in report parameter:


```

<request operationName="runReport" locale="fr">
<argument name="RUN_OUTPUT_FORMAT">JRPRINT</argument>
<resourceDescriptor name="" wsType="" uriString="/reports/samples/EmployeeAccounts"
isNew="false">
<label>null</label>
<parameter name="REPORT_LOCALE">it</parameter>
<parameter name="EmployeeID">emil_id</parameter>
</resourceDescriptor>
</request>

```

If the built-in report parameter is removed from this request, the report is generated in French, based on the locale attribute of the request.

5.9 Errors

When the repository web service returns a code other than 0, an error has occurred during the server execution. The exact error is described in the message field of the operation result.

If the problem is environmental, such as an incorrect service URL, an incorrect user name or password, network errors, or an unavailable service, a web services error is returned.



In deployments with multiple organizations, the organization ID must be included in the user name in the format `username|organization_ID`. When there is only one organization exists in JasperReports Server, such as in the default installation, specify the user name alone.

The following shows an Axis connection refused error:

```

AxisFault
faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.userException
faultSubcode:
faultString: java.net.ConnectException: Connection refused: connect
faultActor:
faultNode:
faultDetail:
{http://xml.apache.org/axis/}stackTrace:java.net.ConnectException: Connection
refused: connect
at java.net.PlainSocketImpl.socketConnect(Native Method)
at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:333)
at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:195)
at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:182)
at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:366)

```

The following shows an Axis user name/password error:

```

AxisFault
faultCode: {http://xml.apache.org/axis/}HTTP
faultSubcode:
faultString: (401)Bad credentials
faultActor:
faultNode:
faultDetail:
{:return code: 401

```

```

<html><head><title>Apache Tomcat/5.5.16 - Error re
port</title><style><!--H1 {font-family:Tahoma,Arial,sans-serif;co
lor:white;background-color:#525D76;font-size:22px;} H2 {font-family:Tahoma,Arial
,sans-serif;color:white;background-color:#525D76;font-size:16px;} H3 {font-famil
y:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;}
BODY {font-family:Tahoma,Arial,sans-serif;color:black;background-color:white;} B
{font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;} P {
font-family:Tahoma,Arial,sans-serif;background:white;color:black;font-size:12px;
}A {color : black;}A.name {color : black;}HR {color : #525D76;}--&gt;&lt;/style&
gt; &lt;/head>&lt;body>&lt;h1>HTTP Status 401 - Bad credentials&lt;/h1&
gt;&lt;hr size="1" noshade="&gt;&lt;p&gt;&lt;b&gt;ty
pe&lt;/b&gt; Status report&lt;/p&gt;&lt;p&gt;&lt;b&gt;message&lt;/b&gt; &lt;u&gt
;Bad credentials&lt;/u&gt;&lt;p&gt;&lt;p&gt;&lt;b&gt;description&lt;/b&gt; &lt;
u&gt;This request requires HTTP authentication (Bad credentials).&lt;/u&gt;&lt;/
p&gt;&lt;hr size="1" noshade="&gt;&lt;h3&gt;Apache T
omcat/5.5.16&lt;/h3&gt;&lt;/body>&lt;/html>
{http://xml.apache.org/axis/}HttpErrorCode:401

(401)Bad credentials
at org.apache.axis.transport.http.HTTPSender.readFromSocket(HTTPSender.java:744)
at org.apache.axis.transport.http.HTTPSender.invoke(HTTPSender.java:144)
at org.apache.axis.strategies.InvocationStrategy.visit(InvocationStrategy.java:32)
at org.apache.axis.SimpleChain.doVisiting(SimpleChain.java:118)
at org.apache.axis.SimpleChain.invoke(SimpleChain.java:83)
at org.apache.axis.client.AxisClient.invoke(AxisClient.java:165)

```

5.10 Implementation Suggestions

The iReport plugin for JasperReports Server relies on the repository web service described in this document. If you use Java, Jaspersoft recommends that you familiarize yourself with the plugin's source code, as it can help you understand how best to implement your own web services client. It is included in JasperReports Server Professional and Enterprise editions. It is also available in the [JasperReports Server source code](#) on SourceForge. In the source code, look for the plug-in files in this location: `jasperserver/jasperserver-ireport-plugin`. Download the ZIP file that contains the iReport plugin. In particular, this JAR can be very illuminating:

```
<js-install>ireport\ireport\modules\com-jaspersoft-ireport-jasperserver.jar
```

Its dependencies, located in `<js-install>ireport\ireport\modules\ext`, are:

- `js_activation-1.1.jar`
- `js_axis-1.4patched.jar`
- `js_commons-codec-1.3.jar`
- `js_commons-discovery-0.2.jar`
- `js_commons-httpclient-3.1.jar`
- `js_jasperserver-common-ws-3.5.0.jar`
- `js_jaxrpc.jar`
- `js_mail-1.4.jar`
- `js_saa-j-api-1.3.jar`
- `js_wsdl4j-1.5.1.jar`

If necessary, you can marshal and unmarshal request and response objects by using the following classes:

- `com.jaspersoft.jasperserver.ws.xml.Marshaller`
- `com.jaspersoft.jasperserver.ws.xml.Unmarshaller`

If you use a development environment other than Java (such as .NET), you can easily generate a client from the WSDL.

CHAPTER 6 SOAP - REPORT SCHEDULING WEB SERVICE

The scheduling web service exposes JasperReports Server's report scheduling functionality to integrating applications by the means of a dedicated web service. The web service is the equivalent of the API report scheduling service (`com.jaspersoft.jasperserver.api.engine.scheduling.service.ReportSchedulingService`) and exposes the same operations as this API service.

The service works via XML-RPC calls that use the SOAP encoding. It uses the HTTP protocol to send and receive requests and responses. By default, it is deployed at `/services/ReportScheduler`. You can retrieve the service WSDL (Web Service Description Language) document by appending `?wsdl` to the service URL. For example:

```
http://localhost:8080/jasperserver-pro/services/ReportScheduler?wsdl
```

This chapter includes the following sections:

- [Types Defined in the WSDL](#)
- [Operations in the Scheduling Service](#)
- [Java Client Classes](#)

6.1 Types Defined in the WSDL

The WSDL defines several types that are used by the parameters and operation result of the service. The types belong to the `http://www.jasperforge.org/jasperserver/ws` namespace. The namespace is only an identifier; it is not a valid URL.

This section provides a partial list of the types used for report scheduling; for the complete reference, refer to the WSDL document. The report scheduling types include:

Type	Type Element	Description
Job		Encapsulates all the attributes of a report job. This type is used when full report job details are passed to or returned by an operation.
	ID and version	Required when updating a report job
	reportUnitURI	URI of the report
	username	Set automatically to the name of the calling user when a report job is created
	label	Job label.

Type	Type Element	Description
	simpleTrigger or calendarTrigger	Job trigger, which can be either a simple (fixed interval) trigger or a calendar trigger
	parameters	List of report parameter/input control values
	baseOutputFilename	Base name for the job output
	outputFormats	List of job output formats (as strings). JasperReports Server has built-in support for the following formats: PDF, HTML, XLS, RTF and CSV.
	outputLocale	String representation of a <code>java.util.Locale</code> to be used as report locale
	repositoryDestination	Location in the repository where the report output is saved
	mailNotification	Information regarding the email notification that is sent when the job executes. Set this value to NULL to suppress notifications. Note: To use this feature, you must configure a mail server, as described in <i>JasperReports Server Administrator Guide</i> .
JobSimpleTrigger		Job trigger that fires at fixed intervals
	startDate and endDate	Start and end dates of the job
	timezone	Time zone of the start and end dates
	occurrenceCount	How many times to run the job. If a single run job is wanted, use 1 as occurrence count; if the job is to be fired indefinitely or until the end date, use -1.
	recurrenceInterval and recurrenceIntervalUnit	Interval at which the job should recur: MINUTE, HOUR, DAY, WEEK.
JobCalendarTrigger		Job trigger that fires at a time specified by a CRON-like expression
	startDate and endDate	Start and end dates of the job
	timezone	Time zone of the start and end dates
	minutes	Minute or minutes of the day when the job is to run.
	hours	Hour or hours of the day when the job is to run.
	daysType	How days are specified. Possible values are ALL, WEEK, and MONTH.
	weekDays	Used when daysType is WEEK; this indicates the days of the week from Saturday (1) to Sunday (7).
	monthDays	Used when daysType is MONTH, this indicates the month days.
	months	Months (from 0 to 11) on which the job should be triggered.
JobRepositoryDestination		Information about where to save the report job output in the repository.
	folderURI	URI of the folder where the report output will be saved.
	sequentialFileNames	Flag indicating whether to append timestamps to the base output name.

Type	Type Element	Description
JobMailNotification		Encapsulates the attributes of the mail notification to send regarding the report job.
	toAddresses	List of email addresses to which the notification will be sent.
	subject	Subject of the email notification.
	resultSendType	Indicates whether to attach the report output to the email; the value is either SEND (only the messages is sent) or SEND_ATTACHMENT (the report output is sent along as a message attachment).
JobSummary		Used when a list of report jobs is retrieved via the service. The full report job information can be retrieved individually for required jobs.
	id	Unique identifier of the job.
	label	Display label of the job.
	state	State of the job. For example, NORMAL indicates that the job is waiting for next execution; EXECUTING means the job is running.
	previousFireTime	Most recent run time.
	nextFireTime	Next run time.

6.2 Operations in the Scheduling Service

6.2.1 Operation Descriptions

The report scheduling web service consists of the following operations:

- `getAllJobs`. Returns the list of all accessible report jobs.
- `getReportJobs`. Returns the list of all accessible report jobs for a specific report (whose URL is sent as a parameter).
- `scheduleJob`. Schedules a new job. The job details must be sent as parameters; the operation returns the saved job details as its result.
- `updateJob`. Updates an existing job. The full job details (as retrieved via `getJob`) must be sent as a parameter; the operation returns the updated job details as saved by the JasperReports Server scheduling service.
- `getJob`. Returns the full job details of a report job whose ID is sent as a parameter.
- `deleteJob` and `deleteJobs`. Delete a single or several report job specified by their IDs. These operations do not return any information.

If an exception occurs while processing an operation request, the exception is converted to a SOAP fault which is sent as its response. In this case, the exception stacktrace is included in the response, and can be used for debugging.

Exceptions thrown by the JasperReports Server code have localizable messages. The operation caller can specify the locale in which the messages of such exceptions are returned by setting a SOAP envelope header. The header should be named `locale` and should use `http://www.jasperforge.org/jasperserver/ws` as namespace; the header value is a string representation of the desired message locale. For more information, refer to [5.1, “Request and Operation Result,” on page 115](#).

6.2.2 Example Request and Operation Result

This is the full SOAP request for a `scheduleJob` operation that creates a job with four report parameters:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
/XMLSchema-instance">
  <soapenv:Body>
    <ns1:scheduleJob soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      " xmlns:ns1="http://www.jasperforge.org/jasperserver/ws">
      <job xsi:type="ns1:Job">
        <reportUnitURI xsi:type="xsd:string">/reports/samples/SalesByMonth
        </reportUnitURI>
        <username xsi:type="xsd:string" xsi:nil="true"/>
        <label xsi:type="xsd:string">Label 3</label>
        <description xsi:type="xsd:string">Description 3</description>
        <simpleTrigger xsi:type="ns1:JobSimpleTrigger">
          <timezone xsi:type="xsd:string" xsi:nil="true"/>
          <startDate xsi:type="xsd:dateTime">2008-10-09T09:25:00.000Z</startDate>
          <endDate xsi:type="xsd:dateTime" xsi:nil="true"/>
          <occurrenceCount xsi:type="xsd:int">1</occurrenceCount>
          <recurrenceInterval xsi:type="xsd:int" xsi:nil="true"/>
          <recurrenceIntervalUnit xsi:type="ns1:IntervalUnit" xsi:nil="true"/>
        </simpleTrigger>
        <calendarTrigger xsi:type="ns1:JobCalendarTrigger" xsi:nil="true"/>
        <parameters soapenc:arrayType="ns1:JobParameter[4]" xsi:type="soapenc:Array"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
          <parameters xsi:type="ns1:JobParameter">
            <name xsi:type="xsd:string">TextInput</name>
            <value xsi:type="soapenc:int">22</value>
          </parameters>
          <parameters xsi:type="ns1:JobParameter">
            <name xsi:type="xsd:string">CheckboxInput</name>
            <value xsi:type="soapenc:boolean">>true</value>
          </parameters>
          <parameters xsi:type="ns1:JobParameter">
            <name xsi:type="xsd:string">ListInput</name>
            <value xsi:type="soapenc:string">2</value>
          </parameters>
          <parameters xsi:type="ns1:JobParameter">
            <name xsi:type="xsd:string">DateInput</name>
            <value xsi:type="xsd:dateTime">2007-10-09T09:00:00.000Z</value>
          </parameters>
        </parameters>
        <baseOutputFilename xsi:type="xsd:string">Sales3</baseOutputFilename>
        <outputFormats soapenc:arrayType="xsd:string[1]" xsi:type="
          "soapenc:Array" xmlns:soapenc="http://schemas.xmlsoap.org/soap/
          encoding/">
          <outputFormats xsi:type="xsd:string">PDF</outputFormats>
        </outputFormats>
        <outputLocale xsi:type="xsd:string" xsi:nil="true"/>
      </job>
    </ns1:scheduleJob>
  </soapenv:Body>
</soapenv:Envelope>
```

```

<repositoryDestination xsi:type="ns1:JobRepositoryDestination">
  <folderURI xsi:type="xsd:string">/ContentFiles</folderURI>
  <sequentialFileNames xsi:type="xsd:boolean">>false
  </sequentialFileNames>
  <overwriteFiles xsi:type="xsd:boolean">>false</overwriteFiles>
</repositoryDestination>
<mailNotification xsi:type="ns1:JobMailNotification" xsi:nil="true"/>
</job>
</ns1:scheduleJob>
</soapenv:Body>
</soapenv:Envelope>

```

The response of the request contains the job details as saved by the server:

```

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance">
  <soapenv:Body>
    <ns1:scheduleJobResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/
      encoding/" xmlns:ns1="http://www.jasperforge.org/jasperserver/ws">
      <scheduleJobReturn xsi:type="ns1:job">
        <id xsi:type="xsd:long">7</id>
        <version xsi:type="xsd:int">0</version>
        <reportUnitURI xsi:type="xsd:string">/reports/samples/SalesByMonth</
          reportUnitURI>
        <username xsi:type="xsd:string">tomcat</username>
        <label xsi:type="xsd:string">Label 3</label>
        <description xsi:type="xsd:string">Description 3</description>
        <simpleTrigger xsi:type="ns1:jobSimpleTrigger">
          <id xsi:type="xsd:long">7</id>
          <version xsi:type="xsd:int">0</version>
        </simpleTrigger>
        <timezone xsi:type="xsd:string">Europe/Minsk</timezone>
        <startDate xsi:type="xsd:dateTime">2008-10-09T09:25:00.000Z</startDate>
        <endDate xsi:type="xsd:dateTime" xsi:nil="true"/>
        <occurrenceCount xsi:type="xsd:int">1</occurrenceCount>
        <recurrenceInterval xsi:type="xsd:int" xsi:nil="true"/>
        <recurrenceIntervalUnit xsi:type="ns1:IntervalUnit" xsi:nil="true"/>
      </scheduleJobReturn>
    </ns1:scheduleJobResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

```

<parameters xsi:type="ns1:jobParameter">
  <name xsi:type="xsd:string">DateInput</name>
  <value xsi:type="xsd:dateTime">2007-10-09T09:00:00.000Z</value>
</parameters>
<parameters xsi:type="ns1:jobParameter">
  <name xsi:type="xsd:string">ListInput</name>
  <value xsi:type="soapenc:string">2</value>
</parameters>
</parameters>
<baseOutputFilename xsi:type="xsd:string">Sales3</baseOutputFilename>
<outputFormats soapenc:arrayType="xsd:string[1]" xsi:type="soapenc:Array"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
  <outputFormats xsi:type="xsd:string">PDF</outputFormats>
</outputFormats>
<outputLocale xsi:type="xsd:string" xsi:nil="true"/>
<repositoryDestination xsi:type="ns1:jobRepositoryDestination">
  <id xsi:type="xsd:long">7</id>
  <version xsi:type="xsd:int">0</version>
  <folderURI xsi:type="xsd:string">/ContentFiles</folderURI>
  <sequentialFileNames xsi:type="xsd:boolean">>false</sequentialFileNames>
  <overwriteFiles xsi:type="xsd:boolean">>false</overwriteFiles>
</repositoryDestination>
<mailNotification xsi:type="ns1:JobMailNotification" xsi:nil="true"/>
</scheduleJobReturn>
</ns1:scheduleJobResponse>
</soapenv:Body>
</soapenv:Envelope>

```

6.3 Java Client Classes

The JasperReports Server web service client jars contain classes that can be used by Java clients to easily communicate with the report scheduling web service.

XML types used by the report scheduling web service are mapped to Java bean classes found in the `weekday` package (for example, `Job`, `JobSimpleTrigger`, and `CalendarDaysType`). Instances of these classes can be used as report job objects that are sent to or returned by the web service.

The service itself is represented by Apache Axis-generated client stub classes. A façade (`com.jaspersoft.jasperserver.ws.scheduling.ReportSchedulerFacade`) has been developed on top of these classes. The façade can be instantiated by providing the information required to locate and connect to a web service (the endpoint URL and the username/password for authentication).

Jaspersoft recommends using the façade because it handles items such as the Axis client configuration and the messages locale header.

CHAPTER 7 SOAP - DOMAIN WEB SERVICE



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 Domain web service exposes a limited subset of JasperReports Server's Domain functionality to integrating applications by the means of a dedicated web service. Applications may only read the contents of a Domain, not create or edit Domains.

The service works via XML-RPC calls that use the SOAP encoding. It uses the HTTP protocol to send and receive requests and responses. By default, it is deployed at `/services/DomainServices`. You can retrieve the WSDL document by appending `?wsdl` to the service URL. For example:

```
http://localhost:8080/jasperserver-pro/services/DomainServices?wsdl
```

This chapter contains the following sections:

- **Types Defined in the WSDL**
- **Operations in the Domain Service**

7.1 Types Defined in the WSDL

The WSDL defines several types that are returned by operations of the service. The types belong to the `http://www.jasperforge.org/jasperserver/ws` namespace. The namespace is only an identifier; it is not a valid URL. For the complete reference, refer to the WSDL document.

These are the objects returned when accessing Domains:

- `SimpleMetaData`. Encapsulates all the sets and items in a Domain structure.
- `SimpleMetaLevel`. Represents an item set in the Domain. It may contain items, other item sets, or both.
- `SimpleMetaItem`. An item in the Domain. Unlike a level or set, an item is a source of data referenceable in a query.
- `ResultSetData`. Object returned by a Domain query. It contains column names and rows of data.
- `DataRow`. Contains values for each column in a row.

7.2 Operations in the Domain Service

The Domain web service provides the following operations:

- ♦ `getDomainMetaData`. Returns the tree structure of sets and items in a Domain. The object returned can be used to render the Domain for users and allow them to select items for a query.
- ♦ `executeDomainQuery`. Returns a set of values in response to a query.

If an exception occurs while processing an operation request, the exception is converted to a SOAP fault that is sent as its response. In this case, the exception stacktrace is included in the response, which can be useful for debugging.

Exceptions thrown by JasperReports Server have localizable messages. The operation caller can specify the locale in which the messages of such exceptions are returned by setting a SOAP envelope header. The header should be named `locale` and should use `http://www.jasperforge.org/jasperserver/ws` as its namespace; the header value is a string representation of the desired message locale. For more information, see [5.1, “Request and Operation Result,” on page 115](#).

7.2.1 The `getDomainMetaData` Operation

The `getDomainMetaData` operation takes these parameters:

- ♦ `domainUri` - a string containing the path to the Domain on the server, for example `/domains/John/ExpenseDomain`.
- ♦ `localeStr` - a string giving the user locale, for example `en`, `en_US`, or `es_ES_Traditional_WIN`.

The operation returns the tree structure of item sets and items in the requested Domain. The tree structure consists of levels that represent the nested sets and items that represent the items in the Domain. Levels may contain sub-levels, items, or both, thus modelling the hierarchical structure of the Domain.

The following object types are combined to create the tree structure in the return value:

- ♦ `SimpleMetaData`. Encapsulates all the item sets and items in a Domain structure:
 - ♦ `rootLevel`. The `SimpleMetaLevel` object that is the root of the Domain tree structure.
 - ♦ `properties`. There are currently no properties on this object.
- ♦ `SimpleMetaLevel`. Represents an item set in the Domain. It has the following attributes:
 - ♦ `id` and `label`. Unique identifier and label string for this item set.
 - ♦ `items`. An array of `SimpleMetaItem` objects representing the items in this set.
 - ♦ `subLevels`. An array of `SimpleMetaLevel` objects representing the sub-sets of this set.
 - ♦ `properties`. The `resourceId` key indicates the resource identifier of this item set.
- ♦ `SimpleMetaItem`. An item in the Domain. It has the following attributes:
 - ♦ `id` and `label`. Unique identifier and label string for this item.
 - ♦ `javaType`. The Java class name of this item, for example `java.lang.String`.
 - ♦ `properties`. The `javaType` key is identical to the `javaType` attribute, and the `resourceId` key indicates the resource identifier of this item.



A resource identifier is an internal property that identifies the data resource that the set or item references. Web applications do not need to process or return this value.

This is the full SOAP request for a `getDomainMetaData` operation:

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance">
  <soapenv:Body>
    <ns1:getDomainMetaData soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/
      encoding/" xmlns:ns1="http://www.jasperforge.org/jasperserver/ws">
      <domainUri xsi:type="xsd:string">/Domains/examples/SampleDomain</domainUri>
      <localeStr xsi:type="xsd:string">US</localeStr>
    </ns1:getDomainMetaData>
  </soapenv:Body>
</soapenv:Envelope>
```

The response of the request contains the tree structure of the Domain:

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:getDomainMetaDataResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns1="http://www.jasperforge.org/jasperserver/ws">
      <getDomainMetaDataReturn xsi:type="ns1:SimpleMetaData">
        <rootLevel xsi:type="ns1:SimpleMetaLevel">
          <id xsi:type="xsd:string">root</id>
          <label xsi:type="xsd:string" xsi:nil="true"/>
          <properties soapenc:arrayType="ns1:SimpleProperty[0]" xsi:type="soapenc:Array"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" />
          <items soapenc:arrayType="ns1:SimpleMetaItem[0]" xsi:type="soapenc:Array"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" />
          <subLevels soapenc:arrayType="ns1:SimpleMetaLevel[7]" xsi:type="soapenc:Array"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" />
          <subLevels xsi:type="ns1:SimpleMetaLevel">
            <id xsi:type="xsd:string">expense_join</id>
            <label xsi:type="xsd:string">expense</label>
            <properties soapenc:arrayType="ns1:SimpleProperty[1]" xsi:type="
              soapenc:Array">
              <properties xsi:type="ns1:SimpleProperty">
                <key xsi:type="xsd:string">resourceId</key>
                <value xsi:type="xsd:string">expense_join</value>
              </properties>
            </properties>
          </subLevels>
          <items soapenc:arrayType="ns1:SimpleMetaItem[2]" xsi:type="soapenc:Array">
            <items xsi:type="ns1:SimpleMetaItem">
              <id xsi:type="xsd:string">ej_expense_fact_exp_date</id>
              <label xsi:type="xsd:string">Exp Date</label>
              <javaType xsi:type="xsd:string">java.sql.Date</javaType>
              <properties soapenc:arrayType="ns1:SimpleProperty[2]" xsi:type="
                soapenc:Array">
                <properties xsi:type="ns1:SimpleProperty">
                  <key xsi:type="xsd:string">JavaType</key>
                  <value xsi:type="xsd:string">java.sql.Date</value>
                </properties>
                <properties xsi:type="ns1:SimpleProperty">
                  <key xsi:type="xsd:string">resourceId</key>
                  <value xsi:type="xsd:string">expense_join.e.exp_date</value>
                </properties>
              </properties>
            </items>
            <items xsi:type="ns1:SimpleMetaItem">
              <id xsi:type="xsd:string">ej_expense_fact_amount</id>
              <label xsi:type="xsd:string">Amount</label>
              <javaType xsi:type="xsd:string">java.math.BigDecimal</javaType>
              <properties soapenc:arrayType="ns1:SimpleProperty[2]" xsi:type="
                soapenc:Array">
                <properties xsi:type="ns1:SimpleProperty">
                  <key xsi:type="xsd:string">JavaType</key>
                  <value xsi:type="xsd:string">java.math.BigDecimal</value>
```

```

        </properties>
        <properties xsi:type="ns1:SimpleProperty">
            <key xsi:type="xsd:string">resourceId</key>
            <value xsi:type="xsd:string">expense_join.e.amount</value>
        </properties>
    </properties>
</items>
</items>
<subLevels soapenc:arrayType="ns1:SimpleMetaLevel[0]" xsi:type="
    soapenc:Array"/>
</subLevels>
<subLevels xsi:type="ns1:SimpleMetaLevel">
    <id xsi:type="xsd:string">expense_join_store</id>
    <label xsi:type="xsd:string">store</label>
    <properties soapenc:arrayType="ns1:SimpleProperty[1]" xsi:type="
        soapenc:Array">
        <properties xsi:type="ns1:SimpleProperty">
            <key xsi:type="xsd:string">resourceId</key>
            <value xsi:type="xsd:string">expense_join</value>
        </properties>
    </properties>
    <items soapenc:arrayType="ns1:SimpleMetaItem[24]" xsi:type="
        soapenc:Array">
        <items xsi:type="ns1:SimpleMetaItem">
            <id xsi:type="xsd:string">ej_store_store_type</id>
            <label xsi:type="xsd:string">Store Type</label>
            <javaType xsi:type="xsd:string">java.lang.String</javaType>
            <properties soapenc:arrayType="ns1:SimpleProperty[2]" xsi:type="
                soapenc:Array">
                <properties xsi:type="ns1:SimpleProperty">
                    <key xsi:type="xsd:string">JavaType</key>
                    <value xsi:type="xsd:string">java.lang.String</value>
                </properties>
                <properties xsi:type="ns1:SimpleProperty">
                    <key xsi:type="xsd:string">resourceId</key>
                    <value xsi:type="xsd:string">expense_join.s.store_type</value>
                </properties>
            </properties>
        </items>
        ...
    </items>
    <subLevels soapenc:arrayType="ns1:SimpleMetaLevel[0]" xsi:type="
        soapenc:Array"/>
</subLevels>
</subLevels>
</rootLevel>
<properties soapenc:arrayType="ns1:SimpleProperty[0]" xsi:type="soapenc:Array"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" />
</getDomainMetaDataReturn>
</ns1:getDomainMetaDataResponse>
</soapenv:Body>
</soapenv:Envelope>

```

7.2.2 The executeDomainQuery Operation

The `executeDomainQuery` operation takes these parameters:

- `domainUri` - a string containing the path to the Domain on the server, for example `/domains/John/ExpenseDomain`.
- `queryStr` - a string containing the Domain query composed of fields and a filter expression (see below for the syntax).
- `localeStr` - a string giving the user locale, for example `en`, `en_US`, or `es_ES_Traditional_WIN`.
- `dateFormatStr` - a string giving the date format desired in date fields, for example `MM/dd/yyyy` or `h:mm a`.



Be sure the format has date and time portions if you expect to have both date and time fields, for example `yyyy.MM.dd G 'at' HH:mm:ss z`

The query string is composed of the following elements that create a syntax for the Domain query:

- `<query>` - encapsulates the whole query.
- `<queryFields>` - contains a sequence of `<queryField>` elements. The order of fields will be preserved in the results.
- `<queryField id="<fullyQualifiedID>" />` - an empty element where `<fullyQualifiedID>` gives the unique identifier of an item you want to appear as a column in the results. The identifier must be fully qualified, which means it includes the identifiers of the set and super-sets to which the item belongs. The fully qualified identifier is similar to the path of the item in the Domain, using a period (.) to separate each set identifier.
- `<queryFilterString>` - the filter string for the query uses an application-specific syntax called Domain Expression Language (DomEL).

The following example shows a filter string that must match two values:

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

Note that when the query string appears in the SOAP example below, special characters such as `<` and `>` are converted to their corresponding character entities, `<` and `>` respectively.

The `executeDomainQuery` operation returns results in the following objects:

- `ResultSetData`. Encapsulates the results of the Domain query. It contains column names and rows of data:
 - `names`. Array of column names in the result set. These names match the order and items in the query fields.
 - `data`. An array of data rows.
- `DataRow`. Represents a record and contains values for each column in a row:
 - `data`. An array of strings, one for the value in each column, in the same order as the names array.



Note that all values are given in string format.

The following example shows the full SOAP request for an executeDomainQuery operation on a sample Domain:

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
  <soapenv:Body>
    <ns1:executeDomainQuery soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/" xmlns:ns1="http://www.jasperforge.org/jasperserver/ws">
      <domainUri xsi:type="xsd:string">/Domains/examples/SampleDomain</domainUri>
      <queryStr xsi:type="xsd:string">&lt;query&gt;  &lt;queryFields&gt;
&lt;queryField id=&quot;expense_join_account.ej_account_account_description&quot;
/&gt;
&lt;queryField id=&quot;expense_join_account.ej_expense_fact_account_id&quot; /&gt;
&lt;queryField id=&quot;expense_join_account.ej_account_account_parent&quot; /&gt;
&lt;queryField id=&quot;expense_join_account.ej_account_account_rollup&quot; /&gt;
&lt;queryField id=&quot;expense_join_account.ej_account_account_type&quot; /&gt;
&lt;queryField id=&quot;expense_join_account.ej_account_Custom_Members&quot; /&gt;
&lt;queryField id=&quot;expense_join.ej_expense_fact_amount&quot; /&gt;
&lt;queryField id=&quot;expense_join.ej_expense_fact_exp_date&quot; /&gt;
&lt;queryField id=&quot;expense_join_store.ej_store_store_type&quot; /&gt;
&lt;queryField id=&quot;expense_join_store.ej_store_store_street_address&quot; /&gt;
&lt;queryField id=&quot;expense_join_store.ej_store_store_city&quot; /&gt;
&lt;queryField id=&quot;expense_join_store.ej_store_store_state&quot; /&gt;
&lt;queryField id=&quot;expense_join_store.ej_store_store_postal_code&quot; /&gt;
&lt;queryField id=&quot;expense_join_store.ej_store_store_country&quot; /&gt;
&lt;queryFields&gt;
&lt;queryFilterString&gt;expense_join_account.ej_account_account_description ==
'Marketing'&lt;/queryFilterString&gt;&lt;/query&gt;</queryStr>
      <localeStr xsi:type="xsd:string">US</localeStr>
      <dateFormatStr xsi:type="xsd:string">MM/dd/yyyy</dateFormatStr>
    </ns1:executeDomainQuery>
  </soapenv:Body>
</soapenv:Envelope>
```

The response to the request contains the current values in the specified Domain:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
  <soapenv:Body>
    <ns1:executeDomainQueryResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/" xmlns:ns1="http://www.jasperforge.org/jasperserver/ws">
      <executeDomainQueryReturn xsi:type="ns1:ResultSetData">
        <names soapenc:arrayType="xsd:string[31]" xsi:type="soapenc:Array"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
          <names xsi:type="xsd:string">expense_join_account.ej_account_account_
description/>
          <names xsi:type="xsd:string">expense_join_account.ej_expense_fact_
account_id/>
          <names xsi:type="xsd:string">expense_join_account.ej_account_account_parent/>
          <names xsi:type="xsd:string">expense_join_account.ej_account_account_rollup/>
          <names xsi:type="xsd:string">expense_join_account.ej_account_account_type/>
          <names xsi:type="xsd:string">expense_join_account.ej_account_Custom_Members/>
          <names xsi:type="xsd:string">expense_join.ej_expense_fact_amount/>
```

```

<names xsi:type="xsd:string">expense_join_store.ej_store_store_type/>
<names xsi:type="xsd:string">expense_join_store.ej_store_store_street_
  address/>
<names xsi:type="xsd:string">expense_join_store.ej_store_store_city/>
<names xsi:type="xsd:string">expense_join_store.ej_store_store_state/>
<names xsi:type="xsd:string">expense_join_store.ej_store_store_postal_code/>
</names>
<data soapenc:arrayType="ns1:DataRow[600]" xsi:type="soapenc:Array"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
  <data xsi:type="ns1:DataRow">
    <data soapenc:arrayType="xsd:string[31]" xsi:type="soapenc:Array">
      <data xsi:type="xsd:string">Marketing</data>
      <data xsi:type="xsd:string">4300</data>
      <data xsi:type="xsd:string">4000</data>
      <data xsi:type="xsd:string">+</data>
      <data xsi:type="xsd:string">Expense</data>
      <data xsi:type="xsd:string" xsi:nil="true"/>
      <data xsi:type="xsd:string">1884.0000</data>
      <data xsi:type="xsd:string">01/01/1997</data>
      <data xsi:type="xsd:string">HeadQuarters</data>
      <data xsi:type="xsd:string">1 Alameda Way</data>
      <data xsi:type="xsd:string">Alameda</data>
      <data xsi:type="xsd:string">CA</data>
      <data xsi:type="xsd:string">94502</data>
      <data xsi:type="xsd:string">USA</data>
    </data>
  </data>
  ...
</data>
</executeDomainQueryReturn>
</ns1:executeDomainQueryResponse>
</soapenv:Body>
</soapenv:Envelope>

```

7.2.3 Java Client Classes

The JasperReports Server web service includes classes in JAR files that can be used by Java clients to easily communicate with the Domain web service.

Java bean classes for the XML types declared in WSDL (for example, `SimpleMetaLevel`, `SimpleMetaItem`, `ResultSetData` and `DataRow`) are located in `ji-common-ws-server-5.2.0.jar` in the `com.jaspersoft.ji.ws.axis2.domain.generate` package. Instances of these classes can be used to communicate with the Domain web service.

CHAPTER 8 SOAP - WEB SERVICES FOR ADMINISTRATION

Web services for administration expose a limited set of JasperReports Server's system administration functionality. There are three services:

- Users and Roles
- Organizations/Tenants
- Permissions

The services work via XML-RPC calls that use the SOAP encoding. They use the HTTP protocol to send and receive requests and responses. By default, they are deployed at `/services/UserAndRoleManagementService`, `/services/OrganizationManagementService`, and `/services/PermissionsManagementService`. You can retrieve the WSDL documents by appending `?wsdl` to the service URL. For example:

```
http://localhost:8080/jasperserver-pro/services/UserAndRoleManagementService?wsdl
```

If an exception occurs while processing an operation request, the exception is converted to a SOAP fault that is sent as its response. In this case, the exception stacktrace is included in the response, which can be useful for debugging.

Exceptions thrown by JasperReports Server have localizable messages. The operation caller can specify the locale in which the messages of such exceptions are returned by setting a SOAP envelope header. The header should be named `locale` and should use `http://www.jasperforge.org/jasperserver/ws` as its namespace; the header value is a string representation of the desired message locale. For more information, refer to section 5.1, “Request and Operation Result,” on page 115.

All authentication and authorization rules established in the system apply to operations run through the web services. Refer to *Jasper Administrator Guide* for more information about the rules.

This chapter includes the following sections:

- **Types Defined in the WSDL**
- **Users and Roles**
- **Organizations/Tenants**
- **Permissions**
- **Related Files**

8.1 Types Defined in the WSDL

The WSDL (Web Services Description Language) document defines the types that are returned by operations of the services. The types belong to the `http://www.jasperforge.org/jasperserver/ws` namespace. The namespace is only an identifier; it is not a valid URL. For the complete reference, refer to the WSDL document in `jasperserver-ws-server-4.0.jar`.

The following tables summarize the services' operations.

Users and Roles Service				
Operation	Parameter	Parameter Type	Return Type	Description
findUsers	criteria	WSUserSearchCriteria	WSUser []	Returns a list of one or more users. criteria has username mask, organization/tenant ID, includeSubOrgs, list of required roles, and maxRecords; null in parameters means "any." Note: includeSubOrgs and tenantID are reserved for use in our commercial products. If they are used in community project products, they must be NULL.
putUser	user	WSUser	WSUser	Adds or updates a user. Returns the new or updated WSUser.
deleteUser	user	WSUser	No return	Deletes the named user.
findRoles	criteria	WSRoleSearchCriteria	WSRole []	Returns WSRole [], a list of roles. criteria has rolename mask, organization/tenant ID, includeSubOrgs, maxRecords; null in parameters means "any." Note: includeSubOrgs and tenantID are reserved for use in our commercial products. If they are used in community project products, they must be NULL.
putRole	role	WSRole	WSRole	Adds or updates a role. Returns new or updated WSRole.
updateRoleName	oldRole	WSRole	WSRole	Returns WSRole.
	newName	String	WSRole	New name of role.
deleteRole	role	WSRole	No return	Deletes the named role.

Organizations/Tenants Service				
Operation	Parameter	Parameter Type	Return Type	Description
getTenant	tenantId	String	WSTenant []	Organization/tenant identifier. Returns an organization/tenant.
getSubTenantList	tenantId	String	WSTenant []	Organization/tenant identifier. Returns WSTenant [], a list of suborganizations in the specified organization.
putTenant	tenant	WSTenant	WSTenant	Adds or updates a tenant. Returns the new or updated WSTenant.
deleteTenant	tenantId	String	No return	Deletes the named organization/tenant.

Permissions Service				
Operation	Parameter	Parameter Type	Return Type	Description
getPermissionsForObject	targetURI	String	WSObjectPermission[]	Repository object URI. Returns WSObjectPermission[], a list of permissions for the specified object.
putPermissions	objPerm	WSObjectPermission	WSObjectPermission	Object permission. Returns WSObjectPermission, a new or updated object permission.
deletePermissions	objPerm	WSObjectPermission	No return	Deletes the named permission.

8.2 Users and Roles

The web service for administration of users and roles has these operations:

- findUsers and findRoles. Return a list of users or roles that meet specified criteria.
- putUser and putRole. Return the named user or role. If the object is not already in the database, the call creates a new one.
- deleteUser and deleteRole. Delete the named user or role.

8.2.1 findUsers

In findUsers, the parameter criteria has the type WSUserSearchCriteria and returns type WSUser. criteria can be a username mask, an organization/tenant ID, includeSubOrgs, a list of required users, and maxRecords. Null values indicate "any."

- The mask has an SQL-like notation. For instance, U2_.
- When includeSubOrgs is TRUE, all objects of the specified type are within a search's scope and result. Otherwise, only objects in the requested organization (or root if tenantId=null) are searched.
- To limit the number of objects to return, set maxRecords to the desired number. To allow an infinite number of objects, set maxRecords to 0.

To call findUsers:

```
WSUserSearchCriteria searchCriteria = new WSUserSearchCriteria();
searchCriteria.setName("demo");
searchCriteria.setTenantId("organization_1"); // Name of organization or null
searchCriteria.setMaxRecords(5);
searchCriteria.setIncludeSubOrgs(false);
searchCriteria.setRequiredRoles(requiredRoles);

WSRole role = new WSRole();
role.setRoleName("ROLE_USER");
role.setTenantId(null);
searchCriteria.setRequiredRoles(new WSRole[] {role});

WSUser[] list = binding.findUsers(searchCriteria);
```

The return is:

```
String getUsername()
String getFullName()
String getPassword()
String getEmailAddress()
Boolean getExternallyDefined()
Boolean getEnabled()
Date getPreviousPasswordChangeTime()
String getTenantId()
WSRole[] getRoles()
String getRoleName()
String getTenantId()
WSUser[] getUsers()
```

8.2.2 putUser

In `putUser`, the parameter `user` has the type `WSUser` and returns type `WSUser`.

`putUser` updates an existing object; if the specified user does not exist, a new one is created.



Before adding users and roles, note that there is a server-side configuration which specifies the default roles that a new user can receive. See the *JasperReports Server Administrator Guide* for details

To call `putUser`:

```
WSUser user = new WSUser();
user.setUsername("john");
user.setTenantId("organization_1");
user.setEnabled(true);
user.setFullName("John Doe");
WSRole role = new WSRole();
role.setRoleName("ROLE_ANONYMOUS");
role.setTenantId(null);
user.setRoles(new WSRole[] {role});

WSUser value = binding.putUser(user);
```

The return is:

```
String getUsername()
String getFullName()
String getPassword()
String getEmailAddress()
Boolean getExternallyDefined()
Boolean getEnabled()
Date getPreviousPasswordChangeTime()
String getTenantId()
WSRole[] getRoles()
String getRoleName()
String getTenantId()
WSUser[] getUsers()
```

8.2.3 deleteUser

In `deleteUser`, the parameter `user` has the type `WSUser`.

To call `deleteUser`:

```
WSUser user = new WSUser();
user.setUsername("john");
user.setTenantId("organization_1");

binding.deleteUser(user);
```

There is no return.

8.2.4 findRoles

In `findRoles`, the parameter `criteria` has the type `WSRoleSearchCriteria` and returns type `WSRole`.

`criteria` can be a rolename mask, an organization/tenant ID, `includeSubOrgs`, a list of required users or roles, and `maxRecords`. Null values indicate "any."

- The mask has a SQL-like notation. For instance, `Ad%, U2_`.
- When `includeSubOrgs` is `TRUE`, all objects of the specified type are within a search's scope and result. Otherwise, only objects in the requested organization (or root if `tenantId=null`) are searched.
- To limit the number of objects to return, set `maxRecords` to the desired number. To allow an infinite number of objects, set `maxRecords` to 0.

To call `findRoles`:

```
WSRoleSearchCriteria searchCriteria = new WSRoleSearchCriteria();
searchCriteria.setRoleName("ROLE_USER");
searchCriteria.setTenantId("organization_1");
searchCriteria.setMaxRecords(5);
searchCriteria.setIncludeSubOrgs(false);

WSRole[] list = binding.findRoles(searchCriteria);
```

The return is:

```
String getUsername()
String getFullName()
String getPassword()
String getEmailAddress()
Boolean getExternallyDefined()
Boolean getEnabled()
Date getPreviousPasswordChangeTime()
String getTenantId()
WSRole[] getRoles()
String getRoleName()
String getTenantId()
WSUser[] getUsers()
```

8.2.5 putRole

In `putRole`, the parameter `role` has the type `WSRole` and returns type `WSRole`. `putRole` updates an existing object; if the specified role does not exist, a new one is created.



Before adding users and roles, note that there is a server-side configuration which specifies the default roles that a new user can receive. See *JasperReports Server Administrator Guide* for details.

To call `putRole`:

```
WSRole role = new WSRole();
role.setRoleName("ROLE_ANONYMOUS");
role.setTenantId(null);
WSRole value = binding.putRole(role);
```

The return is:

```
String getRoleName()
String getTenantId()
WSUser[] getUsers()
String getUsername()
String getFullName()
String getPassword()
String getEmailAddress()
Boolean getExternallyDefined()
Boolean getEnabled()
Date getPreviousPasswordChangeTime()
String getTenantId()
WSRole[] getRoles()
```

8.2.6 updateRoleName

In `updateRoleName`, the parameter `oldRole` has the type `WSRole`, and the parameter `newName` has the type `String`. They both return type `WSRole`.

To update a role with a call to `oldRole`:

```
WSRole oldRole= new WSRole();
role.setRoleName("ROLE_WS");
role.setTenantId("organization_1");
WSRole value = binding.updateRoleName(oldRole, "ROLE_WEB_SERVICE");
```

To rename the role with a call to `newName`: `"ROLE_WEB_SERVICE"`. The return for an updated role:

```
String getRoleName()
String getTenantId()
WSUser[] getUsers()
String getUsername()
String getFullName()
String getPassword()
String getEmailAddress()
Boolean getExternallyDefined()
Boolean getEnabled()
Date getPreviousPasswordChangeTime()
String getTenantId()
WSRole[] getRoles()
```

8.2.7 deleteRole

In `deleteUser`, the parameter `user` has the type `WSUser`. In `deleteRole`, the parameter `role` has the type `WSRole`.

Here are examples of calls to `deleteUser` and `deleteRole`:

```
WSRole role = new WSRole();
role.setRoleName("ROLE_WS");
role.setTenantId("organization_1");
binding.deleteRole(role);
```

There is no return.

8.3 Organizations/Tenants



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.

At present, there is no practical difference between organizations and tenants; both kinds of entity are administered with these tenant operations:

- `getTenant`. Returns a list of tenants that meet specified criteria.
- `getSubTenantList`. Returns a list of sub-tenants (units within a tenant).
- `putTenant`. Returns the named tenant. If the object is not already in the database, the call creates a new one.
- `deleteTenant`. Deletes the named tenant.

8.3.1 getTenant

In `getTenant`, the parameter `tenantId` has the type `String` and returns type `WSTenant`.

To call `getTenant`:

```
String tenantId = "organization_1";
```

The return is:

```
String getTenantId()
String getTenantName()
String getTenantAlias()
String getTenantDesc()
String getTenantNote()
String getTenantUri()
String getTenantFolderUri()
String getParentId()
```

8.3.2 getSubTenantList

In `getSubTenantList`, the parameter `tenantId` has the type `String` and returns type `WSTenant []`.

To call `getSubTenantList`:

```
String tenantId = "organization_1";
```

The return is:

```
String getTenantId()
String getTenantName()
String getTenantAlias()
String getTenantDesc()
String getTenantNote()
String getTenantUri()
String getTenantFolderUri()
String getParentId()
```

8.3.3 putTenant

In `putTenant`, the parameter `tenant` has the type `WSTenant` and returns type `WSTenant`.

To call `putTenant`. Note that `tenantUri` and `tenantFolderUri` are calculated automatically from the tenant's `tenantId` and `parentId`. As a result, the `tenantUri` and `tenantFolderUri` fields of the `WSTenant` object are ignored:

```
WSTenant wsTenant = new WSTenant();
wsTenant.setTenantId("suborg1");
wsTenant.setParentId("organization_1");
wsTenant.setTenantAlias("organization_1");
wsTenant.setTenantName("Sub organization1");
wsTenant.setTenantDesc("Sub organization1 description");
wsTenant.setTenantNote("Sub organization notes");
```

The return is:

```
String getTenantId()
String getTenantName()
String getTenantAlias()

String getTenantDesc()
String getTenantNote()
String getTenantUri()
String getTenantFolderUri()
String getParentId()
```

8.3.4 deleteTenant

In `deleteTenant`, the parameter `tenantId` has the type `String`.

To call `deleteTenant`.

```
String tenantId = "organization_1";
```

There is no return.

8.4 Permissions

The web service for administration of permissions has these operations:

- `getPermissionsForObject`. Returns a list of permissions for the specified object.
- `putPermission`. Returns the named permission. If the object is not already in the database, the call creates a new one.

- ♦ `deletePermission`. Deletes the named permission.

8.4.1 getPermissionsForObject

In `getPermissionsForObject`, the parameter `targetURI` has the type `String` and returns type `WSObjectPermission[]`.

To call `getPermissionsForObject`:

```
WSObjectPermission[] objectPermissions = binding.getPermissionsForObject("repo:/");
```

In the return, the permissioned object can be a user or role:

```
String getUri()
Object getPermissionRecipient()
int getPermissionMask()
String getRoleName()
String getTenantId()
WSUser[] getUsers()
String getUsername()
String getFullName()
String getPassword()
String getEmailAddress()
Boolean getExternallyDefined()
Boolean getEnabled()
Date getPreviousPasswordChangeTime()
String getTenantId()
WSRole[] getRoles()
```

8.4.2 putPermission

In `putPermission`, the parameter `objPerm` has the type `WSObjectPermission` and returns type `WSObjectPermission`.

To call `putPermission`:

```
WSObjectPermission objectPermission = new WSObjectPermission();
objectPermission.setUri(resourceUri);
objectPermission.setPermissionMask(2);

WSUser wsUser = new WSUser();
wsUser.setUsername("joeuser");
wsUser.setTenantId("organization_1");

objectPermission.setPermissionRecipient(wsUser);

WSObjectPermission value = binding.putPermission(objectPermission);
```

The `setPermissionMask()` function accepts the following values. It is not a true mask because bit-wise combinations of these values are not supported by the server. These values should be treated as constants:

- ♦ No access: 0
- ♦ Administer: 1
- ♦ Read-only: 2
- ♦ Read-delete: 18
- ♦ Read-write-delete: 30
- ♦ Execute-only: 32

The return is:

```
String getUri()
Object getPermissionRecipient()
int getPermissionMask()
String getRoleName()
String getTenantId()
WSUser[] getUsers()
String getUsername()
String getFullName()
String getPassword()
String getEmailAddress()
Boolean getExternallyDefined()
Boolean getEnabled()
Date getPreviousPasswordChangeTime()
String getTenantId()
WSRole[] getRoles()
```

8.4.3 deletePermission

In `deletePermission`, the parameter `objPerm` has the type `WSObjectPermission`. To call `deletePermission`:

```
WSObjectPermission objectPermission = new WSObjectPermission();
objectPermission.setUri(resourceUri);
objectPermission.setPermissionMask(2);

WSUser wsUser = new WSUser();
wsUser.setUsername("joeuser");
wsUser.setTenantId("organization_1");

objectPermission.setPermissionRecipient(wsUser);

binding.deletePermission(objectPermission);
```

There is no return.

8.5 Related Files

The web services distribution files include WSDL files as well as client stub classes. The WSDL file for the administration service is in `jasperserver-ws-server-4.0.jar`. Client stub files contain return types; they are in `jasperserver-common-ws-4.0.jar`. The JasperReports Server Professional and Enterprise implementation of the services is in `ji-ws-server-4.0.jar`.

APPENDIX A RESOURCEDESCRIPTOR API CONSTANTS

The constants that the services require are defined in the following classes:

- `com.jaspersoft.jasperserver.api.metadata.xml.domain.impl.ResourceDescriptor`
- `com.jaspersoft.jasperserver.api.metadata.xml.domain.impl.Argument`

The following values are extracted from `ResourceDescriptor`:

```
// Resource wsTypes
TYPE_FOLDER = "folder";
TYPE_REPORTUNIT = "reportUnit";
TYPE_DATASOURCE = "datasource";
TYPE_DATASOURCE_JDBC = "jdbc";
TYPE_DATASOURCE_JNDI = "jndi";
TYPE_DATASOURCE_BEAN = "bean";
TYPE_DATASOURCE_VIRTUAL = "virtual";
TYPE_DATASOURCE_CUSTOM = "custom";
TYPE_DATASOURCE_AWS = "aws"; // Amazon Web Services
TYPE_IMAGE = "img";
TYPE_FONT = "font";
TYPE_JRXML = "jrxml";
TYPE_CLASS_JAR = "jar";
TYPE_RESOURCE_BUNDLE = "prop";
TYPE_REFERENCE = "reference";
TYPE_INPUT_CONTROL = "inputControl";
TYPE_DATA_TYPE = "dataType";
TYPE_OLAP_MONDRIAN_CONNECTION = "olapMondrianCon";
TYPE_OLAP_XMLA_CONNECTION = "olapXmlaCon";
TYPE_MONDRIAN_SCHEMA = "olapMondrianSchema";
TYPE_ACCESS_GRANT_SCHEMA = "accessGrantSchema"; // Pro-only
TYPE_UNKNOW = "unknow";
TYPE_LOV = "lov"; // List of values...
TYPE_QUERY = "query";
TYPE_CONTENT_RESOURCE = "contentResource";
TYPE_STYLE_TEMPLATE = "jrtx";
TYPE_XML_FILE = "xml";
```

```
// These constants are copied here from DataType for facility
DT_TYPE_TEXT = 1;
DT_TYPE_NUMBER = 2;
DT_TYPE_DATE = 3;
DT_TYPE_DATE_TIME = 4;

// These constants are copied here from InputControl for facility
IC_TYPE_BOOLEAN = 1;
IC_TYPE_SINGLE_VALUE = 2;
IC_TYPE_SINGLE_SELECT_LIST_OF_VALUES = 3;
IC_TYPE_SINGLE_SELECT_QUERY = 4;
IC_TYPE_MULTI_VALUE = 5;          // This type is deprecated
IC_TYPE_MULTI_SELECT_LIST_OF_VALUES = 6;
IC_TYPE_MULTI_SELECT_QUERY = 7;
IC_TYPE_SINGLE_SELECT_LIST_OF_VALUES_RADIO = 8;
IC_TYPE_SINGLE_SELECT_QUERY_RADIO = 9;
IC_TYPE_MULTI_SELECT_LIST_OF_VALUES_CHECKBOX = 10;
IC_TYPE_MULTI_SELECT_QUERY_CHECKBOX = 11;

...

// ReportUnit resource properties
...
RU_CONTROLS_LAYOUT_POPUP_SCREEN = 1;
RU_CONTROLS_LAYOUT_SEPARATE_PAGE = 2;
RU_CONTROLS_LAYOUT_TOP_OF_PAGE = 3;
RU_CONTROLS_LAYOUT_IN_PAGE = 4;
...

// Content resource properties
...
CONTENT_TYPE_PDF = "pdf";
CONTENT_TYPE_HTML = "html";
CONTENT_TYPE_XLS = "xls";
CONTENT_TYPE_RTF = "rtf";
CONTENT_TYPE_CSV = "csv";
CONTENT_TYPE_IMAGE = "img";
```

The constants in the Argument class are:

```
// Arguments

MODIFY_REPORTUNIT = "MODIFY_REPORTUNIT_URI";
CREATE_REPORTUNIT = "CREATE_REPORTUNIT_BOOLEAN";
LIST_DATASOURCES = "LIST_DATASOURCES";
IC_GET_QUERY_DATA = "IC_GET_QUERY_DATA";

VALUE_TRUE = "true";
VALUE_FALSE = "false";

RUN_OUTPUT_FORMAT = "RUN_OUTPUT_FORMAT";
RUN_OUTPUT_FORMAT_PDF = "PDF";
RUN_OUTPUT_FORMAT_JRPRINT = "JRPRINT";
RUN_OUTPUT_FORMAT_HTML = "HTML";
RUN_OUTPUT_FORMAT_XLS = "XLS";
RUN_OUTPUT_FORMAT_XML = "XML";
RUN_OUTPUT_FORMAT_CSV = "CSV";
RUN_OUTPUT_FORMAT_RTF = "RTF";
RUN_OUTPUT_IMAGES_URI = "IMAGES_URI";

RUN_OUTPUT_PAGE = "PAGE";

RUN_TRANSFORMER_KEY = "TRANSFORMER_KEY";
RU_REF_URI = "RU_REF_URI";
PARAMS_ARG = "PARAMS_ARG";

LIST_RESOURCES = "LIST_RESOURCES";
RESOURCE_TYPE = "RESOURCE_TYPE";
REPORT_TYPE = "REPORT_TYPE";
START_FROM_DIRECTORY = "START_FROM_DIRECTORY";

NO_RESOURCE_DATA_ATTACHMENT = "NO_ATTACHMENT";
NO_SUBRESOURCE_DATA_ATTACHMENTS = "NO_SUBRESOURCE_ATTACHMENTS";

DESTINATION_URI = "DESTINATION_URI";
```

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 created through the Ad Hoc Editor. Such reports could be added to dashboards and be scheduled, but when edited in iReport, lost their grouping and sorting. In the current version, the Ad Hoc Editor is used to explore views which in turn can be saved as reports. Such reports can be edited in iReport and 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 in order to edit it in the interactive viewer, schedule it, or add it to a dashboard.

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 a Domain, a field whose value is calculated from a user-written formula that may include any number of fields, operators, and constants. A calculated field is defined in the Domain Designer, and it becomes one of the items to which the Domain's security file and locale bundles can apply.

CRM

Customer Relationship Management. The practice of managing every facet of a company's interactions with its clientele. CRM applications help businesses track and support their customers.

CrossJoin

An MDX function that combines two or more dimensions into a single axis (column or row).

Cube

The basis of most OLAP applications, a cube is a data structure that contains three or more dimensions that categorize the cube's quantitative data. When you navigate the data displayed in an OLAP view, you are exploring a cube.

Custom Field

In the Ad Hoc Editor, a field that is created through menu items as a simple function of one or two available fields, including other custom fields. When a custom field becomes too complex or needs to be used in many reports, it is best to define it as a calculated field in a Domain.

Dashboard

A collection of reports, input controls, graphics, labels, and web content displayed in a single, integrated view. Dashboards often present a high level view of your data, but input controls can parameterize the data to display. For example, you can narrow down the data to a specific date range. Embedded web content, such as other web-based applications or maps, make dashboards more interactive and functional.

Derived Table

In a Domain, a derived table is defined by an additional query whose result becomes another set of items available in the Domain. For example, with a JDBC data source, you can write an SQL query that includes complex functions for selecting data. You can use the items in a derived table for other operations on the Domain, such as joining tables, defining a calculated field, or filtering. The items in a derived table can also be referenced in the Domain's security file and locale bundles.

Data Policy

In JasperReports Server, a setting that determines how the server processes and caches data used by Ad Hoc reports. Select your data policies by clicking **Manage > Ad Hoc Settings**.

Data Source

Defines the connection properties that JasperReports Server needs to access data. The server transmits queries to data sources and obtains datasets in return for use in filling reports and previewing Ad Hoc reports. JasperReports Server supports JDBC, JNDI, and Bean data sources; custom data sources can be defined as well.

Dataset

A collection of data arranged in columns and rows. Datasets are equivalent to relational results sets and the `JRDataSource` type in the JasperReports Library.

Datatype

In JasperReports Server, a datatype is used to characterize a value entered through an input control. A datatype must be of type text, number, date, or date-time. It can include constraints on the value of the input, for example maximum and minimum values. As such, a datatype in JasperReports Server is more structured than a datatype in most programming languages.

Denormalize

A process for creating table joins that speeds up data retrieval at the cost of having duplicate row values between some columns.

Dice

An OLAP operation to select columns.

Dimension

A categorization of the data in a cube. For example, a cube that stores data about sales figures might include dimensions such as time, product, region, and customer's industry.

Domain

A virtual view of a data source that presents the data in business terms, allows for localization, and provides data-level security. A Domain is not a view of the database in relational terms, but it implements the same functionality within JasperReports Server. The design of a Domain specifies tables in the database, join clauses, calculated fields, display names, and default properties, all of which define items and sets of items for creating Ad Hoc reports.

Domain Topic

A Topic that is created from a Domain by the Data Chooser. A Domain Topic is based on the data source and items in a Domain, but it allows further filtering, user input, and selection of items. Unlike a JRXML-based Topic, a Domain Topic can be edited in JasperReports Server by users with the appropriate permissions.

Drill

To click on an element of an OLAP view to change the data that is displayed:

- Drill down. An OLAP operation that exposes more detailed information down the hierarchy levels by delving deeper into the hierarchy and updating the contents of the navigation table.
- Drill through. An OLAP operation that displays detailed transactional data for a given aggregate measure. Click a fact to open a new table beneath the main navigation table; the new table displays the low-level data that constitutes the data that was clicked.
- Drill up. An OLAP operation for returning the parent hierarchy level to view to summary information.

Eclipse

An open source Integrated Development Environment (IDE) for Java and other programming languages, such as C/C++.

ETL

Extract, Transform, Load. A process that retrieves data from transactional systems, and filters and aggregates the data to create a multidimensional database. Generally, ETL prepares the database that your reports will access. The JasperSoft ETL product lets you define and schedule ETL processes.

Fact

The specific value or aggregate value of a measure for a particular member of a dimension. Facts are typically numeric.

Field

A field is equivalent to a column in the relational database model. Fields originate in the structure of the data source, but you may define calculated fields in a Domain or custom fields in the Ad Hoc Editor. Any type of field, along with its display name and default formatting properties, is called an item and may be used in the Ad Hoc Editor.

Frame

A dashboard element that displays reports or custom URLs. Frames can be mapped to input controls if their content can accept parameters.

Group

In a report, a group is a set of data rows that have an identical value in a designated field.

- In a table, the value appears in a header and footer around the rows of the group, while the other fields appear as columns.
- In a chart, the field chosen to define the group becomes the independent variable on the X axis, while the other fields of each group are used to compute the dependent value on the Y axis.

Hierarchy Level

In an OLAP cube, a member of a dimension containing a group of members.

Input Control

A button, check box, drop-down list, text field, or calendar icon that allows users to enter a value when running a report or viewing a dashboard that accepts input parameters. For JRXML reports, input controls and their associated datatypes must be defined as repository objects and explicitly associated with the report. For Domain-based reports that prompt for filter values, the input controls are defined internally. When either type of report is used in a dashboard, its input controls are available to be added as special content.

iReport Designer

An open source tool for graphically designing reports that leverage all features of the JasperReports Library. The Jaspersoft iReport Designer lets you drag and drop fields, charts, and sub-reports 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 iReport, or upload it to JasperReports Server. iReport is implemented in NetBeans.

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 Library

An embeddable, open source, Java API for generating a report, filling it with current data, drawing charts and tables, and exporting to any standard format (HTML, PDF, Excel, CSV, and others). JasperReports processes reports defined in JRXML, an open XML format that allows the report to contain expressions and logic to control report output based on run-time data.

JasperReports Server

A commercial open source, server-based application that calls the JasperReports library to generate and share reports securely. JasperReports Server authenticates users and lets them upload, run, view, schedule, and send reports from a web browser. Commercial versions provide metadata layers, interactive report and dashboard creation, and enterprise features such as organizations and auditing.

Jaspersoft ETL

A graphical tool for designing and implementing your data extraction, transforming, and loading (ETL) tasks. It provides hundreds of data source connectors to extract data from many relational and non-relational systems. Then, it schedules and performs data aggregation and integration into data marts or data warehouses that you use for reporting.

Jaspersoft OLAP

A relational OLAP server integrated into JasperReports Server that performs data analysis with MDX queries. The product includes query builders and visualization clients that help users explore and make sense of multidimensional data. Jaspersoft OLAP also supports XML/A connections to remote servers.

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 given field of each table. Only the fields in a same join tree or calculated from the fields in a same join tree may appear together in a report.

JPivot

An open source graphical user interface for OLAP operations. For more information, visit <http://jpivot.sourceforge.net/>.

JRXML

An XML file format for saving and sharing reports created for the JasperReports Library and the applications that use it, such as iReport Designer and JasperReports Server. JRXML is an open format that uses the XML standard to define precisely all the structure and configuration of a report.

MDX

Multidimensional Expression Language. A language for querying multidimensional objects, such as OLAP (On Line Analytical Processing) cubes, and returning cube data for analytical processing. An MDX query is the query that determines the data displayed in an OLAP view.

Measure

Depending on the context:

- In a report, a formula that calculates the values displayed in a table's columns, a crosstab's data values, or a chart's dependent variable (such as the slices in a pie).
- In an OLAP view, a formula that calculates the facts that constitute the quantitative data in a cube.

Mondrian

A Java-based, open source multidimensional database application.

Mondrian Connection

An OLAP client connection that consists of an OLAP schema and a data source. 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 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 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 helpdesk tickets. Such outliers may indicate a problem (or an important achievement) in your business. The analysis features of Jaspersoft OLAP excel at revealing outliers.

Parameter

Named values that are passed to the engine at report-filling time to control the data returned or the appearance and formatting of the report. A report parameter is defined by its name and type. In JasperReports Server, parameters can be mapped to input controls that users can interact with.

Pivot

To rotate a crosstab such that its row groups become column groups and its column groups become rows. In the Ad Hoc Editor, pivot a crosstab by clicking .

Pivot Table

A table with two physical dimensions (for example, X and Y axis) for organizing information containing more than two logical dimensions (for example, PRODUCT, CUSTOMER, TIME, and LOCATION), such that each physical dimension is capable of representing one or more logical dimensions, where the values described by the dimensions are aggregated using a function such as SUM. Pivot tables are used in Jaspersoft OLAP.

Properties

Settings associated with an object. The settings determine certain features of the object, such as its color and label. Properties are normally editable. In Java, properties can be set in files listing objects and their settings.

Report

In casual usage, *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 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

The tree structure of folders that contain all saved reports, dashboards, OLAP views, and resources. Users access the repository through the JasperReports Server web interface or through iReport. Applications can access the repository through the web service API. Administrators use the import and export utilities to back up the repository contents.

Resource

In JasperReports Server, anything residing in the repository, such as an image, file, font, data source, Topic, Domain, report element, saved report, report output, dashboard, or OLAP view. Resources also include the folders in the repository. Administrators set user and role-based access permissions on repository resources to establish a security policy.

Role

A security feature of JasperReports Server. Administrators create named roles, assign them to user accounts, and then set access permissions to repository objects based on those roles. Certain roles also determine what functionality and menu options are displayed to users in the JasperReports Server interface.

Schema

A logical model that determines how data is stored. For example, the schema in a relational database is a description of the relationships between tables, views, and indexes. In Jaspersoft OLAP, an OLAP schema is the logical model of the data that appears in an OLAP view; they are uploaded to the repository as resources. For Domains, schemas are represented in XML design files.

Schema Workbench

A graphical tool for easily designing OLAP schemas, data security schemas, and MDX queries. The resulting cube and query definitions can then be used in Jaspersoft OLAP to perform simple but powerful analysis of large quantities of multi-dimensional data stored in standard RDBMS systems.

Set

In Domains and Domain Topics, a named collection of items grouped together for ease of use in the Ad Hoc Editor. A set can be based on the fields in a table or entirely defined by the Domain creator, but all items in a set must originate in the same join tree. The order of items in a set is preserved.

Slice

An OLAP operation for filtering data rows.

SQL

Structured Query Language. A standard language used to access and manipulate data and schemas in a relational database.

System Admin

Also called the system administrator. A user who has unlimited access to manage all organizations, users, roles, repository permissions, and repository objects across the entire JasperReports Server instance. The system admin can create root-level organizations and manage all server settings. The default system admin is the `superuser` account.

Topic

A JRXML file created externally and uploaded to JasperReports Server as a basis for Ad Hoc reports. Topics are created by business analysts to specify a data source and a list of fields with which business users can create reports in the Ad Hoc Editor. Topics are stored in the Ad Hoc Components folder of the repository and displayed when a user launches the Ad Hoc Editor.

Transactional Data

Data that describe measurable aspects of an event, such as a retail transaction, relevant to your business. Transactional data are often stored in relational databases, with one row for each event and a table column or field for each measure.

User

Depending on the context:

- A person who interacts with JasperReports Server through the web interface. There are generally three categories of users: administrators who install and configure JasperReports Server, database experts or business analysts who create data sources and Domains, and business users who create and view reports and dashboards.
- A user account that has an ID and password to enforce authentication. Both people and API calls accessing the server must provide the ID and password of a valid user account. Roles are assigned to user accounts to determine access to objects in the repository.

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 set of REST (Representational State Transfer) and SOAP (Simple Object Access Protocol) APIs that enable applications to access certain features of JasperReports Server. The features include repository, scheduling, Domain services, and user administration tasks.

XML

eXtensible Markup language. A standard for defining, transferring, and interpreting data for use across any number of XML-enabled applications.

XML/A

XML for Analysis. An XML standard that uses Simple Object Access protocol (SOAP) to access remote data sources. For more information, see <http://www.xmla.org/>

XML/A Connection

A type of OLAP client connection that consists of Simple Object Access Protocol (SOAP) definitions used to access data on a remote server. OLAP client connections populate OLAP views.