



TIBCO JASPERREPORTS® SERVER

REST API REFERENCE

RELEASE 6.4

<http://www.jaspersoft.com>

Copyright ©2005-2017 TIBCO Software Inc. All Rights Reserved. TIBCO Software Inc.

This is version 0217-JSP64-28 of the *TIBCO JasperReports Server REST API Reference*.

TABLE OF CONTENTS

Chapter 1 REST API Overview	7
1.1 List of Services	7
1.2 Sending REST Requests from a Browser	9
1.3 HTTP Response Codes	9
1.4 Deprecated Web Services	11
Chapter 2 The serverInfo Service	13
Chapter 3 Authentication Methods	15
3.1 Overview of REST Authentication	15
3.2 HTTP Basic Authentication	16
3.3 The login Service	16
3.4 Login Encryption	18
3.5 The Security Check	19
3.6 Using an SSO Token	20
3.7 Using a Pre-Authentication Token	21
3.8 Logout	21
Chapter 4 Working With Resources	23
4.1 Resource URI	23
4.2 Custom Media Types	24
4.3 Accept HTTP Headers	24
4.4 Content-Type HTTP Headers	25
4.5 JSON Format	25
4.6 Nested Resources	26
4.7 Referenced Resources	26
4.8 Local Resources	28
4.9 Optimistic Locking	29
4.10 Update-only Passwords	29
Chapter 5 Resource Descriptors	31
5.1 Common Attributes	32
5.2 Folder	32
5.3 JNDI Data Source	33
5.4 JDBC Data Source	33

5.5 AWS Data Source	33
5.6 Virtual Data Source	34
5.7 Custom Data Source	34
5.8 Bean Data Source	35
5.9 Datatypes	35
5.10 List of Values	36
5.11 Query	36
5.12 Input Control	36
5.13 File	38
5.14 Report Unit (JRXML Report)	38
5.15 Report Options	40
5.16 Domain (semanticLayerDataSource)	40
5.17 Domain Topic	41
5.18 XML/A Connection	42
5.19 Mondrian Connection	42
5.20 Secure Mondrian Connection	42
5.21 OLAP Unit	43
5.22 Mondrian XML/A Definition	44
5.23 Other Types	44
Chapter 6 The resources Service	45
6.1 Searching the Repository	45
6.2 Paginating Search Results	47
6.2.1 Default Pagination	48
6.2.2 Full Page Pagination	49
6.2.3 No Pagination	50
6.3 Viewing Resource Details	51
6.4 Creating a Resource	52
6.5 Modifying a Resource	54
6.6 Copying a Resource	55
6.7 Moving a Resource	56
6.8 Deleting Resources	56
Chapter 7 Working With File Resources	59
7.1 MIME Types	59
7.2 Downloading File Resources	60
7.3 Uploading File Resources	61
7.4 Updating File Resources	62
Chapter 8 Working With Domains	65
8.1 The metadata Service	65
8.2 Fetching a Domain Schema	69
8.3 Fetching Domain Bundles and Security Files	70
Chapter 9 The permissions Service	71
9.1 Permission Constants	71
9.2 Viewing Multiple Permissions	72
9.3 Viewing a Single Permission	73

9.4 Setting Multiple Permissions	74
9.5 Setting a Single Permission	75
9.6 Deleting Multiple Permissions	76
9.7 Deleting a Single Permission	76
Chapter 10 The export Service	79
10.1 Requesting an Export	79
10.2 Polling the Export Status	82
10.3 Fetching the Export Output	82
10.4 Canceling an Export Operation	83
Chapter 11 The import Service	85
Chapter 12 The reports Service	87
12.1 Running a Report	87
12.2 Finding Running Reports	88
12.3 Stopping a Running Report	89
Chapter 13 The reportExecutions Service	91
13.1 Running a Report Asynchronously	91
13.2 Polling Report Execution	95
13.3 Requesting Report Execution Details	95
13.4 Requesting Report Output	97
13.5 Exporting a Report Asynchronously	98
13.6 Modifying Report Parameters	99
13.7 Polling Export Execution	100
13.8 Finding Running Reports and Jobs	101
13.9 Stopping Running Reports and Jobs	102
Chapter 14 The inputControls Service	105
14.1 Listing Input Control Structure	105
14.2 Listing Input Control Values	107
14.3 Setting Input Control Values	108
Chapter 15 The options Service	111
15.1 Listing Report Options	111
15.2 Creating Report Options	112
15.3 Updating Report Options	113
15.4 Deleting Report Options	113
Chapter 16 The jobs Service	115
16.1 Listing Report Jobs	115
16.2 Extended Job Search	117
16.3 Viewing a Job Definition	118
16.4 Scheduling a Report	121
16.5 Viewing Job Status	123
16.6 Editing a Job Definition	123
16.7 Updating Jobs in Bulk	123
16.8 Pausing Jobs	125

16.9 Resuming Jobs	126
16.10 Restarting Failed Jobs	126
16.11 Deleting Jobs	127
16.12 Specifying FTP Output	128
16.13 Specifying File System Output	130
Chapter 17 The calendars Service	133
17.1 Listing All Registered Calendar Names	133
17.2 Viewing an Exclusion Calendar	134
17.3 Deleting an Exclusion Calendar	134
17.4 Adding or Updating an Exclusion Calendar	135
Chapter 18 The queryExecutor Service	139
Chapter 19 The caches Service	143
Chapter 20 The organizations Service	145
20.1 Searching for Organizations	145
20.2 Viewing an Organization	147
20.3 Creating an Organization	148
20.4 Modifying Organization Properties	149
20.5 Setting the Theme of an Organization	149
20.6 Deleting an Organization	150
Chapter 21 The users Service	151
21.1 Searching for Users	151
21.2 Viewing a User	153
21.3 Creating a User	155
21.4 Modifying User Properties	156
21.5 Deleting a User	156
Chapter 22 The roles Service	159
22.1 Searching for Roles	159
22.2 Viewing a Role	161
22.3 Creating a Role	162
22.4 Modifying a Role	163
22.5 Setting Role Membership	163
22.6 Deleting a Role	163
Chapter 23 The attributes Service	165
23.1 Attribute Descriptors	165
23.2 Secure Attributes	166
23.3 Entities with Attributes	167
23.4 Permissions for Accessing Attributes	167
23.5 Referencing Attributes	167
23.6 Attribute Limitations	168
23.7 Viewing Attributes	169
23.8 Setting Attributes	170
23.9 Deleting Attributes	172

CHAPTER 1 REST API OVERVIEW

The JasperReports Server REST API is an Application Programming Interface that follows the guidelines of REpresentational State Transfer design to allow client application to interact with the server through the HTTP protocol. With a few exceptions, the REST API allows clients to interact with all features of the server, such as running, exporting, and scheduling reports, reading and writing resources in the repository, and managing organizations, roles, and users. The REST API requires credentials for every operation and enforces the same permissions and administrator restrictions as the server's user interface.

Client applications send requests to named URLs that are called services. A service provides several operations on a feature, for example the roles service lists the roles in an organization, gives the properties and members of a role, writes new roles, updates existing roles, and deletes roles. This chapter lists all the services of the current REST API. The other chapters of this API Reference each describe one of the services.

In order to describe resources and objects in the server, the REST API sends and receives data structures called descriptors. Most services support descriptors in both XML (eXtensible Markup Language) and JSON (JavaScript Object Notation). The descriptors are specific to each service, and are defined in the corresponding chapter of this reference. Descriptors are usually sent and received in the body of HTTP requests and responses, so your client application usually relies on further APIs to handle the HTTP communications.

Historically, the REST API is considered a web service, and JasperReports Server provided several other web services. The current REST API is the second version and all services use the `rest_v2/` prefix. The first REST API with the `rest/` prefix and the earlier SOAP API (Simple Object Access Protocol) are deprecated and no longer maintained. Although the server might still respond to deprecated services, they are not updated for new features of the server and are never guaranteed to succeed or be accurate. For completeness, the deprecated service names are listed at the end of this chapter.

This chapter includes the following sections:

- **List of Services**
- **Sending REST Requests from a Browser**
- **HTTP Response Codes**
- **Deprecated Web Services**

1.1 List of Services

The REST API of JasperReports Server responds to HTTP requests from client applications, in particular the following methods (sometimes called verbs):

- GET to list, search and acquire information about server resources.

- POST to create new resources and execute reports.
- PUT to modify existing resources.
- DELETE to remove resources.

As with any RESTful service, not all methods (GET, PUT, POST, and DELETE) are supported on every service. The URLs usually include a path to the resource being acted upon, as well as any parameters that are accepted by the method. For example, to search for input control resources in the repository, your application would send the following HTTP request:

```
GET http://<host>:<port>/jasperserver[-pro]/rest_v2/resources?type=inputControl
```

In all URLs in this API Reference:

- <host> is the name of the computer hosting JasperReports Server
- <port> is the port you specified during installation
- jasperserver[-pro] indicates that the service is available in both Community and Commercial editions.
- jasperserver-pro indicates that the service is available only in Commercial editions.
- The context name (by default jasperserver or jasperserver-pro) may be customized in your specific installation of JasperReports Server

The REST services are available at the following URLs:

Table 1-1 REST API Services and URLs

Web Service	URLs
Login (optional)	<p>The following services use the rest/ prefix but are still supported as part of the REST API:</p> <p>http://<host>:<port>/jasperserver[-pro]/rest/GetEncryptionKey</p> <p>http://<host>:<port>/jasperserver[-pro]/rest/login</p> <p>http://<host>:<port>/jasperserver[-pro]/j_spring_security_check</p> <p>http://<host>:<port>/jasperserver[-pro]/logout.html</p> <p>http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo</p>
Repository	<p>http://<host>:<port>/jasperserver[-pro]/rest_v2/resources</p> <p>http://<host>:<port>/jasperserver-pro/rest_v2/domains/.../metadata *</p> <p>http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions</p> <p>http://<host>:<port>/jasperserver[-pro]/rest_v2/export</p> <p>http://<host>:<port>/jasperserver[-pro]/rest_v2/import</p>
Reports	<p>http://<host>:<port>/jasperserver[-pro]/rest_v2/reports</p> <p>http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions</p> <p>http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/.../inputControls</p> <p>http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/.../options</p> <p>http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs</p> <p>http://<host>:<port>/jasperserver-pro/rest_v2/queryExecutor *</p> <p>http://<host>:<port>/jasperserver-pro/rest_v2/caches/vds *</p>

Web Service	URLs
Administration without organizations	<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_v2/users/.../attributes">http://<host>:<port>/jasperserver[-pro]/rest_v2/users/.../attributes <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/roles">http://<host>:<port>/jasperserver[-pro]/rest_v2/roles <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/attributes">http://<host>:<port>/jasperserver[-pro]/rest_v2/attributes
Administration with organizations *	<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_v2/organizations/.../attributes">http://<host>:<port>/jasperserver-pro/rest_v2/organizations/.../attributes <a href="http://<host>:<port>/jasperserver-pro/rest_v2/organizations/.../users">http://<host>:<port>/jasperserver-pro/rest_v2/organizations/.../users <a href="http://<host>:<port>/jasperserver-pro/rest_v2/organizations/.../users/.../attributes">http://<host>:<port>/jasperserver-pro/rest_v2/organizations/.../users/.../attributes <a href="http://<host>:<port>/jasperserver-pro/rest_v2/organizations/.../roles">http://<host>:<port>/jasperserver-pro/rest_v2/organizations/.../roles <a href="http://<host>:<port>/jasperserver-pro/rest_v2/attributes">http://<host>:<port>/jasperserver-pro/rest_v2/attributes
* Available only in commercial editions of JasperReports Server.	

For programmers creating a client application, the reference chapters in this guide give the full description of the methods supported by each REST service, the path or resource expected for each method, and the parameters that are required or optional in the URL. The description of each method includes an example of the descriptors it uses and a sample of the return value.

For tools that can parse the Web Application Description Language (WADL), the following URL gives a machine-readable XML description of all supported REST v2 services:

[http://<host>:<port>/jasperserver\[-pro\]/rest_v2/application.wadl](http://<host>:<port>/jasperserver[-pro]/rest_v2/application.wadl)

1.2 Sending REST Requests from a Browser

Normally, you program your client application to send REST requests to your instance of JasperReports Server. You may also want to test certain requests or examine the response from the server, and some browsers have plug-ins to send a REST request and view the response.

However, the server includes cross-session request forgery (CSRF) protection that does not allow requests, including REST, from a browser in a different domain. Sending POST, PUT, or DELETE requests from a browser will often fail for this reason. REST requests from REST-client applications are secure and are not stopped by CSRF protection.

To allow testing of the REST API through a browser, configure your browser REST client to include the following header in every request:

X-REMOTE-DOMAIN: 1

1.3 HTTP Response Codes

JasperReports Server REST services return standard HTTP status codes. In case of an error, a detailed message may be present in the body as 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. Each service returns typical success and error messages that are given in the reference chapter for that service.

Table 1-2 HTTP Response 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		

1.4 Deprecated Web Services

The server's first REST API (now called v1) is deprecated. These services are no longer supported, do not work with the latest features of the server, and are never guaranteed to succeed. Note that meanings of PUT and POST were reversed in the REST v1 API.

Table 1-3 Deprecated REST v1 Services

Web Service	URLs
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
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/jobsummary">http://<host>:<port>/jasperserver[-pro]/rest/jobsummary <a href="http://<host>:<port>/jasperserver[-pro]/rest/job">http://<host>:<port>/jasperserver[-pro]/rest/job
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/attribute">http://<host>:<port>/jasperserver[-pro]/rest/attribute <a href="http://<host>:<port>/jasperserver[-pro]/rest/role">http://<host>:<port>/jasperserver[-pro]/rest/role
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/user">http://<host>:<port>/jasperserver-pro/rest/user <a href="http://<host>:<port>/jasperserver-pro/rest/attribute">http://<host>:<port>/jasperserver-pro/rest/attribute <a href="http://<host>:<port>/jasperserver-pro/rest/role">http://<host>:<port>/jasperserver-pro/rest/role
* Available only in commercial editions of JasperReports Server.	

The original SOAP web services at the following URLs are also deprecated and no longer supported. The SOAP web services will no longer be maintained or updated to work with new features of the server. In particular, the SOAP web services do not support interactive charts or interactive HTML5 tables. Though the server may still respond to these methods, they are never guaranteed to work.

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

Table 1-4 Deprecated SOAP Web Services

Edition	Web Service	URL
Community Project	Repository	<a href="http://<host>:<port>/jasperserver/services/repository">http://<host>:<port>/jasperserver/services/repository
	Scheduling	<a href="http://<host>:<port>/jasperserver/services/ReportScheduler">http://<host>:<port>/jasperserver/services/ReportScheduler
	Administration	<a href="http://<host>:<port>/jasperserver/services/UserAndRoleManagementService">http://<host>:<port>/jasperserver/services/UserAndRoleManagementService

Edition	Web Service	URL
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

CHAPTER 2 THE serverInfo SERVICE

The `rest_v2/serverInfo` service returns the same information as the **About JasperReports Server** link in the user interface.

Use the following methods to verify the server information, such as version number and supported features for compatibility with your REST client application. Your application should also use the date and date-time patterns to interpret all date or date-time strings it receives from the server and to format all date and date-time strings it sends to the server.

Method	URL
GET	<code>http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo</code>
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:

```
<serverInfo>
  <build>20141121_1750</build>
  <dateFormatPattern>yyyy-MM-dd</dateFormatPattern>
  <datetimeFormatPattern>yyyy-MM-dd'T'HH:mm:ss</datetimeFormatPattern>
  <edition>PRO</edition>
  <editionName>Enterprise</editionName>
  <features>Fusion AHD EXP DB AUD ANA MT </features>
  <licenseType>Commercial</licenseType>
  <version>6.0.0</version>
</serverInfo>

{
  "dateFormatPattern": "yyyy-MM-dd",
```

```
"datetimeFormatPattern": "yyyy-MM-dd'T'HH:mm:ss",
"version": "6.0.0",
"edition": "PRO",
"editionName": "Enterprise",
"licenseType": "Commercial",
"build": "20150527_1942",
"features": "Fusion AHD EXP DB AUD ANA MT "
}
```

You can access each value separately with the following URLs. Note that some information does not apply to community editions of the server. The response is the raw value, XML or JSON are not accepted formats.

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

CHAPTER 3 AUTHENTICATION METHODS

This chapter demonstrates several ways for REST client applications to authenticate with JasperReports Server. Choose an authentication method that matches the usage patterns and needs of your REST client application.

This chapter includes the following sections:

- **Overview of REST Authentication**
- **HTTP Basic Authentication**
- **The login Service**
- **Login Encryption**
- **The Security Check**
- **Using an SSO Token**
- **Using a Pre-Authentication Token**
- **Logout**

3.1 Overview of REST Authentication

When using the REST API, the client application must provide a valid user ID and password to JasperReports Server. The REST services 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 described in the next section.
- Several login methods that, once authenticated, 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.

Normally, RESTful implementations do not rely on the use of persistent sessions, such as the login service and user sessions stored on the server. However, the JasperReports Server architecture automatically creates user sessions internally, and the login methods take advantage of this. There are several use cases for either type of authentication.

- If a username or password contain UTF-8 characters, they may be corrupted by basic authentication and the services will always return an error. In this case, use the login service as described in [3.3, “The login Service,” on page 16](#).
- With basic authentication, the server verifies user credentials for each API request. If your client applications perform many requests in a short time, you can avoid the overhead by using the login service or the security check and passing the session ID instead. For more information, see [3.5, “The Security Check,” on page 19](#).

- If your client makes sporadic requests, for example running a report every hour, it is easier to use basic authentication and send the credentials in the same request.
- However, sessions are kept for 20 minutes by default, so if your client makes a request every 15 minutes with the same credentials, the corresponding session will be kept in memory indefinitely. This can be a problem if you have many different clients running large reports, because some report output is stored in the user session, and they can fill up the available memory. In this case, you should use the logout call to make sure the memory is freed. For more information, see [3.8, “Logout,” on page 21](#).
- If your server is configured with external authentication, repeatedly verifying credentials may be a performance issue you can avoid with the login service.
- If your server is configured with SSO (Single Sign-On), you should use the security check method. For more information, see [3.6, “Using an SSO Token,” on page 20](#).
- If your server is configured with Pre-Authentication, see [3.7, “Using a Pre-Authentication Token,” on page 21](#).

3.2 HTTP Basic Authentication

In HTTP basic authentication, the client application must supply a valid user and password in every service request. 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 default organization admin’s 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.

- In commercial editions where there is only one organization, such as in the JasperReports Server default installation, you should specify the user ID without any qualifiers, for example `jasperadmin`.
- In commercial deployments with multiple organizations, the organization ID or organization alias must be appended to the user ID, for example `jasperadmin|organization_1` or `jasperadmin|org2`.

If login encryption is enabled in your server, you must encrypt the password before base64-encoding it with the username. For more information about encryption, see [3.4, “Login Encryption,” on page 18](#).

3.3 The login Service

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

Method	URL
POST	<code>http://<host>:<port>/jasperserver[-pro]/rest/login/</code>
GET	<code>http://<host>:<port>/jasperserver[-pro]/rest/login?<arguments></code>

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 encoding for the character).
j_password?	Text	The user's password. The argument is optional but authentication will fail without the password. If the server has login encryption enabled, the password must be encrypted as explained in 3.4, "Login Encryption," on page 18.
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
```

Maintaining a session with cookies is not mandatory, and your application can use any combination of session cookie, HTTP Basic Authentication, or both. However, if you use the session ID, it is good practice to close the session as described in [3.8, “Logout,” on page 21](#). Closing the session frees up any associated resources in memory.

3.4 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 Security 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 REST service request.
- 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 request using HTTP Basic Authentication or every session using the login service.

After using this service to obtain the server's public key, your client must encrypt the user's password with the public key using the Bouncy Castle library and the RSA/NONE/NoPadding algorithm. Then your client can send the encrypted password in simple authentication or using the login service.

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

Method	URL
GET	<code>http://<host>:<port>/jasperserver[-pro]/GetEncryptionKey/</code>

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}

3.5 The Security Check

The security check request is equivalent to a login request. You send the user credentials and you can tell from the response whether they are valid or not on the server. If they are valid, the server creates a user session or if the user has already performed an operation with valid credentials, it accesses the existing user session.

In either case, the successful response contains the JSESSIONID cookie of the user session. As with the login service, once you receive the session cookie, you should return it with future requests and use it to close the session as described in [3.8, “Logout,” on page 21](#).

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/j_spring_security_check?<arguments>	
Argument	Type/Value	Description
orgId?	Text	The organization ID or alias. Required for organization admins and users when there is more than one organization defined. Not required for the system admin (<i>superuser</i> by default).
j_username	Text	The user ID.
j_password	Text	The user's password. If the server has login encryption enabled, the password must be encrypted as explained in 3.4, “Login Encryption,” on page 18 .
userLocale?	Java locale string	Set the optional locale for user in this session.
userTimezone?	Java time zone	Set the optional time zone for the user in this session.
Options		
accept: application/json		

Return Value on Success	Typical Return Values on Failure
302 Moved Temporarily – Response HTTP Header "Location" redirects to /loginsuccess.html by default, but often depends on the last session operation. See below if you specify JSON.	302 Moved Temporarily – Response HTTP Header "Location" redirects to /login.html?error=1.

If you specify `accept: application/json` in your request, the location of the redirect in case of success is always the file `/scripts/bower_components/js-sdk/src/common/auth/loginSuccess.json`. The content of this file is:

```
{
  "success":true
}
```

You can configure the location of this file. Edit the configuration file `.../WEB-INF/applicationContext-security-web.xml` and change the constructor value of the following bean:

```
<bean id="authSuccessJsonRedirectUrl" class="java.lang.String">

  <constructor-arg type="java.lang.String"
    value="/scripts/bower_components/js-sdk/src/common/auth/loginSuccess.json"/>

</bean>
```

3.6 Using an SSO Token

If you are using Single Sign-On for authentication, you can use the security check to submit the ticket.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/j_spring_security_check?<arguments>	
Argument	Type/Value	Description
ticket	Text	The ticket for your SSO mechanism. The default parameter name for an SSO authentication token is <code>ticket</code> . This parameter name can be changed in the configuration file <code>applicationContext-externalAuth-<sso>.xml</code> .
Options		
accept: application/json		
Return Value on Success	Typical Return Values on Failure	
302 Moved Temporarily – Response HTTP Header "Location" redirects to /loginsuccess.html by default, but often depends on the last session operation.	302 Moved Temporarily – Response HTTP Header "Location" redirects to /login.html?error=1.	

For example, if you have configured the server to use CAS as your SSO provider, you can authenticate and receive the session ID with the following request:

```
GET http://localhost:8080/jasperserver-pro/j_spring_security_check?ticket=ST-40-CZeUUnGPxEggScNbxh91-
sso-cas.eng.jaspersoft.com
```

The response has the same behavior as the password-based security check, including the use of a JSON file if requested.

3.7 Using a Pre-Authentication Token

When using a pre-authentication mechanism, the verification of the credentials is performed at the base URL of the server.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/?<arguments>	
Argument	Type/Value	Description
pp	Text	The token for your pre-authentication mechanism. The default parameter name for a pre authentication token is <code>pp</code> . This parameter name can be changed in the configuration file <code>.../WEB-INF/applicationContext-externalAuth-preAuth.xml</code> .
Options		
accept: application/json		
Return Value on Success		Typical Return Values on Failure
302 Moved Temporarily – Response HTTP Header "Location" redirects to <code>/loginsuccess.html</code> by default, but often depends on the last session operation.		302 Moved Temporarily – Response HTTP Header "Location" redirects to <code>/login.html?error=1</code> .

For example, if you have configured the server to use pre-authentication, you can authenticate and receive the session ID with the following request:

```
GET http://localhost:8080/jasperserver-pro?pp=u%3DSteve%7Cr%3DExt_User%7Co%3Dorganization_
1%7Cpa1%3DUSA%7Cpa2%3D1
```

3.8 Logout

While REST calls are often stateless, JasperReports Server uses a session to hold some information such as generated reports. The session and its report data take up space in memory and it's good practice to explicitly close the session when it is no longer needed. This allows the server to free up and reuse resources much faster.

To close a session and free its resources, invoke the logout page and include the `JSESSIONID` cookie in the request.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/logout.htm
Header	
Cookie: \$Version=0; JSESSIONID=52E79BCEE51381DF32637EC69AD698AE; \$Path=/jasperserver	

CHAPTER 4 WORKING WITH RESOURCES

The JasperReports Server repository stores the resources such as data sources and reports that REST clients can interact with. Before you can use the `rest_v2/resources` service to access the repository, you should understand how resources are represented. This chapter introduces concepts that are common to all resources as well as complex topics such as nested resources.

For further information, see:

- **Chapter 5, “Resource Descriptors,” on page 31** for a reference to every type of resource and its attributes.
- **Chapter 6, “The resources Service,” on page 45** for methods to operate on resources in the repository.

This chapter includes the following sections:

- **Resource URI**
- **Custom Media Types**
- **Accept HTTP Headers**
- **Content-Type HTTP Headers**
- **JSON Format**
- **Nested Resources**
- **Referenced Resources**
- **Local Resources**
- **Optimistic Locking**
- **Update-only Passwords**

4.1 Resource URI

Resources (such as reports, images, queries, and data sources) are stored in the server's repository. The repository 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 report is a resource of type `reportUnit`, and images are of type `file`.

Every resource has an ID that is unique within the folder where it resides. The IDs of all parent folders create a path, and appending the resource's own ID to the path gives the URI (Universal Resource Identifier) of the resource in the repository. Resource descriptors do not have an explicit ID attribute, but the ID is always the last component of the URI field in responses from the server.

In commercial editions of the server, the URI of a resource is relative to the organization of the user whose credentials are used to authenticate the request. Thus the path `/datasources/JServerJdbcDS` for an organization_1

user is the same resource as the path `/organizations/organization_1/datasources/JServerJdbcDS` for the system admin (`superuser`). The `/public` folder is a special path that is absolute for any user in any organization (including `superuser`).



As with all server operations, the folders and resources that are visible and accessible to a given user depend on permissions that are set in the repository on those folders and resources. REST services return an error when attempting an operation on resources that the authenticated user does not have permission to access.

The URI and ID of a created resource is determined in one of the following ways:

- POST operations on the resources service specify a folder. The resource descriptor in the request is created in the specified folder. The ID is created automatically from the label of the resource by replacing special characters with underscores (`_`). The URI of the new resource is returned in the server's response and consists of the target folder with the automatic ID appended to it.
- PUT operations on the resources service send a descriptor to create the resource at the URI specified in the request. The resource ID is the last element of this URI, as long as it is unique in the parent folder. The server's response should confirm that the resource was successfully created with the requested URI.

All resources also have a label string and a description string that can be presented to your client's users. The label and description support special characters (such as spaces, punctuation, and accented characters) and even Unicode if configured in your server during installation.

4.2 Custom Media Types

In order to specify all the different types of resources, the resources service relies on custom media types with the following syntax:

`application/repository.<resourceType>+<format>`

where:

- `<resourceType>` is the name for each type of repository resource, such as `reportUnit`, `dataType`, or `jdbcDataSource`. The names of all supported types are given in [Chapter 5, “Resource Descriptors,” on page 31](#).
- `<format>` is the representation format of the descriptor, either `json` or `xml`.

For example:

`application/repository.dataType+json` – JSON representation of a datatype resource

`application/repository.reportUnit+xml` – XML representation of a JRXML report

The custom media types should be used in Content-Type and Accept HTTP headers, as described in the following sections. According to the HTTP specification, headers should be case insensitive; the headers and custom media types can be upper case, lower case, or any mixture of upper and lower case.

4.3 Accept HTTP Headers

Client applications should use the Accept HTTP header in a request to specify the desired format in the server's response. Generally, regardless of the resource type, it's enough to specify:

- Accept: `application/json` to get response in JSON format or
- Accept: `application/xml` to get response in XML format.

The server will respond with the specific custom media type for the requested resource, as described in the next section.

However, there are some special cases where client must specify a precise resource type:

- When requesting the resource details of the root folder, client must specify `application/repository.folder+<format>` to get its resource descriptor. Otherwise, the request is considered a search of the root folder.
- When requesting the resource details of a file resource, as opposed to the file contents, the client must specify `application/repository.file+<format>`. Without this Accept header, the response will contain the file contents. The custom media type also distinguishes between the XML descriptor of a file and the contents of an XML file.

If the client specifies a custom type in the Accept header that does not match the resource being requested, the server responds with the error code 406 Not Acceptable.

4.4 Content-Type HTTP Headers

The Content-Type HTTP header indicates the media type being sent in the body of the request or response. For example, if the client requests a valid datatype resource, and depending on the format that the client specified in the Accept header of the request, the server's response includes:

- Content-Type: `application/repository.dataType+json` or
- Content-Type: `application/repository.dataType+xml`

When the client uploads a resource descriptor to create or update a resource, it must set the Content-Type connector accurately. For example, when uploading a datatype resource represented in XML, the client must send:

Content-Type: `application/repository.dataType+xml`

The server relies on the Content-Type header to parse the body of the request, and it will respond with the error code 400 Bad Request if there is a mismatch. In the example above, the following headers will result in an error:

- Content-Type: `application/xml` – custom media type not included
- Content-Type: `application/repository.reportUnit+xml` – media type mismatch
- Content-Type: `application/repository.dataType+json` – format mismatch

4.5 JSON Format

JasperReports Server uses the standard JSON (JavaScript Object Notation) format to send and receive representations of resources and other structures. The JSON marshalling and unmarshalling (parsing) uses the following conventions:

- Attributes with no value or a null value are not transmitted in a request.
- Unknown properties that JasperReports Server does not recognize are ignored without error.
- Dates should be given in [ISO 8601](#) format.

4.6 Nested Resources

Many types of resources in the repository are defined in terms of other resources. For example, some types of input controls require a query, and the query itself requires a data source. The nested query and data source can be defined in two ways:

- Referenced resources - a link to a valid resource defined elsewhere in the repository. JasperReports Server manages the references between resources by enforcing permissions and protecting dependencies from deletion.
- Local resources - a resource descriptor nested within the parent descriptor. The nested resource is fully defined within the parent resource and not available for being referenced from elsewhere.

Both types of nested resources are further described in the following sections.

4.7 Referenced Resources

Referenced resources are defined by special structures within the descriptors of other resources. For example, in the following query resource, the data source field contains a `dataSourceReference` object that contains the URI of the target reference:

```
{
  "version": 0,
  "permissionMask": 1,
  "creationDate": "2013-10-03T16:32:37",
  "updateDate": "2013-10-03T16:32:37",
  "label": "Country Query",
  "description": null,
  "uri": "/adhoc/topics/Cascading_multi_select_topic_files/Country_multi_select_files/
    country_query",
  "dataSource": { contents }, <*>
  "value": "select distinct billing_address_country from accounts order by billing_address_country",
  "language": "sql"
}

<*> or "dataSourceReference": {
  "uri": "/datasources/JServerJNDIDS"
},
```

To create referenced resources, send requests to the server that contain the appropriate reference objects for the target resource. See [4.7, “Referenced Resources,” on page 26](#) for the specific reference objects available in each resource descriptor.

When reading resources with referenced resources, the `uri` attribute gives the repository URI of the reference. To simplify the parsing of referenced resources, the resources service GET method supports the `expanded=true` parameter. Instead of following references and requiring two or more GET requests, the `expanded=true` parameter returns all referenced resources fully expanded within the parent resource, as if it were a local resource.

The following resource types support referenced resources, and the table gives the name of the field that contains the referenced URI, and the name of the expanded type that replaces the reference.

Resource Type	Reference Attribute(s)	Expanded Name and Descriptor
query	dataSourceReference	awsDataSource, beanDataSource, customDataSource, jdbcDataSource, jndiJdbcDataSource, virtualDataSource, semanticLayerDataSource or advDataSource (adhocDataView)
inputControl	datatypeReference	dataType
	listOfValuesReference	listOfValues
	queryReference	query
reportUnit	jrxmlFileReference	jrxmlFile with file attributes
	dataSourceReference	see query dataSourceReference
	queryReference	query
	inputControlReference	inputControl
	fileReference (images, ...)	fileResource with file attributes
semanticLayerDataSource (Domain)	dataSourceReference	see query dataSourceReference
	schemaFileReference	schemaFile with file attributes
	fileReference (bundle)	file of appropriate type
	securityFileReference	securityFile with file attributes
olapUnit	olapConnectionReference	xmlaConnection, mondrianConnection, or secureMondrianConnection
mondrianConnection	dataSourceReference	see query dataSourceReference
	schemaReference	schema with file attributes
secureMondrianConnection	dataSourceReference	see query dataSourceReference
	schemaReference	schema with file attributes
	accessGrantSchemaReference	accessGrantSchema with file attributes
mondrianXmlaDefinition	mondrianConnectionReference	mondrianConnection or secureMondrianConnection

4.8 Local Resources

Nested resources that are not referenced resources must be defined locally within the parent resource. The nested resource is defined by a complete resource descriptor of the appropriate type. The following example shows a data source that is defined locally within the parent query resource:

```
{
  "version": 0,
  "permissionMask": 1,
  "creationDate": "2013-10-03T16:32:37",
  "updateDate": "2013-10-03T16:32:37",
  "label": "Country Query",
  "description": null,
  "uri": "/adhoc/topics/Cascading_multi_select_topic_files/Country_multi_select_files/country_query",
  "dataSource": {
    "jndiJdbcDataSource": {
      "version": 0,
      "permissionMask": 1,
      "creationDate": "2013-10-03T16:32:05",
      "updateDate": "2013-10-03T16:32:05",
      "label": "my JNDI ds",
      "description": "Local JNDI Data Source",
      // URI of expanded nested resource is ignored. Resource is created locally
      "uri": "/datasources/JServerJNDIDS",
      "jndiName": "jdbc/sugarcrm",
      "timezone": null
    }
  },
  "value": "select distinct billing_address_country from accounts order by billing_address_country",
  "language": "sql"
}
```

Use nested descriptors such as the ones above to create resources that contain local resources. Descriptors can be nested to any level, as long as the syntax of each descriptor is valid. See [4.6, “Nested Resources,” on page 26](#) for the correct syntax of both the parent and the nested resource.

Internally, the resources service handles local resources as normal resources contained in a hidden folder. The hidden folder containing local resources has the following name:

<parentURI>_files/

and local resources can be accessed at the following URI:

<parentURI>_files/<resourceID>

In the example above, we can see that the parent query resource is a nested resource itself. Its URI shows us that it is the query resource for a query-based input-control of a topic resource:

/adhoc/topics/Cascading_multi_select_topic_files/Country_multi_select_files/country_query

and the new nested data source will have the following URI:

/adhoc/topics/Cascading_multi_select_topic_files/Country_multi_select_files/country_query_files/my_JNDI_ds

The ID of the nested resource (my_JNDI_ds) is created automatically from the label of the nested resource.

The _files folder that exists in all parents of local resources is hidden so that its local resources do not appear in repository searches. You can set the `showHiddenItems=true` parameter on the resources request to search for a _files folder in all local resources, such as in a JRXML report (reportUnit).

Local resources in the hidden `_files` folder can also be created and updated separately from their parent resources by using PUT and POST methods of the resources service and specifying the complete URI of the local resource as shown above.

4.9 Optimistic Locking

The resources service supports optimistic locking on all write and update operations (PUT, POST, and PATCH). When using the service to search the repository and receive descriptors of the resources, all descriptors contain a version number field. Clients should return the same version number when writing or updating a given resources. The server compares the version number in the modify request the current version of the resource to assure that no other client has updated the same resource.

If the version numbers do not match, the server replies with error code 409 Conflict. In that case, the client should request the resource again (read operation with GET) and send the modify request with an updated version number.

When a modify operation is successful, the server increments the version number on the affected resource and returns the new descriptor with the new version as confirmation that the operation was successful.

4.10 Update-only Passwords

Some resource descriptors such as `jdbcDataSource` and `xmlaConnection` contain a password field. All password fields are blank or missing when reading (GET) a resource descriptor. This prevents anyone, even administrators from seeing existing passwords.

Write or update operations (PUT or POST) may send the password field in descriptors that support it. In this case, the password value is updated in the resource in the repository. Make sure that resources with sensitive passwords have the proper permissions so that only authorized users can modify them.

For complete security, you should only send passwords over HTTPS connections, otherwise they appear unencrypted in network packets.

CHAPTER 5 RESOURCE DESCRIPTORS

This chapter provides a reference by example for every type of resource descriptor that exists in the repository. Use the resources service to get and set resources with these descriptors. For further information, see:

- **Chapter 4, “Working With Resources,” on page 23** for general guidelines about using descriptors.
- **Chapter 6, “The resources Service,” on page 45** for methods to operate on resources in the repository.

This chapter does not cover descriptors for objects that are not stored in the repository. Descriptors that represent jobs, calendars, organizations, roles, users, and attributes are described with the service that operates on them.

This chapter includes the following sections:

- **Common Attributes**
- **Folder**
- **JNDI Data Source**
- **JDBC Data Source**
- **AWS Data Source**
- **Virtual Data Source**
- **Custom Data Source**
- **Bean Data Source**
- **Datatypes**
- **List of Values**
- **Query**
- **Input Control**
- **File**
- **Report Unit (JRXML Report)**
- **Report Options**
- **Domain (semanticLayerDataSource)**
- **Domain Topic**
- **XML/A Connection**
- **Mondrian Connection**
- **Secure Mondrian Connection**
- **OLAP Unit**
- **Mondrian XML/A Definition**
- **Other Types**

5.1 Common Attributes

All resource types contain the following attributes. Of these common attributes, only the label and description fields are writable.

In general, writable fields are ones that can be set by the client when sending a descriptor for a write or update operation (PUT or POST). The other fields are read-only fields that the server sets automatically.

application/repository.{resourceType}.json	application/repository.{resourceType}.xml
<pre>{ "uri" : "/sample/resource/uri", "label": "Sample Label", "description": "Sample Description", "permissionMask": "0", "creationDate": "2013-07-04T12:18:47", "updateDate": "2013-07-04T12:18:47", "version": "0" ... }</pre>	<pre><?xml version="1.0" encoding="UTF-8" standalone="yes"?> <{resourceType}> <uri>/sample/resource/uri</uri> <label>Sample Label</label> <description>Sample Description </description> <permissionMask>0</permissionMask> <creationDate>2013-07-04T12:18:47 </creationDate> <updateDate>2013-07-04T12:18:47 </updateDate> <version>0</version> ... </{resourceType}></pre>

Throughout the rest of the resource type sections, the common attributes are included in every descriptor as {commonAttributes}.

5.2 Folder

Folder types do not contain any additional fields beyond the common attributes shown above.

application/repository.folder.json	application/repository.folder.xml
<pre>{ "uri" : "{resourceUri}", "label": "Sample Label", "description": "Sample Description", "permissionMask": "0", "creationDate": "2013-07-04T12:18:47", "updateDate": "2013-07-04T12:18:47", "version": "0" }</pre>	<pre><folder> <uri>/sample/resource/uri</uri> <label>Sample Label</label> <description>Sample Description </description> <permissionMask>0</permissionMask> <creationDate>2013-07-04T12:18:47 </creationDate> <updateDate>2013-07-04T12:18:47 </updateDate> <version>0</version> </folder></pre>

Only the label and description fields are writable.

5.3 JNDI Data Source

application/repository.jndiJdbcDataSource+json	application/repository.jndiJdbcDataSource+xml
<pre>{ {commonAttributes}, "jndiName": "{jndiName}", "timezone": "{timezone}" }</pre>	<pre><jndiDataSource> {commonAttributes} <jndiName>{jndiName}</jndiName> <timezone>{timezone}</timezone> </jndiDataSource></pre>

5.4 JDBC Data Source

application/repository.jdbcDataSource+json	application/repository.jdbcDataSource+xml
<pre>{ {commonAttributes}, "driverClass": "{driverClass}", "password": "{password}", "username": "{username}", "connectionUrl": "{connectionUrl}", "timezone": "{timezone}" }</pre>	<pre><jdbcDataSource> {commonAttributes} <driverClass>{driverClass}</driverClass> <password>{password}</password> <username>{username}</username> <connectionUrl> {connectionUrl} </connectionUrl> <timezone>{timezone}</timezone> </jdbcDataSource></pre>

5.5 AWS Data Source

application/repository.awsDataSource+json	application/repository.awsDataSource+xml
<pre>{ {commonAttributes}, "driverClass": "{driverClass}", "password": "{password}", "username": "{username}", "connectionUrl": "{connectionUrl}", "timezone": "{timezone}", "accessKey": "{accessKey}", "secretKey": "{secretKey}", "roleArn": "{roleArn}", "region": "{region}", "dbName": "{dbName}", "dbInstanceIdentifier": "{dbInstanceIdentifier}", "dbService": "{dbService}" }</pre>	<pre><awsDataSource> {commonAttributes} <driverClass>{driverClass}</driverClass> <password>{password}</password> <username>{username}</username> <connectionUrl> {connectionUrl} </connectionUrl> <timezone>{timezone}</timezone> <accessKey>{accessKey}</accessKey> <secretKey>{secretKey}</secretKey> <roleArn>{roleArn}</roleArn> <region>{region}</region> <dbName>{dbName}</dbName> <dbInstanceIdentifier> {dbInstanceIdentifier} </dbInstanceIdentifier> <dbService>{dbService}</dbService> </awsDataSource></pre>

The {region} values are specified in the file `.../WEB-INF/application-context.xml`, with their corresponding display labels defined in `.../WEB-INF/bundles/jasperserver_messages.properties`. By default, the following regions are defined:

Values of AWS {region} in <code>.../WEB-INF/application-context.xml</code>	Labels for AWS regions in <code>.../WEB-INF/bundles/jasperserver_messages.properties</code>
<code>us-east-1.amazonaws.com</code>	US East (Northern Virginia) Region
<code>us-west-2.amazonaws.com</code>	US West (Oregon) Region
<code>us-west-1.amazonaws.com</code>	US West (Northern California) Region
<code>eu-west-1.amazonaws.com</code>	EU (Ireland) Region
<code>eu-central-1.amazonaws.com</code>	EU (Frankfurt) Region
<code>ap-southeast-1.amazonaws.com</code>	Asia Pacific (Singapore) Region
<code>ap-southeast-2.amazonaws.com</code>	Asia Pacific (Sydney) Region
<code>ap-northeast-1.amazonaws.com</code>	Asia Pacific (Tokyo) Region
<code>sa-east-1.amazonaws.com</code>	South America (São Paulo) Region

5.6 Virtual Data Source

The `id` of each `subDataSource` must be unique. The server does not prevent duplicates, and the last one to be defined silently overwrites the previous definition.

<code>application/repository.virtualDataSource+json</code>	<code>application/repository.virtualDataSource+xml</code>
<pre>{ {commonAttributes}, "subDataSources": [{ "id": "{subDataSourceId}", "uri": "{subDataSourceUri}" }, ...] }</pre>	<pre><virtualDataSource> {commonAttributes} <subDataSources> <subDataSource> <id>{subDataSourceId}</id> <uri>{subDataSourceUri}</uri> </subDataSource> ... </subDataSources> </virtualDataSource></pre>

5.7 Custom Data Source

The value of the `serviceClass` attribute is read-only and depends on the specific type of the custom data source, as defined in the server's `applicationContext` configuration files.

application/repository.customDataSource+json	application/repository.customDataSource+xml
<pre>{ {commonAttributes}, "serviceClass": "{serviceClass}", "dataSourceName": "{dataSourceName}", "properties": [{ "key": "{key}", "value": "{value}" }, ...] }</pre>	<pre><customDataSource> {commonAttributes} <serviceClass> {serviceClass} </serviceClass> <dataSourceName> {dataSourceName} </dataSourceName> <properties> <property> <key>{key}</key> <value>{value}</value> </property> ... </properties> </customDataSource></pre>

5.8 Bean Data Source

application/repository.beanDataSource+json	application/repository.beanDataSource+xml
<pre>{ {commonAttributes}, "beanName": "{beanName}", "beanMethod": "{beanMethod}" }</pre>	<pre><beanDataSource> {commonAttributes} <beanName>{beanName}</beanName> <beanMethod>{beanMethod}</beanMethod> </beanDataSource></pre>

5.9 Datatypes

application/repository.dataType+json	application/repository.dataType+xml
<pre>{ {commonAttributes}, "type": "text number date dateTime time", "pattern": "{pattern}", "maxValue": "{maxValue}", "strictMax": "true false", "minValue": "{minValue}", "strictMin": "true false", "maxLength": "{maxLengthInteger}" }</pre>	<pre><dataType> {commonAttributes} <type>text number date dateTime time</type> <pattern>{pattern}</pattern> <maxValue>{maxValue}</maxValue> <strictMax>true false</strictMax> <minValue>{minValue}</minValue> <strictMin>true false</strictMin> <maxLength>{maxLengthInteger}</maxLength> </dataType></pre>

5.10 List of Values

application/repository.listOfValues+json	application/repository.listOfValues+xml
<pre>{ {commonAttributes}, "items":[{ "label":"{label}", "value":"{value}" }, ...] }</pre>	<pre><listOfValues> {commonAttributes} <items> <item> <label>{label}</label> <value>{value}</value> </item> ... </items> </listOfValues></pre>

5.11 Query

The `dataSource` field of the query may be null. Set an empty `dataSource` field when you want to remove a local data source, either a reference or a local definition. When the data source of a query is not defined, the query uses the data source of its parent, for example its JRXML report (`reportUnit`).

application/repository.query+json	application/repository.query+xml
<pre>{ {commonAttributes}, "value":"{query}", "language":"{language}", "dataSource":{ "dataSourceReference": { "uri":"{dataSourceUri}" } } }</pre>	<pre><query> {commonAttributes} <value>{query}</value> <language>{language}</language> <dataSourceReference> <uri>{dataSourceUri}</uri> </dataSourceReference> </query></pre>

5.12 Input Control

Input controls come in several types that require different fields. The following table shows all possible fields, not all of which are mutually compatible.

application/repository.inputControl+json	application/repository.inputControl+xml
<pre> { {commonAttributes}, "mandatory": "{true false}", "readOnly": "{true false}", "visible": "{true false}", "type": "{inputControlTypeByteValue}", "usedFields": "{field1;field2;...}", "dataType": { "dataTypeReference": { "uri": "{dataTypeResourceUri}" } }, "listOfValues": { "listOfValuesReference": { "uri": "listOfValuesResourceUri" } } "visibleColumns": ["column1", "column2", ...], "valueColumn": "{valueColumn}", "query": { "queryReference": { "uri": "{queryResourceUri}" } } } </pre>	<pre> <inputControl> {commonAttributes} <mandatory>true false</mandatory> <readOnly>true false</readOnly> <visible>true false</visible> <type>{inputControlTypeByteValue}</type> <usedFields> {field1;field2;...}</usedFields> <dataTypeReference> <uri>{dataTypeResourceUri}</uri> </dataTypeReference> <listOfValuesReference> <uri>{listOfValuesResourceUri}</uri> </listOfValuesReference> <queryReference> <uri>{queryResourceUri}</uri> </queryReference> <visibleColumns> <column>{column1}</column> <column>{column2}</column> <column>...</column> </visibleColumns> <valueColumn>{valueColumn}</valueColumn> </inputControl> </pre>

The following list shows the numerical code and meaning for {inputControlTypeByteValue}. The input control type determines the other fields that are required. The list of required fields may appear in a field named usedFields, separated by semi-colons (;).

Type	Type of Input Control	Other Fields Required (usedFields)
1	Boolean	None
2	Single value	dataType
3	Single-select list of values	listOfValues
4	Single-select query	query; queryValueColumn
5	<i>Not used</i>	
6	Multi-select list of values	listOfValues
7	Multi-select query	query; queryValueColumn
8	Single-select list of values radio buttons	listOfValues
9	Single-select query radio buttons	query; queryValueColumn

Type	Type of Input Control	Other Fields Required (usedFields)
10	Multi-select list of values check boxes	listOfValues
11	Multi-select query check boxes	query; queryValueColumn

5.13 File

The `repository.file+<format>` descriptor is used to identify the file type. The content field is used only when uploading a file resource as base-64 encoded content. For other ways to upload file contents, see [7.3, “Uploading File Resources,” on page 61](#). The content field is absent when requesting a file resource descriptor. To download file contents, see [7.2, “Downloading File Resources,” on page 60](#).

<code>application/repository.file+json</code>	<code>application/repository.file+xml</code>
<pre>{ {commonAttributes}, "type":"pdf html xls rtf csv odt txt docx ods xlsx img font jrxml jar prop jrtx xml css olapMondrianSchema accessGrantSchema unspecified}", // content is write-only; // it is not included in a response "content":{"base64EncodedContent}" }</pre>	<pre><file> {commonAttributes} <type>pdf html xls rtf csv odt txt docx ods xlsx img font jrxml jar prop jrtx xml css olapMondrianSchema accessGrantSchema unspecified} </type> <content>{base64EncodedContent}</content> </file></pre>

5.14 Report Unit (JRXML Report)

A report unit contains mostly references to the files that make up a report within the server. A report unit is a composite resource that may contain other local resources. In this case, the URIs that it references include a URI in the following format:

```
<reportUnitURI>_files/<localResourceID>
```

For example, the main JRXML of a sample report is referenced as follows:

```
/reports/samples/Cascading_multi_select_report_files/Cascading_multi_select_report
```

For more information, see [4.6, “Nested Resources,” on page 26](#).

application/repository.reportUnit+json	application/repository.reportUnit+xml
<pre> { {commonAttributes}, "alwaysPromptControls": "{true false}", // default is "popupScreen" "controlsLayout": "{popupScreen separatePage topOfPage inPage}", "inputControlRenderingView": "{inputControlRenderingView}", "reportRenderingView": "{reportRenderingView}", "dataSource":{ "dataSourceReference": { "uri":"{dataSourceUri}" } }, // "query" is nullable "query:" { "queryReference": { uri: "{queryResourceUri}" } }, "jrxml": { "jrxmlFileReference": { "uri": "{jrxmlFileResourceUri}" } or "jrxmlFile": { "label": "Main jrxml", "type": "jrxml" } } "inputControls": [{ "inputControlReference": { "uri": "{inputControlUri}" } }, ...], "resources": ["resource": { "name": "{resourceName}", "file": {contents} <*> } }, ...] } <*> or "fileReference": { "uri": "{fileResourceUri}" } </pre>	<pre> <reportUnit> {commonAttributes} <alwaysPromptControls>true false </alwaysPromptControls> <!-- default is "popupScreen" --> <controlsLayout> popupScreen separatePage topOfPage inPage </controlsLayout> <inputControlRenderingView> {inputControlRenderingView} </inputControlRenderingView> <reportRenderingView> {reportRenderingView} </reportRenderingView> <dataSource> <dataSourceReference> <uri>{dataSourceUri}</uri> </dataSourceReference> </dataSource> <query> <queryReference> <uri>{queryResourceUri}</uri> </queryReference> </query> <jrxml> <jrxmlFileReference> <uri>{jrxmlFileResourceUri}</uri> </jrxmlFileReference> </jrxml> <inputControls> <inputControlReference> <uri>{inputControlUri}</uri> </inputControlReference> ... </inputControls> <resources> <resource> <name>{resourceName}</name> <file>contents</file> {*> </resource> ... </resources> </reportUnit> {*> or <fileReference> <uri>{fileResourceUri}</uri> </fileReference> </pre>

5.15 Report Options

application/repository.reportOptions+json	application/repository.reportOptions+xml
<pre>{ {commonAttributes}, "reportUri":"{reportUri}", "reportParameters":[{ "name":"{parameterName}", "value":["value_1", "value_2", ...] }, ...] }</pre>	<pre><reportOptions> {commonAttributes} <reportUri>{reportUri}</reportUri> <reportParameters> <reportParameter> <name>{parameterName}</name> <value>value_1</value> <value>value_2</value> ... </reportParameter> ... </reportParameters> </reportOptions></pre>

5.16 Domain (semanticLayerDataSource)

For more information about accessing the schema of a Domain, see [Chapter 8, “Working With Domains,”](#) on [page 65](#).

application/repository.semanticLayerDataSource+json	application/repository.semanticLayerDataSource+xml
<pre> { {commonAttributes}, "dataSource":{ "dataSourceReference": { "uri":"{dataSourceUri}" } }, "schema": { "schemaFileReference": { "uri": "{schemaFileResourceUri}" } }, "bundles": [{ // empty localeString indicates default bundle "locale": "{localeString}", "file": { "fileReference": { "uri": "{propertiesFileResourceUri}" } } }, ...], "securityFile": { "securityFileReference": { "uri": "{securityFileResourceUri}" } } } </pre>	<pre> <semanticLayerDataSource> {commonAttributes} <dataSourceReference> <uri>{dataSourceUri}</uri> </dataSourceReference> <schemaFileReference> <uri>{schemaFileResourceUri}</uri> </schemaFileReference> <bundles> <bundle> <!-- <locale/> indicates default bundle --> <locale>{localeString}</locale> <fileReference> <uri>{propertiesFileResourceUri}</uri> </fileReference> </bundle> ... </bundles> <securityFileReference> <uri>{securityFileResourceUri}</uri> </securityFileReference> </semanticLayerDataSource> </pre>

5.17 Domain Topic

A Domain Topic is a Topic created by selecting database fields from a Domain. It is structurally equivalent to a JRXML report, and thus it has the same type attributes (see [Chapter 5, “Resource Descriptors,” on page 31](#)). The only difference is that the data source field will reference a Domain (semanticLayerDataSource).

application/repository.domainTopic+json	application/repository.domainTopic+xml
Same attributes as application/repository.reportUnit+json	Same attributes as application/repository.reportUnit+xml

5.18 XML/A Connection

application/repository.xmlaConnection+json	application/repository.xmlaConnection+xml
<pre>{ {commonAttributes}, "url": "{xmlaServiceUrl}", "xmlaDataSource": "{xmlaDataSource}", "catalog": "{catalog}", "username": "{username}", "password": "{password}" }</pre>	<pre><xmlaConnection> {commonAttributes} <url>{xmlaServiceUrl}</url> <xmlaDataSource> {xmlaDataSource} </xmlaDataSource> <catalog>{catalog}</catalog> <username>{username}</username> <password>{password}</password> </xmlaConnection></pre>

5.19 Mondrian Connection

Mondrian connections without the access grant schemas are used in the Community edition of JasperReports Server.

application/repository.mondrianConnection+json	application/repository.mondrianConnection+xml
<pre>{ {commonAttributes}, "dataSource": { "dataSourceReference": { "uri": "{dataSourceUri}" } }, "schema": { "schemaReference": { "uri": "{schemaFileResourceUri}" } } }</pre>	<pre><mondrianConnection> {commonAttributes} <dataSourceReference> <uri>{dataSourceUri}</uri> </dataSourceReference> <schemaReference> <uri>{schemaFileResourceUri}</uri> </schemaReference> </mondrianConnection></pre>

5.20 Secure Mondrian Connection

Secure Mondrian connections are available only in commercial releases of JasperReports Server.

application/repository.secureMondrianConnection+json	application/repository.secureMondrianConnection+xml
<pre> { {commonAttributes}, "dataSource":{ "dataSourceReference": { "uri":"{dataSourceUri}" } }, "schema": { "schemaReference": { "uri": "{schemaFileResourceUri}" } }, "accessGrantSchemas": [{ "accessGrantSchemaReference": { "uri": "{accessGrantS- chemaFileResourceUri}" } }, ...] } </pre>	<pre> <secureMondrianConnection> {commonAttributes} <dataSourceReference> <uri>{dataSourceUri}</uri> </dataSourceReference> <schemaReference> <uri>{schemaFileResourceUri}</uri> </schemaReference> <accessGrantSchemas> <accessGrantSchemaReference> <uri>{accessGrantS- chemaFileResourceUri}</uri> </accessGrantSchemaReference> </accessGrantSchemas> </secureMondrianConnection> </pre>

5.21 OLAP Unit

application/repository.olapUnit+json	application/repository.olapUnit+xml
<pre> { {commonAttributes}, "mdxQuery":"{mdxQuery}", "olapConnection": { "olapConnectionReference": { "uri": "{olapConnectionReferenceUri}" } } } </pre>	<pre> <olapUnit> {commonAttributes} <mdxQuery>{mdxQuery}</mdxQuery> <olapConnectionReference> <uri>{olapCon- nectionReferenceUri}</uri> </olapConnectionReference> </olapUnit> </pre>

5.22 Mondrian XML/A Definition

application/repository.mondrianXmlaDefinition+json	application/repository.mondrianXmlaDefinition+xml
<pre>{ {commonAttributes}, "catalog": "{catalog}", "mondrianConnection": { "mondrianConnectionReference": { "uri": "{mondrianConnectionResourceUri}" } } }</pre>	<pre><mondrianXmlaDefinition> {commonAttributes} <catalog>{catalog}</catalog> <mondrianConnectionReference> <uri>{mondrianConnectionResourceUri}</uri> </mondrianConnectionReference> </mondrianXmlaDefinition></pre>

5.23 Other Types

The following types are defined in commercial editions of the server and appear in the repository. However, they are meant only to describe the corresponding resources as read-only objects in the repository. The REST API does not support services for clients to create or modify these types.

The types in the following table contain only the common attributes described in [Chapter 5, “Resource Descriptors,”](#) on page 31.

Type String	Description
application/repository.dashboard+json application/repository.dashboard+xml	The dashboard resource descriptors are deprecated and subject to change.
application/repository.adhocDataView+json application/repository.adhocDataView+xml	The Ad Hoc view type is not fully defined yet and subject to change. Ad Hoc views may be referenced as data sources in other repository types, in which case they are called advDataSource.

CHAPTER 6 THE resources SERVICE

The `rest_v2/resources` service searches the repository and access the resources it contains. This service provides performance and consistent handling of resource descriptors for all repository resource types. The service has two formats, one takes search parameters to find resources, the other takes a repository URI to access resource descriptors and file contents.

For further information, see:

- **Chapter 4, “Working With Resources,” on page 23** for general guidelines about using descriptors.
- **Chapter 5, “Resource Descriptors,” on page 31** for a reference to every type of resource and its attributes.
- **Chapter 7, “Working With File Resources,” on page 59** to download and upload file resources.
- **Chapter 8, “Working With Domains,” on page 65** to view Domains and their nested resources.

This chapter includes the following sections:

- **Searching the Repository**
- **Paginating Search Results**
- **Viewing Resource Details**
- **Creating a Resource**
- **Modifying a Resource**
- **Copying a Resource**
- **Moving a Resource**
- **Deleting Resources**

6.1 Searching the Repository

The resources service, when used without specifying any repository URI, is used to search the repository. The various parameters listed in the following table let you refine the search and specify how you receive search results. For example, the search and results pagination parameters can be used to implement an interface to repository resources in a REST client application.

Method	URL
GET	<code>http://<host>:<port>/jasperserver[-pro]/rest_v2/resources?<parameters></code>

Parameter	Type/Value	Description
q	String	Search for resources having the specified text in the name or description. Note that the search string does not match in the ID of resources.
folderUri	String	The path of the base folder for the search.
recursive	true false	Indicates whether search should include all sub-folders recursively. When omitted, the default behavior is recursive (true).
type	String	Match only resources of the given type. Valid types are listed in Chapter 5, “Resource Descriptors,” on page 31 , for example: dataType, jdbcDataSource, reportUnit, or file. Multiple type parameters are allowed. Wrong values are ignored.
accessType	viewed modified	Filters the results by access events: viewed (by current user) or modified (by current user). By default, no access event filter is applied.
dependsOn	/path/to/resource	Searches for all resources depending on specified resource. Only data source and reportUnit resources may be specified. If this parameter is specified, then all the other parameters except pagination are ignored.
showHiddenItems	true false	When set to true, results include nested local resources (in _files) as if they were in the repository. For more information, see 4.8, “Local Resources,” on page 28 . By default, hidden items are not shown (false).
sortBy	optional String	One of the following strings representing a field in the results to sort by: uri, label, description, type, creationDate, updateDate, accessTime, or popularity (based on access events). By default, results are sorted alphabetically by label.
limit offset forceFullPage forceTotalCount		<p>Pagination is enabled by default, and the default limit is 100 results. If you work with large repositories, you may not receive all results in a single request, and you must handle pagination issues. These parameters are described in 6.2, “Paginating Search Results,” on page 47.</p> <p>Note that the pagination limit is applied to raw search results before permissions are computed. After permissions, the result set may be less than 100. In extreme cases, the search may return only a few items, but the search is still incomplete. In that case use forceFullPage as described in 6.2.2, “Full Page Pagination,” on page 49, or set the limit to 0 as described in 6.2.3, “No Pagination,” on page 50.</p>
Options		
accept: application/json (default) accept: application/xml		

Return Value on Success	Typical Return Values on Failure
200 OK – The body contains a list of resourceLookup descriptors representing the results of the search.	404 Not Found – The specified folder is not found in the repository. 204 No Content – All type values specified are invalid.

The response of a search is a set of shortened descriptors showing only the common attributes of each resource. One additional attribute specifies the type of the resource. This allows the client to quickly receive a list of resources for display or further processing.

application/json	application/xml
<pre>[{ "uri" :"/sample/resource/uri", "label":"Sample Label", "description": "Sample Description", "type":"folder", "permissionMask":"0", "creationDate": "2013-07-04T12:18:47", "updateDate": "2013-07-04T12:18:47", "version":"0" }, ...]</pre>	<pre><resources> <resourceLookup> <uri>/sample/resource/uri</uri> <label>Sample Label</label> <description>Sample Description </description> <type>folder</type> <permissionMask>0</permissionMask> <creationDate>2013-07-04T12:18:47 </creationDate> <updateDate>2013-07-04T12:18:47 </updateDate> <version>0</version> </resourceLookup> ... </resources></pre>

6.2 Paginating Search Results

Paginating search results can speed up the user experience by making smaller queries and displaying the fewer results one page at a time. By default, a page is approximately 100 repository items. If and when users request another page, your application needs to send another request to the server with the same search parameters but an updated offset number that fetches the next page.



When any folder in your repository contains more than 100 subfolders and resources, then the search results will be paginated by default. This means you will not receive all results in a single request. In this case, you must use the pagination parameters to obtain more pages or change the pagination strategy as explained below.

Your application could perform further optimizations such as requesting a page and storing it before the user requests it. That way, the results can be displayed immediately, and each page can be fetched in the background while the user is looking at the previous page.

Pagination is complicated by the fact that JasperReports Server enforces permissions after performing the query based on your search parameters. This means that a default search can return fewer results than a full page, but this behavior can be configured.

There are 3 different combinations of settings that you can use for pagination.

- Default pagination - Every page may have less than a complete page of results, but this is the fastest strategy and the easiest to implement.
- Full page pagination - Ensures that every page has exactly the number of results that you specify, but this makes the server perform more queries, and it requires extra logic in the client.
- No pagination - Requests all search results in a single reply, which is simplest to process but can block the caller for a noticeable delay when there are many results.

The advantages and disadvantages of each pagination strategy are described in the following sections. Choose a strategy for your repository searches based on the types searches being performed, the user performing the search, and the contents of your repository. Every request to the resources service can use a different pagination strategy; it's up to your client app to use the appropriate strategy and process the results accordingly.

6.2.1 Default Pagination

With the default pagination, every page of results returned by the server may contain less than the designated page size. You can determine the number of actual results from the HTTP headers of the response. The headers also indicate whether there are further pages to fetch.

Default pagination has the best performance and, when configured with the right limit for the size of your repository, almost no delay in response for your users. Because results are filtered by permissions, the user credentials that you specify for the request determine how full each page is:

- The system admin (`superuser`) has access to every resource, and therefore the results are effectively unfiltered and each page is full. But the same can be true when you perform a search as `jasperadmin` within his organization, or even as a plain user within a folder where the user has full read permission. In these cases the default pagination is very efficient and has no partially-full pages.
- If you are performing a sparse search, for example finding all reports that a given user has permission to access within an entire and large organization, then the results may have many partially-full page, all of differing lengths. In this case, you may prefer to use [6.2.2, “Full Page Pagination,” on page 49](#).

Parameters of resources for Default Pagination		
Parameter	Type/Value	Description
limit	integer default is 100	This defines the page size, which is maximum number of resources to return in each response. However, with default pagination, the response is likely have less than this value of responses. The default limit is 100. You can set the limit higher or lower if you want to process generally larger or smaller pages, respectively.
offset	integer	By setting the offset to a whole multiple of the limit, you select a specific page of the results. The default offset is 0 (first page). With a limit of 100, subsequent calls should set offset=100 (second page), offset=200 (third page), etc.
forceFullPage	false (default)	The default is false, so you do not need to specify this parameter.
forceTotal Count	true false	When true, the Total-Count header is set in every paginated response, which impacts performance. When false, the default, the header is set in the first page only. Note that Total-Count is the intermediate, unfiltered count of results, not the number of results returned by this service.

With each response, you can process the HTTP headers to help you display pagination controls:

Headers in Responses for Default Pagination	
Header	Description
Result-Count	This is the number of results that are contained in the current response. It can be less than or equal to the limit.
Start-Index	The Start-Index in the response is equal to the offset specified in the request. With a limit=100, it will be 0 on the first page, 100 on the second page, etc.
Next-Offset	This is the offset to request the next page. With forceFullPage=false, the Next-Offset is equivalent to Start-Index+limit, except on the last page. On the last page, the Next-Offset is omitted to indicate there are no further pages.
Total-Count	<p>This is the total number of results before permissions are applied. This is not the total number of results for this search by this user, but it is an upper bound. Dividing this number by the limit gives the number of pages that will be required, though not every page will have the full number of results.</p> <p>As described in the previous table, this header only appears on the first response, unless forceTotalCount=true.</p>

6.2.2 Full Page Pagination

Full Page pagination ensures that every page, except the last one, has the same number of results, the number given by the limit parameter. To do this, JasperReports Server performs extra queries after filtering results for permission, until each page has the full number of results. Though small, the extra queries have a performance impact and may slow down the request. In addition, your client must read the HTTP header in every response to determine the offset value for the next page.

For full page pagination, set the pagination parameters of the resources service as follows:

Parameters of resources for Full Page Pagination		
Parameter	Type/Value	Description
limit	integer default is 100	Specifies the exact number of resources to return in each response. This is equivalent to the number of results per page. The default limit is 100. You can set the limit higher or lower if you want to process larger or smaller pages, respectively.
offset	integer	Specifies the overall offset to use for retrieving the next page of results. The default offset is 0 (first page). For subsequent pages, you must specify the value given by the Next-Offset header, as described in the next table.
forceFullPage	true	Setting this parameter to true enables full page pagination. Depending on the type of search and user permissions, this parameter can cause significant performance delays.

Parameters of resources for Full Page Pagination		
Parameter	Type/Value	Description
forceTotal Count	do not use	When forceFullPage is true, the Total-Count header is set in every response, even if this parameter is false by default.

With each response, you must process the HTTP headers as follows:

Headers in Responses for Full Page Pagination	
Header	Description
Result-Count	This is the number of results that are contained in the current response. With full page pagination, it is equal to the limit in every response except for the last page.
Start-Index	The Start-Index in the response is equal to the offset specified in the request. It changes with every request-response.
Next-Offset	The server calculates this value based on the extra queries it performed to fill the page with permission-filtered results. In order to avoid duplicate results or skipped results, your client must read this number and submit it as the offset in the request for the next page. When this value is omitted from the header, it indicates there are no further pages.
Total-Count	This is the total number of results before permissions are applied. This is not the total number of results for this search by this user, but it is an upper bound.

6.2.3 No Pagination

In certain cases, you can turn off pagination. Use this for small search request that you want to process as a whole, for example a listing of all reports in a folder. In this case, you receive and process all results in a single response and do not need to implement the logic for pagination. You should only use this for result sets that are known to be small.

To turn off pagination, set the pagination parameters of the resources service as follows:

Parameters of resources for No Pagination		
Parameter	Type/Value	Description
limit	0	To return all results without pagination, set limit=0. Do not set limit=0 for large searches, for example from the root of the repository, because it can cause significant delays and return a very large number of results.
offset	do not use	The default offset is 0, which is the start of the single page of results.
forceFullPage	do not use	This setting has no meaning when there is no limit.

Parameters of resources for No Pagination		
Parameter	Type/Value	Description
forceTotal Count	do not use	The Total-Count header is included in the first (and only) response. Note that Total-Count is the intermediate, unfiltered count of results, not the number of results returned by this service.

With each response, you must process the HTTP headers as follows:

Headers in Responses for No Pagination	
Header	Description
Result-Count	This is the number of results contained in the current response. Thus, this header indicates how many results you should process in the single response.
Start-Index	This is 0 for a single response containing all the search results.
Next-Offset	This header is omitted because there is no next page.
Total-Count	This is the total number of results before permissions are applied. It is of little use.

6.3 Viewing Resource Details

Use the GET method and a resource URI to request the resource's complete descriptor.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/ rest_v2/resources /path/to/resource	
Parameter	Type/Value	Description
expanded	true false	When true, all nested resources will be given as full descriptors. The default behavior, false, has all nested resources given as references. For more information, see 4.8, “Local Resources,” on page 28 .
Options		
accept: application/json (default) accept: application/xml accept: application/repository.folder+<format> (specifically to view the folder resource)		

Return Value on Success	Typical Return Values on Failure
200 OK – The response will indicate the content-type and contain the corresponding descriptor, for example: application/repository.dataType+json	404 Not Found – The specified resource is not found in the repository.

6.4 Creating a Resource

The POST and PUT methods offer alternative ways to create resources. Both take a resource descriptor but each handles the URL differently.

With the POST method, specify a folder in the URL, and the new resource ID is created automatically from the label attribute in its descriptor.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/folder	
Parameter	Type/Value	Description
create Folders	true false	By default, this is true, and the service will create all parent folders if they don't already exist. When set to false, the folders specified in the URL must all exist, otherwise the service returns an error.
Content-Type		Content
application/repository.<resourceType>+json application/repository.<resourceType>+xml		A well defined descriptor of the specified type and format. See Chapter 5, “Resource Descriptors,” on page 31
Return Value on Success		Typical Return Values on Failure
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just created.		400 Bad Request – Mismatch between the content-type and the fields or syntax of the actual descriptor.

With the PUT method, specify a unique new resource ID as part of the URL. For more information, see [4.1, “Resource URI,”](#) on page 23.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/resource? ?<parameters>

Parameters	Type/Value	Description
create Folders	true false	True by default, and the service will create all parent folders if they don't already exist. When set to false, the folders specified in the URL must all exist, otherwise the service returns an error.
overwrite	true false	When true, the resource given in the URL is overwritten even if it is a different type than the resource descriptor in the content. The default is false.
Content-Type		Content
application/repository.<resourceType>+json application/repository.<resourceType>+xml		A well defined descriptor of the specified type and format. See Chapter 5, “Resource Descriptors,” on page 31
Return Value on Success		Typical Return Values on Failure
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just created.		400 Bad Request – Mismatch between the content-type and the fields or syntax of the actual descriptor.

The POST method also supports a way to create complex resources and their nested resources in a single multipart request.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/folder
Content-Type	Content
multipart/form-data	<p>Root resource multipart item name: resource</p> <p>Root resource multipart Content-type and corresponding item names:</p> <ul style="list-style-type: none"> mondrianConnection <ul style="list-style-type: none"> schema – Mondrian schema XML file secureMondrianConnection <ul style="list-style-type: none"> schema – Mondrian schema XML file accessGrantSchemas.accessGrantSchema[{itemIndex}] – XML file semanticLayerDataSource <ul style="list-style-type: none"> schema – Domain schema XML file securityFile – XML security file bundles.bundle[{bundleIndex}] – Properties file for internationalization reportUnit <ul style="list-style-type: none"> jrxml – Report unit JRXML file files.{fileName} – Report unit attached resource file (e.g. images)

Return Value on Success	Typical Return Values on Failure
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just created.	400 Bad Request – Mismatch between the content-type and the fields or syntax of the actual descriptor.

6.5 Modifying a Resource

Use the PUT method above to overwrite an entire resource. Specify the path of the target resource in the URL and specify resource of the same type. Use `overwrite=true` to replace a resource of a different type.

The resource descriptor must completely describe the updated resource, not use individual fields. The descriptor must also use only references for nested resources, not other resources expanded inline. You can update the local resources using the hidden folder `_file`. For more information, see [4.8, “Local Resources,” on page 28](#).

As of JasperReports Server 5.5, the PATCH method updates individual descriptor fields on the target resource. It also accepts expressions that modify the descriptor in the Spring Expression Language. This expression language lets you easily modify the structure and values of descriptors.

Method	URL
PATCH	<code>http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/resource</code>
Content-Type	Content
application/json	<pre>{ "version" : 0, "patch":[{ "field":"label", "value":"New Label" }, { "field":"query", "value":"/path/to/query/resource" }, { "expression":"inputControls.add(new com.jaspersoft. jasperserver.dto.resources.ClientReference(). setUri('/datatypes/decorate'))" }] }</pre>

application/xml	<pre> <patchItems> <version>0</version> <patch> <field>label</field> <value>New Label</value> </patch> <patch> <field>query</field> <value>/path/to/query/resource</value> </patch> <patch> <expression>inputControls.add(new com.jaspersoft. jasperserver.dto.resources.ClientReference(). setUri('/datatypes/decorate')) </expression> </patch> </patchItems> </pre>	
Return Value on Success		Typical Return Values on Failure
For confirmation, the response contains the full descriptor of the resource that was just modified.		400 Bad Request – Mismatch between the patch fields and the fields or syntax of the actual descriptor. 409 Conflict – Old version number; the resource was updated more recently than the last version number received.

The patch descriptor contains attributes of the target resource. It can't be used to specify the attributes of nested resources. However, you can apply the patch operation directly to the local resource in the hidden `_files` folder.

If your client does not support the PATCH method, use the POST method and specify the following HTTP header:

X-HTTP-Method-Override: PATCH

6.6 Copying a Resource

Copying a resource uses the Content-Location HTTP header to specify the source of the copy operation. If any resource descriptor is sent in the request, it is ignored.

Method	URL	
POST	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/folder?<parameters>">http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/folder?<parameters>	
Parameters	Type/Value	Description
create Folders	true false	True by default, and the service will create all parent folders if they don't already exist. When set to false, the folders specified in the URL must all exist, otherwise the service returns an error.

overwrite	true false	When true, the target resource given in the URL is overwritten even if it is a different type than the resource descriptor in the content. The default is false.
Options		
Content-Location: {resourceSourceUri} - Specifies the resource to be copied.		
Return Value on Success		Typical Return Values on Failure
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just copied.		404 Not Found – When the {resourceSourceUri} is not valid.

6.7 Moving a Resource

Moving a resource uses the PUT method, whereas copying it uses the POST method.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/folder?<parameters>	
Parameters	Type/Value	Description
create Folders	true false	True by default, and the service will create all parent folders if they don't already exist. When set to false, the folders specified in the URL must all exist, otherwise the service returns an error.
overwrite	true false	When true, the target resource given in the URL is overwritten even if it is a different type than the resource descriptor in the content. The default is false.
Options		
Content-Location: {resourceSourceUri} - Specifies the resource to be moved.		
Return Value on Success		Typical Return Values on Failure
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just moved.		404 Not Found – When the {resourceSourceUri} is not valid.

6.8 Deleting Resources

The DELETE method has two forms, one for single resources and one for multiple resources.

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/resource
Return Value on Success	Typical Return Values on Failure
204 No Content – The request was successful and there is no descriptor to return.	404 Not Found – When the resource path or ID is not valid.

To delete multiple resources at once, specify multiple URIs with the resourceUri parameter.

Method	URL	
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources?resourceUri={uri}&...	
Parameter	Type/Value	Description
resourceUri	string	Specifies a resource to delete. Repeat this parameter to delete multiple resources.
Return Value on Success	Typical Return Values on Failure	
204 No Content – The request was successful and there is no descriptor to return.	404 Not Found – When the resourceUri is not valid.	

CHAPTER 7 WORKING WITH FILE RESOURCES

This chapter includes the following sections:

- **MIME Types**
- **Downloading File Resources**
- **Uploading File Resources**
- **Updating File Resources**

7.1 MIME Types

When downloading or uploading file contents, you must specify the MIME type (Multi-Purpose Internet Mail Extensions) that corresponds with the desired file type, as shown in the following table.

You can customize this list of MIME types in the server by editing the `contentTypeMapping` map in the file `.../WEB-INF/applicationContext-rest-services.xml`. You can change MIME types for predefined types, add MIME types, or add custom types.

Table 7-1 MIME Types for File Contents

File Types	Corresponding MIME Types
pdf	application/pdf
html	text/html
xls	application/xls
rtf	application/rtf
csv	text/csv
ods	application/vnd.oasis.opendocument.spreadsheet
odt	application/vnd.oasis.opendocument.text
txt	text/plain

File Types	Corresponding MIME Types
docx	application/vnd.openxmlformats-officedocument.wordprocessingml.document
xlsx	application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
font	font/*
img	image/*
jrxml	application/jrxml
jar	application/zip
prop	application/properties
jrtx	application/jrtx
xml	application/xml
css	text/css
accessGrantSchema	application/accessGrantSchema
olapMondrianSchema	application/olapMondrianSchema

7.2 Downloading File Resources

There are two read operations on file resources:

- Viewing the file resource details to determine the file format
- Downloading the binary file contents

To view the file resource details, specify the URL and the file descriptor type as follows:

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/ rest_v2/resources /path/to/file/resource
Options	
accept: application/repository.file+json accept: application/repository.file+xml	
Return Value on Success	Typical Return Values on Failure
200 OK – The response will contain the file resource descriptor.	404 Not Found – The specified resource is not found in the repository.

The type attribute of the file resource descriptor indicates the format of the contents. However, you can also download the binary file contents directly, with the format indicated by the MIME content-type of the response:

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/file/resource
Return Value on Success	Typical Return Values on Failure
200 OK – The response content-type will indicate the MIME type of the binary contents. See Table 7-1, “MIME Types for File Contents,” on page 59 for the list of MIME types that correspond to file resource types.	404 Not Found – The specified resource is not found in the repository.

7.3 Uploading File Resources

There are several ways of uploading file contents to create file resources. The simplest way is to POST a file descriptor containing the file in base64 encoding.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/folder	
Parameters	Type/Value	Description
create Folders	true false	True by default, and the service will create all parent folders if they don't already exist. When set to false, the folders specified in the URL must all exist, otherwise the service returns an error.
Content-Type	Content	
application/repository.file+json application/repository.file+xml	A well defined file resource descriptor, as described in 5.13, “File,” on page 38. The contents of the file are base64-encoded in the <code>content</code> attribute of the descriptor.	
Return Value on Success	Typical Return Values on Failure	
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just created.	400 Bad Request – Mismatch between the content-type and the fields or syntax of the actual descriptor.	

You can also create a file resource with a multipart form request. The request parameters contain information that becomes the name and description of the new file resource.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/folder	
Content-Type		Content
multipart/form-data		<p>The request should include the following parameters:</p> <ul style="list-style-type: none"> label – contains the name of the file resource description – contains a description for the resource type – contains a file type shown in Table 7-1, “MIME Types for File Contents,” on page 59 data – contains the file contents
Return Value on Success		Typical Return Values on Failure
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just created.		400 Bad Request – Mismatch between the content-type and the fields or syntax of the actual descriptor.

Another form allows you to create a file resource by direct streaming, without needing to create it first as a descriptor object. In this case, the required fields of the file descriptor are specified in HTTP headers.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/folder	
Options		
Content-Description: <file-description> – Becomes the description field of the created file resource Content-Disposition: attachment; filename=<filename> – Becomes the name of the file resource		
Content-Type	Content	
{MIME type}	The MIME type from Table 7-1, “MIME Types for File Contents,” on page 59 that corresponds to the desired file type. The body of the request then contains the binary data representation of that file format.	
Return Value on Success		Typical Return Values on Failure
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just created.		400 Bad Request – Mismatch between the content-type and the fields or syntax of the actual descriptor.

7.4 Updating File Resources

For an existing file resource, you can update its name, description or file contents in several ways.

The simplest way is to PUT a file descriptor containing the new file in base64 encoding. This new definition of the file resource overwrites the previous one.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/resource	
Content-Type		Content
application/repository.file+json application/repository.file+xml		A well defined file resource descriptor, as described in 5.13, “File,” on page 38 . The new contents of the file are base64-encoded in the <code>content</code> attribute of the descriptor.
Return Value on Success		Typical Return Values on Failure
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just created.		400 Bad Request – Mismatch between the content-type and the fields or syntax of the actual descriptor.

The second method allows you to update a file resource by direct streaming. You can specify the Content-Description and Content-Disposition headers to update the resource description or name, respectively.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/resource	
Options		
Content-Description: <file-description> – Becomes the description field of the created file resource		
Content-Disposition: attachment; filename=<filename> – Becomes the name of the file resource		
Content-Type	Content	
{MIME type}	The MIME type from Table 7-1, “MIME Types for File Contents,” on page 59 that corresponds to the desired file type. The body of the request then contains the binary data representation of that file format.	
Return Value on Success		Typical Return Values on Failure
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just created.		400 Bad Request – Mismatch between the content-type and the fields or syntax of the actual descriptor.

CHAPTER 8 WORKING WITH DOMAINS



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.

This chapter explains the limited interaction with Domains that is available through the REST API. The metadata service retrieves the display layer of a Domain containing sets and items and their labels. You can also retrieve the full Domain schema and security files through the resources service, but the API provides no functionality to parse these.

This chapter includes the following sections:

- [The metadata Service](#)
- [Fetching a Domain Schema](#)
- [Fetching Domain Bundles and Security Files](#)

8.1 The metadata Service

The `rest_v2/domains/metadata` service gives access to the sets and items exposed by a Domain for use in Ad Hoc reports. Items are database fields exposed by the Domain, after all joins, filters, and calculated fields have been applied to the database tables selected in the Domain. Sets are groups of items, arranged by the Domain creator for use by report creators.



A limitation of the metadata service only allows it to operate on Domains with a single data island. A data island is a group of fields that are all related by joins between the database tables in the Domain. Fields that belong to tables that are not joined in the Domain belong to separate data islands.

If your Domain contains localization bundles you can specify a locale and an optional alternate locale and preference (called q-value, a decimal between 0 and 1).

Method	URL
GET	<code>http://<host>:<port>/jasperserver[-pro]/rest_v2/domains/path/to/Domain/metadata</code>

Options	
Accept-Language: <locale>[, <alt-locale>;q=0.8] Accept: application/xml (default) Accept: application/json	
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 Domain URI is not found in the repository. This service also returns an XML <code>errorDescriptor</code> giving a human-readable error message.

The response of the metadata service is an XML or JSON structure that describes the sets and items available in the selected Domain. This metadata includes the localized labels for the sets and items, as well as the datatypes of the items. The `resourceId` of the sets and items are internal to the Domain and not meaningful or otherwise useable.

For more information about Domains, refer to the *JasperReports Server User Guide*.

The following example shows the JSON response for a Domain with:

- A set named expense containing:
 - An item named Exp Date of type Date
 - An item named Amount of type BigDecimal
- A set named store containing:
 - An item named Store Type of type String
 - ...

```
{
  "rootLevel": {
    "id": "root",
    "subLevels": [
      {
        "id": "expense_join",
        "label": "expense",
        "properties": {
          "resourceId": "expense_join"
        }
      },
      {
        "id": "ej_expense_fact_exp_date",
        "label": "Exp Date",
        "properties": {
          "JavaType": "java.sql.Date",
          "resourceId": "expense_join.e.exp_date"
        }
      },
      {
        "id": "ej_expense_fact_amount",
        "label": "Amount",
        "properties": {

```

```

        "JavaType": "java.math.BigDecimal",
        "resourceId": "expense_join.e.amount"
    }
}
],
{
    "id": "expense_join_store",
    "label": "store",
    "properties": {
        "resourceId": "expense_join"
    },
    "items": [
        {
            "id": "ej_store_store_type",
            "label": "Store Type",
            "properties": {
                "JavaType": "java.lang.String",
                "resourceId": "expense_join.s.store_type"
            }
        },
        ...
    ]
}
]
}
}
}

```

The following example shows the same Domain as returned by the metadata service in XML format:

```

<?xml version="1.0" encoding="UTF-8"?>
<domainMetadata>
  <rootLevel>
    <id>root</id>
    <subLevels>
      <subLevel>
        <id>expense_join</id>
        <label>expense</label>
        <properties>
          <entry>
            <key>resourceId</key>
            <value>expense_join</value>
          </entry>
        </properties>
        <items>
          <item>
            <id>ej_expense_fact_exp_date</id>
            <label>Exp Date</label>
            <properties>
              <entry>
                <key>JavaType</key>
                <value>java.sql.Date</value>
              </entry>
              <entry>

```

```
        <key>resourceId</key>
        <value>expense_join.e.exp_date</value>
    </entry>
</properties>
</item>
<item>
    <id>ej_expense_fact_amount</id>
    <label>Amount</label>
    <properties>
        <entry>
            <key>JavaType</key>
            <value>java.math.BigDecimal</value>
        </entry>
        <entry>
            <key>resourceId</key>
            <value>expense_join.e.amount</value>
        </entry>
    </properties>
</item>
</items>
</subLevel>
<subLevel>
    <id>expense_join_store</id>
    <label>store</label>
    <properties>
        <entry>
            <key>resourceId</key>
            <value>expense_join</value>
        </entry>
    </properties>
    <items>
        <item>
            <id>ej_store_store_type</id>
            <label>Store Type</label>
            <properties>
                <entry>
                    <key>JavaType</key>
                    <value>java.lang.String</value>
                </entry>
                <entry>
                    <key>resourceId</key>
                    <value>expense_join.s.store_type</value>
                </entry>
            </properties>
        </item>
        ...
    </items>
</subLevel>
</subLevels>
</rootLevel>
</domainMetadata>
```

8.2 Fetching a Domain Schema

The metadata service returns only the display information about a Domain, not its internal definition. The fields, joins, filters, and calculated fields that define the internal structure of a Domain make up the Domain design. The XML representation of a Domain design is called the Domain schema.

Currently, there is no REST service to interact with Domain schemas, but you can use the resources service to retrieve the raw schema. First, retrieve the resource descriptor for the Domain. For example, to view the descriptor for the Supermart Domain, use the following request (when logged in as `jasperadmin`):

GET `http://<host>:<port>/jasperserver-pro/rest_v2/resources/Domains/supermartDomain`

This descriptor contains the Domain schema as an internal resource:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<semanticLayerDataSource>
  <creationDate>2013-10-10T15:30:31</creationDate>
  <description>Comprehensive example of Domain (pre-joined table sets for complex reporting, custom
query based dataset, column and row security, I18n bundles)</description>
  <label>Supermart Domain</label>
  <permissionMask>1</permissionMask>
  <updateDate>2013-10-10T15:30:31</updateDate>
  <uri>/organizations/organization_1/Domains/supermartDomain</uri>
  <version>1</version>
  <dataSourceReference>
    <uri>/organizations/organization_1/analysis/datasources/FoodmartDataSourceJNDI</uri>
  </dataSourceReference>
  <bundles>
    <bundle>
      <fileReference><uri>/organizations/organization_1/Domains/supermartDomain_files/supermart_
domain.properties</uri></fileReference>
      <locale></locale>
    </bundle>
    <bundle>
      <fileReference><uri>/organizations/organization_1/Domains/supermartDomain_files/supermart_
domain_en_US.properties</uri></fileReference>
      <locale>en_US</locale>
    </bundle>
    <bundle>
      <fileReference><uri>/organizations/organization_1/Domains/supermartDomain_files/supermart_
domain_de.properties</uri></fileReference>
      <locale>de</locale>
    </bundle>
    <bundle>
      <fileReference><uri>/organizations/organization_1/Domains/supermartDomain_files/supermart_
domain_fr.properties</uri></fileReference>
      <locale>fr</locale>
    </bundle>
    <bundle>
      <fileReference><uri>/organizations/organization_1/Domains/supermartDomain_files/supermart_
domain_es.properties</uri></fileReference>
      <locale>es</locale>
    </bundle>
    <bundle>
      <fileReference><uri>/organizations/organization_1/Domains/supermartDomain_files/supermart_
domain_ja.properties</uri></fileReference>
      <locale>ja</locale>
    </bundle>
  </bundles>
</semanticLayerDataSource>
```

```
</bundle>
<bundle>
  <fileReference><uri>/organizations/organization_1/Domains/supermartDomain_files/supermart_
domain_zh_CN.properties</uri></fileReference>
  <locale>zh_CN</locale>
</bundle>
</bundles>
<schemaFileReference>
  <uri>/organizations/organization_1/Domains/supermartDomain_files/supermartDomain_schema</uri>
</schemaFileReference>
<securityFileReference>
  <uri>/organizations/organization_1/Domains/supermartDomain_files/supermartDomain_domain_secur-
ity</uri>
</securityFileReference>
</semanticLayerDataSource>
```

Use the following request to access the Domain schema file inside the Domain resource:

```
GET http://<host>:<port>/jasperserver-pro/rest_v2/resources/Domains/supermartDomain_
files/supermartDomain_schema
```

The Domain schema is an XML file with a structure explained in the *JasperReports Server User Guide*. If you wish to modify the schema programmatically, you must write your own parser to access its fields and definitions. You can then replace the schema file in the Domain with one of the file updating methods described in [7.3, “Uploading File Resources,” on page 61](#).

8.3 Fetching Domain Bundles and Security Files

Once you have the descriptor of a Domain resource as shown in the previous section, you can access the other files that help define a Domain. For example, you can access the language bundles of the Supermart Domain with the following request:

```
GET http://<host>:<port>/jasperserver-pro/rest_v2/resources/Domains/supermartDomain_files/supermart_
domain_<locale>.properties
```

Language bundles are Java properties files that follow the language bundle naming convention, and that contain the names of the sets and fields in the language of the locale in the filename.

You can also retrieve the localized set and item names by specifying Accept-Language when using the metadata service. However, by accessing the language bundles through the Domain descriptor, you read the default bundle to see the pattern of keys and values, and then create a bundle for a new locale.

Domains may also contain a security file that is also stored as an internal resource of the Domain descriptor. Use the following example to request the security file of the Supermart Domain in the sample data:

```
GET http://<host>:<port>/jasperserver-pro/rest_v2/resources/Domains/supermartDomain_files/supermart_
domain_security
```

A security file defines a complex set of access permissions to the data in the rows and columns returned by the Domain, based on the username, roles, or profile attributes of the user running a Domain-based report. As with the Domain schema file, you must write your own parser to interpret this file and modify it.

You can then upload an updated language bundle or security file for the Domain with one of the methods described in [7.3, “Uploading File Resources,” on page 61](#).

For more information about language bundles and security files in Domains, see the *JasperReports Server User Guide*.

CHAPTER 9 THE permissions SERVICE

The `rest_v2/permissions` service reads and sets permissions on resources in the repository.

This chapter includes the following sections:

- **Permission Constants**
- **Viewing Multiple Permissions**
- **Viewing a Single Permission**
- **Setting Multiple Permissions**
- **Setting a Single Permission**
- **Deleting Multiple Permissions**
- **Deleting a Single Permission**

9.1 Permission Constants

In the permissions service, the syntax allows you to 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-write: 6
- 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_ADMINISTRATOR (this is a root role and thus has no organization specified)
user:/organization_1/joeuser
```

Recipients are listed when viewing permissions, and they are also used to set a permission. A recipient can be specified in a URL parameter when allowed, but in this case, the slash (/) character 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.



There is one permission that is not defined: you cannot read or write the permission for ROLE_SUPERUSER on the root .

9.2 Viewing Multiple Permissions

The GET method of the 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 organization as %2F<orgID>%2F<recipientID> (%2F is the / character).
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>
```

9.3 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	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 object ID, and the organization ID if necessary. The slash character 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, or when the recipient does not have any assigned permission (only inherited).

9.4 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	A JSON object that describes a set of permissions, for example: <pre>{ "permission" : [{ "uri": "/properties", "recipient": "role:/ROLE_USER", "mask": "1" }, { "uri": "/properties", "recipient": "role:/ROLE_ADMINISTRATOR", "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 exiting 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 <code>uri</code> 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.	<p>400 Bad Request – If a recipient or mask is invalid.</p> <p>404 Not Found – If the resource in the URL is invalid.</p>

9.5 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	<p>A JSON object that describes a single permission on a single resource, for example:</p> <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.

9.6 Deleting Multiple Permissions

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.

9.7 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[-proj]/ 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.

CHAPTER 10 THE export SERVICE

The `rest_v2/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.

This chapter includes the following sections:

- **Requesting an Export**
- **Polling the Export Status**
- **Fetching the Export Output**
- **Canceling an Export Operation**

10.1 Requesting an Export

Use the following method to specify the export options for your export request:

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 running export operation.			401 Unauthorized – Export is available only to the system admin user (<code>superuser</code>).

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

```
{
  "roles": ["ROLE_USER", "ROLE_MANAGER|organization_1"],
  "users": ["superuser", "joeuser|organization_1"],
  "uris": ["/public/Samples/Reports/AllAccounts",
           "/organizations/organization_1/reports/Survey/Survey_Data"]
  "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 options you can list in the request.

Export Options	Description
roles	A list of role names to export. Specify the role-users parameter to also export all users who have these roles.
users	A list of user names to export.
uris	A list of resources or folders to export, specified as repository URIs. When a folder is specified, all its contents and all its subfolders recursively are included. To export all resources in the repository or in an organization, specify "/" (root) in this list. When you specify an organization ID below, the URIs in this list are all relative to the organization.
scheduledJobs	A list of report URIs for which all scheduled jobs are exported. If you specify a folder URIs, the scheduled jobs for all reports in the folder, recursively, are exported.
resourceTypes	A list of resource types that filters any selected resources for export. When omitted, all resources specified by URI or folder URI are exported. When specified, only the resource types in this list are exported.
organization	A single organization ID that determines a branch of the repository for export. When this option is specified, this organization becomes the root for all roles, users, and URIS to be listed for export.
parameters	A list of parameters that act as flags: if specified, the corresponding action is taken, if omitted they have no effect. The export parameters are listed in the following table.

The following table describes the export parameters that can be specified in the parameters option:

Export Parameters	Description
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 that role. This option should only be used if roles are specified.

Export Parameters	Description
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.
skip-dependent-resources	When specified, only the resources specified by URIs are exported, no dependent resources such as data sources, queries, or files included by reference are exported. This will create broken dependencies during import unless the same dependencies already exist in the same relative locations in the destination.
skip-suborganizations	When specified, the export will omit all the items such as roles, users, and resources that belong to suborganizations, even if they are directly specified using the corresponding options. When no organization ID is specified, this flag applies to the root such that no top-level organizations are included in the export, only the contents of the root.
include-attributes	Includes all attributes that are associated with a item being exported, such as a user, an organization, or the root.
skip-attribute-values	When specified with include-attributes, only attribute names are exported with null values. Use this to prevent applying attributes that are specific to one server or one organization.
include-server-settings	When specified, the configuration and security settings on the server are exported. When imported into another server, these settings will take effect immediately.
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",
  "phase": "inprogress",
  "message": "Progress..."
}
```

The response may also warn you of any broken dependencies in the export that may affect a future import operation:

```
{
  "id": "njkhfs8374",
  "phase": "inprogress",
  "message": "Progress..."
  "warnings": [
    {
      "code": "export.broken.dependency",
      "message": "Resource with broken dependencies",
      "parameters": [
        "path_to_broken_resource"
      ], ...
    }, ...
  ]
}
```

10.2 Polling the Export Status

After receiving the export ID in the response to the export request, you can check the state of the export operation. The server takes up to several seconds to generate the export catalog, depending on the size of the requested resources and the load on the server.

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"
}
```

10.3 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[-pro]/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.

10.4 Canceling an Export Operation

To cancel any export operation that you have started, send a DELETE request with the ID of the export operation.

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/export/<export-id>
Return Value on Success	Typical Return Values on Failure
204 No Content – The specified export operation was canceled.	404 Not Found – When the specified export ID is not found.

CHAPTER 11 THE import SERVICE

Use the `rest_v2/import` service to upload a catalog as a zip file and import it into the server 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`) to import into root, but organization admins may import into their organizations or suborganizations.

Jaspersoft does not recommend uploading files greater than 2 gigabytes.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/import?<arguments>	
Argument	Value	Description
update?	true	Resources in the catalog replace those in the repository if their URIs and types match.
skipUserUpdate?	true	When used with <code>update=true</code> , users in the catalog are not imported or updated. Use this option to import catalogs without overwriting currently defined users.
broken Dependencies?	skip include fail	Defines the strategy when importing a resource with broken dependencies. The default value is fail. <ul style="list-style-type: none">• skip – The resource with broken dependency won't be imported, but the import operation will continue.• include – Attempts to import the resource by resolving dependencies with local resources. If unsuccessful, this resource is skipped.• fail – The import operation will stop and return an error.
organization?	orgID	Destination organization for importing. The file being imported must have been exported from an organization, not the root of the server. If this argument is not specified, the organization of the user performing the operation is used.

merge Organization?	true	When importing from one organization into a different organization, specify this argument. The resulting organization takes its ID from the import file. If organization IDs of import and destination do not match, and this argument is not specified, the operation stops with an error.
skipThemes?	true	When this argument is specified, any themes in the import other than the default theme is ignored. Use this argument when importing catalogs from servers before release 5.5 whose themes are incompatible.
includeAccess Events?	true	Restores the date, time, and user name of last modification if they are included in the catalog to import.
includeAudit Events?	true	Imports audit events if they are included in the catalog.
includeMonitoring Events?	true	Imports audit events if they are included in the catalog.
includeServer 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 contains the ID of the import operation needed to check its status:

```
<state>
  <id>2cc871ee-4645-4be0-b5b4-7d7c45c561cb</id>
  <message>Import in progress.</message>
  <phase>inprogress</phase>
</state>
```

To check the status of the import, use its ID in the following method:

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

CHAPTER 12 THE reports SERVICE

The `rest_v2/reports` service has a simple API for obtaining report output, such as PDF and XLS. The service also provides functionality to interact with running reports, report options, and input controls.

This chapter includes the following sections:

- [Running a Report](#)
- [Finding Running Reports](#)
- [Stopping a Running Report](#)

12.1 Running a Report

The reports service allows clients to receive report output in a single request-response. The reports service is a synchronous request, meaning the caller will be blocked until the report is generated and returned in the response. For large datasets or long reports, the delay can be significant. If you want to use a non-blocking (asynchronous) request, see [Chapter 13, “The reportExecutions Service,” on page 91](#).

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>	
Argument	Type/Value	Description
<format>	output type	One of the following formats: pdf, html, xls, xlsx, rtf, csv, xml, docx, odt, ods, jrprint. As of JasperReports Server 6.0, it is also possible to specify json if your reports are designed for data export. For more information, see the JasperReports Library samples documentation.
page?	Integer > 0	An integer value used to export a specific page.

ignore pagination?	Boolean	When set to true, the report will be generated as a single page. This can be useful for some formats such as csv. When omitted, this argument's default value is false and the report is paginated normally.
<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.
onePage PerSheet?	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.
baseUrl	String	Specifies the base URL that the report will use to load static resources such as JavaScript files. You can also set the deploy.base.url property in the .../WEB-INF/js.config.properties file to set this value permanently. If both are set, the baseUrl parameter in this request takes precedence.
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.

12.2 Finding Running Reports

The 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 reports service is deprecated. See [Chapter 13, “The reportExecutions Service,”](#) on page 91.

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>. In the commercial editions, %7C<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.

12.3 Stopping a Running Report

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



This syntax of the reports service is deprecated. See [Chapter 13, “The reportExecutions Service,”](#) on page 91.

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.

CHAPTER 13 THE reportExecutions SERVICE

As described in [Chapter 12, “The reports Service,” on page 87](#), synchronous report execution blocks the client waiting 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 `rest_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 (also called polling). 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 `reportExecutions` allows you to access jobs that are triggered by the scheduler. Finally, the `reportExecutions` service allows the client to stop any report execution or job that has been triggered.

This chapter includes the following sections:

- [Running a Report Asynchronously](#)
- [Polling Report Execution](#)
- [Requesting Report Execution Details](#)
- [Requesting Report Output](#)
- [Exporting a Report Asynchronously](#)
- [Modifying Report Parameters](#)
- [Polling Export Execution](#)
- [Finding Running Reports and Jobs](#)
- [Stopping Running Reports and Jobs](#)

13.1 Running a Report Asynchronously

In order to run a report asynchronously, the `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 13-1 Report Execution Properties

Property	Required or Default	Description
reportUnitUri	Required	Repository path (URI) of the report to run. For commercial editions with organizations, the URI is relative to the logged-in user's organization.

Property	Required or Default	Description
outputFormat	Required	Specifies the desired output format: pdf, html, xls, xlsx, rtf, csv, xml, docx, odt, ods, jrprint. As of JasperReports Server 6.0, it is also possible to specify json if your reports are designed for data export. For more information, see the JasperReports Library samples documentation.
freshData	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.
saveDataSnapshot	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.
interactive	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).
allowInlineScripts	true	Affects HTML export only. If true, then inline scripts are allowed, otherwise no inline script is included in the HTML output.
ignorePagination	Optional	When set to true, the report is generated as a single long page. This can be used with HTML output to avoid pagination. When omitted, the ignorePagination property on the JRXML, if any, is used.
pages	Optional	Specify a page range to generate a partial report. The format is: <startPageNumber>-<endPageNumber>
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.

Property	Required or Default	Description
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 <code>net.sf.jasperreports.engine.export.GenericElementTransformer</code> . These transformers are pluggable as JasperReports Library extensions.
attachmentsPrefix	attachments	For HTML output, this property specifies the URL path to use for downloading the attachment files (JavaScript and images). The full path of the default value is: <code>{contextPath}/rest_v2/reportExecutions/{reportExecutionId}/exports/{exportExecutionId}/attachments/</code> You can specify a different URL path using the placeholders <code>{contextPath}</code> , <code>{reportExecutionId}</code> , and <code>{exportExecutionId}</code> .
baseUrl	String	Specifies the base URL that the report will use to load static resources such as JavaScript files. You can also set the <code>deploy.-base.url</code> property in the <code>.../WEB-INF/js.config.properties</code> file to set this value permanently. If both are set, the <code>baseUrl</code> parameter in this request takes precedence.
parameters	see example	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>f3a9805a-4089-4b53-b9e9-b54752f91586</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.

13.2 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. As of JasperReports Server 5.6, this service supports the extended status value that includes an appropriate message.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/status/	
Options		Sample Return Value
accept: application/xml (default)		<code><status>ready</status></code>
accept: application/status+xml		<pre> <status> <errorDescriptor> <errorCode>input.controls.validation.error</errorCode> <message>Input controls validation failure</message> <parameters> <parameter>Specify a valid value for type Integer. </parameter> </parameters> </errorDescriptor> <value>failed</value> </status> </pre>
accept: application/json		<code>{ "value": "ready" }</code>
accept: application/status+json		<pre> { "value": "failed", "errorDescriptor": { "message": "Input controls validation failure", "errorCode": "input.controls.validation.error", "parameters": ["Specify a valid value for type Integer."] } } </pre>
Return Value on Success		Typical Return Values on Failure
200 OK – The content contains the report status, as shown above. In the extended format, error reports contain error messages suitable for display.		404 Not Found – When the specified requestID does not exist.

13.3 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 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 `reportExecution` 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.

```
{
  "status": "ready",
  "totalPages": 47,
  "requestId": "b487a05a-4989-8b53-b2b9-b54752f998c4",
  "reportURI": "/reports/samples/AllAccounts",
  "exports": [{
    "id": "195a65cb-1762-450a-be2b-1196a02bb625",
    "options": {
      "outputFormat": "html",
      "attachmentsPrefix": "./images/",
      "allowInlineScripts": false
    },
    "status": "ready",
    "outputResource": {
      "contentType": "text/html"
    },
    "attachments": [{
      "contentType": "image/png",
      "fileName": "img_0_46_0"
    },
    {
      "contentType": "image/png",
      "fileName": "img_0_0_0"
    },
    {
      "contentType": "image/jpeg",
      "fileName": "img_0_46_1"
    }
  ]
},
{
  "id": "4bac4889-0e63-4f09-bbe8-9593674f0700",
  "options": {
    "outputFormat": "html",
    "attachmentsPrefix": "{contextPath}/rest_v2/reportExecutions/{reportExecutionId}/exports/{exportExecutionId}/attachments/",
    "baseUrl": "http://localhost:8080/jasperserver-pro",
```



```

        "allowInlineScripts": true
    },
    "status": "ready",
    "outputResource": {
        "contentType": "text/html"
    },
    "attachments": [{
        "contentType": "image/png",
        "fileName": "img_0_0_0"
    }]
}
}

```

13.4 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 195a65cb-1762-450a-be2b-1196a02bb625. To download the main report output, specify this export ID in the following method:

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/exports/exportID/outputResource
Response Header	Description
output-final	As of JasperReports Server 5.6, this value indicates whether the output is in its final form or not. When false, report items such as total page count are not finalized, but output is available early. You should reload the output resource again until this value is true.
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/b487a05a-4989-8b53-b2b9-b54752f998c4/exports/195a65cb-1762-450a-be2b-1196a02bb625/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 13-1, “Report Execution Properties,”](#) on page 92.

Method	URL
GET	<code>http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/exports/exportID/attachments/fileName</code>
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`

13.5 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	<code>http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/exports/</code>
Content-Type	Content
application/xml application/json	Send an export 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>
```

13.6 Modifying Report Parameters

As of JasperReports Server 5.6, you can update the report parameters, also known as input controls before running report executions.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/parameters	
Argument	Type/Value	Description
freshData	default=true	When data snapshots are enabled, new parameters must force the server to get fresh data by querying the data source. This overrides the default in Table 13-1, “Report Execution Properties,” on page 92 .
Media-Type		Content
application/json		<pre>[{ "name": "someParameterName", "value": ["value 1", "value 2"] }, { "name": "someAnotherParameterName", "value": ["another value"] }]</pre>

application/xml	<pre> <reportParameters> <reportParameter name="Country_multi_select"> <value>Mexico</value> </reportParameter> <reportParameter name="Cascading_state_multi_select"> <value>Guerrero</value> <value>Sinaloa</value> </reportParameter> </reportParameters> </pre>
Return Value on Success	Typical Return Values on Failure
204 No Content – There is no content to return.	404 Not Found – When the request ID specified in the request does not exist.

13.7 Polling Export Execution

As with the execution of the main report, you can also poll the execution of the export process. As of JasperReports Server 5.6, this service supports the extended status value that includes an appropriate message.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/exports/exportID/status
Options	Sample Return Value
accept: application/xml (default)	<status>ready</status>
accept: application/status+xml	<pre> <status> <errorDescriptor> <errorCode>input.controls.validation.error</errorCode> <message>Input controls validation failure</message> <parameters> <parameter>Specify a valid value for type Integer.</parameter> </parameters> </errorDescriptor> <value>failed</value> </status> </pre>
accept: application/json	{ "value": "ready" }
accept: application/status+json	<pre> { "value": "failed", "errorDescriptor": { "message": "Input controls validation failure", "errorCode": "input.controls.validation.error", "parameters": ["Specify a valid value for type Integer."] } } </pre>

Return Value on Success	Typical Return Values on Failure
200 OK – The content contains the export status, as shown above. In the extended format, error reports contain error messages suitable for display.	404 Not Found – When the specified request ID 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 **13.4, “Requesting Report Output,” on page 97**. 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

13.8 Finding Running Reports and Jobs

The 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	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions?<arguments>">http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions?<arguments>	
Argument	Type/Value	Description
reportURI	Optional String	This string matches the repository URI of the running report, relative the currently logged-in user's organization.
jobID	Optional String	For scheduler jobs, this argument matches the ID of the job that triggered the running report.
jobLabel	Optional String	For scheduler jobs, this argument matches the name of the job that triggered the running report.
userName	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 [13.3, “Requesting Report Execution Details,” on page 95](#).

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.

13.9 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 <code>cancelled</code> . 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:</p> <p>XML: <status>cancelled</status></p> <p>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>

CHAPTER 14 THE inputControls SERVICE

The reports service includes methods for reading and setting input controls. The `rest_v2/reports/inputControls` methods return an XML descriptor by default, but you can optionally specify the JSON format. The examples in this section use the JSON format.

This chapter includes the following sections:

- [Listing Input Control Structure](#)
- [Listing Input Control Values](#)
- [Setting Input Control Values](#)

14.1 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
200 OK – The content is a JSON object that describes the input control structure. See example below.	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. 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. Each input control also has a type that indicates how the user should be allowed to make a choice:

bool	singleValue
singleSelect	singleValueText
singleSelectRadio	singleValueNumber
multiSelectCheckbox	singleValueDate
multiSelect	singleValueDatetime
	singleValueTime

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": [ { ... } ]
    "dataType": {
      "type": "number",
      "maxValue": "20",
      "strictMax": true,
      "minValue": "5",
      "strictMin": true
    }
  }
  "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"
    }
  }
}

```

14.2 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" : {
      "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~.

14.3 Setting Input Control Values

The following method validates the input controls that you send, to ensure that they can be used in 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	<p>A JSON object that lists your selected values. The value of every input control is given as an array of string values, even for single-select controls or multi-select controls with a single value. See also the example below:</p> <pre>{ "boolean-input-control" : ["true"], "integer-input-control" : ["123456"], "single-select-input-control" : ["some value"], "multiple-select-input-control-1" : ["another value"], "multiple-select-input-control-2" : ["first", "second", "third"] }</pre>	
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 [14.2, “Listing Input Control Values,” on page 107](#), 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" : {  
  "label" : "Canada",  
  "selected" : "false",  
  "value" : "Canada"  
}, {  
  "label" : "Mexico",  
  "selected" : "true",  
  "value" : "Mexico"  
}, {  
  "label" : "USA",  
  "selected" : "false",  
  "value" : "USA"  
}  
},  
...  
]  
}
```


CHAPTER 15 THE options SERVICE

This chapter describes the `rest_v2/reports/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.

This chapter includes the following sections:

- [Listing Report Options](#)
- [Creating Report Options](#)
- [Updating Report Options](#)
- [Deleting Report Options](#)

15.1 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"
    }
  ]
}
```

15.2 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"
}
```

15.3 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 [15.2, “Creating Report Options,” on page 112](#) 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"]
}
```

15.4 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.

CHAPTER 16 THE jobs SERVICE

The `rest_v2/jobs` service provides the interface to schedule reports and manage scheduled reports (also called jobs). This 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.

This chapter includes the following sections:

- **Listing Report Jobs**
- **Extended Job Search**
- **Viewing a Job Definition**
- **Scheduling a Report**
- **Viewing Job Status**
- **Editing a Job Definition**
- **Updating Jobs in Bulk**
- **Pausing Jobs**
- **Resuming Jobs**
- **Restarting Failed Jobs**
- **Deleting Jobs**
- **Specifying FTP Output**
- **Specifying File System Output**

16.1 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	<code>http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs</code>	
Argument	Type/Value	Description
reportUnitURI?	/path/to/report	Optional URI of a report or report option to list all of its jobs. When specified, the results are only for the specified report or report option.

Options	
accept: application/xml accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – The body contains an array or list of <code>jobsummary</code> descriptors.	404 Not Found – When no job is found.

The jobs are described in a `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 deprecated `rest/jobsummary (v1)` service.

```
JSON: {
  "jobsummary": [
    {
      "id": 1898,
      "version": 0,
      "reportUnitURI": "/reports/AllAccounts",
      "label": "SampleJobName",
      "description": "Accounts Sample Job",
      "owner": "jasperadmin|organization_1",
      "reportLabel": "Accounts Report",
      "state": {
        "previousFireTime": null,
        "nextFireTime": "2013-10-12T00:00:00+03:00",
        "value": "NORMAL"
      }
    },
    ...
  ]
}

XML: <jobs>
  <jobsummary>
    <id>1898</id>
    <label>SampleJobName</label>
    <description>Accounts Sample Job</description>
    <reportUnitURI>/reports/AllAccounts</reportUnitURI>
    <reportLabel>Accounts Report</reportLabel>
    <state>
      <nextFireTime>2013-10-12T00:00:00+03:00</nextFireTime>
      <value>NORMAL</value>
    </state>
    <owner>jasperadmin|organization_1</owner>
    <version>0</version>
  </jobsummary>
  ...
</jobs>
```

16.2 Extended Job Search

The GET method is also used for more advanced job searches. You can search by owner or for strings in the label, or you can match arbitrary fields of the job descriptor with a special syntax. 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	Performs a case-insensitive string search for this value anywhere within the label field of every job.
owner	string	The name of the user who scheduled the report, if necessary in the format <username>%7C<organization> (%7C is the character).
reportUnitURI?	/path/to/report	Specify the URI of a report or report option to list all of its jobs.
example?	JSON jobModel	Searches for jobs that match the JSON jobModel, which is a fragment of a job descriptor containing one or more fields to be matched. Because the JSON fragment appears as an argument to the URL, it must be properly URL-encoded (percent-encoded). See the example below.
limit	integer	Turns on pagination of the result by specifying the number of jobsummary descriptors per results page. The default is -1 for no limit and thus no pagination (all results are returned together).
offset	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.
Options		
accept: application/xml accept: application/json		
Return Value on Success		Typical Return Values on Failure
200 OK – The body contains an array or list of jobsummary descriptors that match the search criteria.		404 Not Found – When no matching job is found in the server.

The body of the response contains `jobsummary` elements, in the same format as shown in [16.1, “Listing Report Jobs,” on page 115](#).

The `example` parameter lets you specify a search on fields in the job descriptor, such as output formats. You can specify any field in the job descriptor or in any of its nested structures. 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 an output format of PDF. The JSON `jobModel` 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
```

16.3 Viewing a Job Definition

Once you search for and find the ID of a job, use the GET method with that specific job ID to retrieve its detailed information.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/<jobID>/
Options	
accept: application/job+xml accept: application/job+json	
Return Value on Success	Typical Return Values on Failure
200 OK – The body contains a descriptor with all details about the job.	404 Not Found – When the specified job is not found in the server.

The GET method returns a descriptor that fully describes all the aspects of a scheduled job, such as recurrence, parameters, output, email notifications, and alerts, if any. For additional options around file output, see [16.12, “Specifying FTP Output,” on page 128](#) and [16.13, “Specifying File System Output,” on page 130](#).

JSON:

```
{
  "id": 1906,
  "version": 0,
  "username": "jasperadmin|organization_1",
  "label": "Sample Job Name",
  "description": "Sample description",
  "creationDate": "2013-10-11T14:52:04.955+03:00",
  "outputFormats": {
    "outputFormat": ["XLS",
      "PDF"]
  }
}
```

```

}

"trigger": {
  "simpleTrigger": {
    "id": 0,
    "version": 0,
    "timezone": "America/Los_Angeles",
    "calendarName": null,
    "startType": 2,
    "startDate": "2016-10-26 10:00",
    "endDate": null,
    "misfireInstruction": 0,
    "occurrenceCount": 1,
    "recurrenceInterval": null
  }
},

"source": {
  "reportUnitURI": "/adhoc/topics/Cascading_multi_select_topic",
  "parameters": {
    "parameterValues": {
      "Country_multi_select": ["Mexico"],
      "Cascading_name_single_select": ["Chin-Lovell Engineering Associates"],
      "Cascading_state_multi_select": ["DF",
        "Jalisco",
        "Mexico"]
    }
  }
},

"alert": {
  "id": 0,
  "version": -1,
  "recipient": "OWNER_AND_ADMIN",
  "toAddresses": {
    "address": []
  },
  "jobState": "FAIL_ONLY",
  "messageText": null,
  "messageTextWhenJobFails": null,
  "subject": null,
  "includingStackTrace": true,
  "includingReportJobInfo": true
},

"baseOutputFilename": "Cascading_multi_select_report",
"outputLocale": null,
"mailNotification": null,
"outputTimeZone": null,
"repositoryDestination": {
  "id": 0,
  "version": -1,
  "folderURI": "/temp",
  "sequentialFileNames": false,
  "overwriteFiles": false,
  "outputDescription": null,
  "timestampPattern": null,

```

```

        "saveToRepository": true,
        "defaultReportOutputFolderURI": null,
        "usingDefaultReportOutputFolderURI": false,
        "outputLocalFolder": null,
        "outputFTPInfo": {
            "userName": "anonymous",
            "password": null,
            "folderPath": null,
            "serverName": null,
            "type": "ftp",
            "protocol": null,
            "port": 990,
            "implicit": true,
            "pbsz": 0,
            "prot": null
        }
    },
}

```

XML:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<clientJob>
  <creationDate>2015-11-05T13:38:09.759+02:00</creationDate>
  <id>5484</id>
  <label>test</label>
  <username>superuser</username>
  <version>0</version>
  <outputFormats>
    <outputFormat>PDF</outputFormat>
  </outputFormats>
  <simpleTrigger>
    <id>5482</id>
    <misfireInstruction>0</misfireInstruction>
    <startDate>2015-11-06T00:00:00+02:00</startDate>
    <startType>2</startType>
    <timezone>Europe/Helsinki</timezone>
    <version>0</version>
    <occurrenceCount>1</occurrenceCount>
    <startDate>2015-11-06 00:00</startDate>
  </simpleTrigger>
  <source>
    <parameters>
      <parameterValues>
        <entry>
          <key>Country_multi_select</key>
          <value xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:-
type="collection">
            <item xmlns:xs="http://www.w3.org/2001/XMLSchema"
xsi:type="xs:string">Mexico</item>
          </value>
        </entry>
        <entry>
          <key>Cascading_name_single_select</key>
          <value xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:-

```



```

type="collection">
    <item xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">Chin-
Lovell Engineering Associates</item>
    </value>
</entry>
<entry>
    <key>Cascading_state_multi_select</key>
    <value xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:-
type="collection">
        <item xmlns:xs="http://www.w3.org/2001/XMLSchema"
xsi:type="xs:string">DF</item>
        <item xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:-
type="xs:string">Jalisco</item>
        <item xmlns:xs="http://www.w3.org/2001/XMLSchema"
xsi:type="xs:string">Mexico</item>
    </value>
</entry>
</parameterValues>
</parameters>
<reportUnitURI>/organizations/organization_1/adhoc/topics/Cascading_multi_select_top-
ic</reportUnitURI>
</source>

<outputTimeZone>Europe/Helsinki</outputTimeZone>
<baseOutputFilename>Cascading_multi_select_topic</baseOutputFilename>
<repositoryDestination>
    <folderURI>/organizations/organization_1/adhoc/topics</folderURI>
    <id>5483</id>
    <outputFTPInfo>
        <implicit>true</implicit>
        <password/>
        <pbsz>0</pbsz>
        <port>21</port>
        <type>ftp</type>
        <userName>anonymous</userName>
    </outputFTPInfo>
    <overwriteFiles>true</overwriteFiles>
    <saveToRepository>true</saveToRepository>
    <sequentialFileNames>false</sequentialFileNames>
    <usingDefaultReportOutputFolderURI>false</usingDefaultReportOutputFolderURI>
    <version>-1</version>
</repositoryDestination>
</clientJob>

```

16.4 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 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/job+xml application/job+json	A well-formed XML or JSON job descriptor as described.
Options	
accept: application/job+xml accept: application/job+json	
Return Value on Success	Typical Return Values on Failure
201 Created – The body contains the 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.

When you schedule a report, you need only specify the relevant fields in the job descriptor. For example in JSON:

```
{
  "label": "Sample Job Name",
  "description": "Sample description",
  "trigger": {
    "simpleTrigger": {
      "timezone": "America/Los_Angeles",
      "startType": 2,
      "startDate": "2013-10-26 10:00",
      "occurrenceCount": 1
    }
  },
  "source": {
    "reportUnitURI": "/adhoc/topics/Cascading_multi_select_topic",
    "parameters": {
      "parameterValues": {
        "Country_multi_select": ["Mexico"],
        "Cascading_name_single_select": ["Chin-Lovell Engineering Associates"],
        "Cascading_state_multi_select": ["DF",
        "Jalisco",
        "Mexico"]
      }
    }
  },
  "baseOutputFilename": "Cascading_multi_select_report",
  "outputTimeZone": "America/Los_Angeles",
  "repositoryDestination": {
    "folderURI": "/temp"
  },
  "outputFormats": {
    "outputFormat": ["PDF", "XLS"]
  }
}
```

The response of the PUT request is the same as the GET request shown in [16.3, “Viewing a Job Definition,” on page 118](#). It includes all the fields of the job descriptor, including the server-assigned ID and all the null fields.

16.5 Viewing Job Status

The following method returns the current run time 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.

16.6 Editing a Job Definition

The POST method replaces the entire definition of a job with a new descriptor. To modify an existing job definition, use the GET method to read its job descriptor, modify the descriptor as required, then use the POST method of the jobs service.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/<jobID>/
Content-Type	Content
application/job+xml application/job+json	A well-formed XML or JSON job descriptor. The request can include either a complete descriptor such as the result of a GET request described in 16.3, “Viewing a Job Definition,” on page 118 , or it can be a minimally sufficient descriptor as shown in 16.4, “Scheduling a Report,” on page 121 .
Options	
accept: application/job+xml accept: application/job+json	
Return Value on Success	Typical Return Values on Failure
200 OK – The response include the complete job descriptor, for an example, see 16.3, “Viewing a Job Definition,” on page 118 .	404 Not Found – When the specified <jobID> does not exist.

16.7 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 job IDs to update.
replace Trigger IgnoreType	true / false	Specify true when you send a new trigger type. By default, this is false, and the trigger can be updated but not changed to a different type. See below.
Content-Type		Content
application/json application/xml		A well-formed <code>jobModel</code> descriptor, which is a fragment of a job descriptor containing only the fields to be updated. See example below.
Options		
accept: application/json accept: application/xml		
Return Value on Success		Typical Return Values on Failure
200 OK – The array or list of jobs that were restarted.		404 Not Found – When the specified <jobID> does not exist.

In this usage, the POST method allows you to send a partial job description, called a `jobModel`, that contains any subset of the job descriptor's fields. This update applies to one or more jobs whose ID is specified by the `id` argument. For example, the following simple request will update the job description in several jobs:

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

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

However, the `jobModel` provides two mechanisms to create complex updates:

- You can describe nested structures by using the *nestedNameModel* equivalent element. Like the `jobModel`, nested model elements contain only the subset that you want to modify. Thus you could change one value within a parameter, the end date within a schedule, or an email address within a notification.
- You can remove the definition of an element by using the *isElementNameModified* element and giving it the value `true`. This indicates that the element's new value is null, and thus that the element should be removed altogether from the job descriptor.

In the following example, the description will be removed from the target jobs, and their triggers will be modified. Note that XML descriptors do not use the `trigger` element and thus do not have a `triggerModel` element:

```

JSON: {
  "label": "Modified label",
  "isDescriptionModified": true,
  "triggerModel": {
    "simpleTriggerModel": {
      "timezone": "Europe/Helsinki",
    }
  }
  "baseOutputFilename": "NewOutputName"
}

XML: <jobModel>
  <label>Modified label</label>
  <isDescriptionModified>true</isDescriptionModified>
  <simpleTriggerModel>
    <timezone>Europe/Helsinki</timezone>
  </simpleTriggerModel>
  <baseOutputFilename>NewOutputName</baseOutputFilename>
</jobModel>

```

The response has an array or list of `jobId` elements that were updated:

```

JSON: {"jobId": [8321, 8322]}

XML: <jobIdList>
  <jobId>8321</jobId>
  <jobId>8322</jobId>
</jobIdList>

```

16.8 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 application/json		An array or list of job IDs to pause. See example below. If the body of the request is empty, or the list is empty, all jobs in the scheduler will be paused.
Options		
accept: application/json accept: application/xml		
Return Value on Success		Typical Return Values on Failure
200 OK – The array or list of jobs that were paused.		

The request and the response have the same format, an array or list of `jobId` elements:

```
JSON: {"jobId": [1236, 1237, 1238, 1239]}

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

16.9 Resuming Jobs

Use the following method to resume any or all paused jobs in the scheduler. Resuming a job means that any defined trigger in the schedule that occurs after the time it is resumed will cause the report to run again. Missed schedule triggers that occur before the job is resumed are never run.

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/resume/	
Content-Type		Content
application/xml application/json		An array or list of job IDs to resume. See example below. If the body of the request is empty, or the list is empty, all paused jobs in the scheduler will resume.
Options		
accept: application/json accept: application/xml		
Return Value on Success		Typical Return Values on Failure
200 OK – The array or list of jobs that were resumed.		

The request and the response have the same format, an array or list of `jobId` elements:

```
JSON: {"jobId": [1236, 1237]}

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

16.10 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 application/json		An array or list of job IDs to restart. See example below.
Options		
accept: application/json accept: application/xml		
Return Value on Success		Typical Return Values on Failure
200 OK – The array or list of jobs that were restarted.		

The request and the response have the same array or list of `jobId` elements:

```
JSON: { "jobId": [8321, 8322] }
```

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

16.11 Deleting Jobs

Use the DELETE method to remove jobs from the scheduler. There are two forms to specify a single job or multiple jobs to delete.

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

Method	URL	
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs?<arguments>	
Argument	Type/Value	Description
id	Multiple String	Enter as many job IDs as you want to delete, for example: ?id=5594&id=5645&id=5761
Options		
accept: application/xml accept: application/json		
Return Value on Success		Typical Return Values on Failure
200 OK – The content is a list of deleted jobs, as shown in the example below.		

The list of deleted jobs in the response has an array or list of `jobId` elements:

```
JSON: {"jobId": [5594, 5645, 5761]}
```

```
XML: <jobIdList>
      <jobId>5594</jobId>
      <jobId>5645</jobId>
      <jobId>5761</jobId>
    </jobIdList>
```

16.12 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>
    <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.

As of JasperReports Server 6.3, FTP output also supports SSH private keys stored in the repository for authentication. All of the fields for the `outputFTPInfo` element are described in the following table:

Field	Description	JSON Example
type	Type of FTP connection requested: ftp (default), ftps, or sftp.	"type": "{ftp ftps sftp}"
serverName	The host name of the FTP server.	"serverName": "ftp.example.com"
port	Integer value that specifies the port number of the ftp server. The default value depends on the connection type: ftp = 21, sftp = 22, and ftps = 990.	"port": 21
userName	The login user name for the FTP server.	"userName": "anonymous"
password	The login password for the given userName on the FTP server.	"password": "securePassword"
folderPath	The path of the folder where the job output resources should be created.	"folderPath": "/path/to/folder"
implicit	Specifies the security mode for FTPS, Implicit if true (default) or Explicit if false. If implicit is true, the default port is set to 990.	"implicit": {true false}
protocol	Specifies the secure socket protocol to be used, for example SSL or TLS.	"protocol": "TLS"

Field	Description	JSON Example
prot	Specifies the PROT command for FTP. Supported values: <ul style="list-style-type: none"> • C - Clear • S - Safe (SSL protocol only) • E - Confidential (SSL protocol only) • P - Private 	"prot": "C"
pbsz	Specifies the protection buffer size: 0 to (2 ³²)-1 decimal integer. The default is 0.	"pbsz": 0
sshKey	Repository URI of the SSH private key resource (used for SFTP authentication).	"sshKey": "/sshKeys/myOpenSSHKey"
sshPassphrase	The passphrase for the SSH private key (used for SFTP authentication).	"sshPassphrase": "mySecurePhrase"

Administrators can store SSH key files in the repository by right-clicking a folder and selecting **Add Resource > File > Secure File** from the context menu. For more information, see the *JasperReports Server Administrator Guide*.

16.13 Specifying File System Output

When configured, you can also specify a path on the local file system to write job output. The user running the server process must have write permission in that location.

In order for file system output to work, the server must be properly configured. In the file `.../WEB-INF/applicationContext.xml`, you must set the `enableSaveToHostFS` property to `true`. As described in the configuration chapter of the *JasperReports Server Administrator Guide*, this setting also enables file system output from the scheduler user interface for all users, which could be a security risk.

To create a job with file system output, add the `outputLocalFolder` 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>
    <overwriteFiles>true</overwriteFiles>
    <sequentialFileNames>false</sequentialFileNames>
    <outputLocalFolder>/temp/scheduledReports/</outputLocalFolder>
  </repositoryDestination>
  <outputFormats>
    <outputFormat>XLS</outputFormat>
    <outputFormat>PDF</outputFormat>
  </outputFormats>
  ...
</job>
```

As with FTP output, file system output is always specified in addition to repository output, and the output will be written to both the repository and the local folder. 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 file system output.

CHAPTER 17 THE calendars SERVICE

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 `rest_v2/jobs/calendars` service defines any number of exclusion calendars. 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.

This chapter includes the following sections:

- [Listing All Registered Calendar Names](#)
- [Viewing an Exclusion Calendar](#)
- [Deleting an Exclusion Calendar](#)
- [The calendars Service](#)

17.1 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[-pro]/ rest_v2/jobs/calendars /?.<parameter>	
Argument	Type/Value	Description
calendar Type	optional string	A type of calendar to return: annual, cron, daily, holiday, monthly, or weekly. See 17.4, “Adding or Updating an Exclusion Calendar,” on page 135 for a description of the various types. You may specify only one <code>calendarType</code> parameter. When <code>calendarType</code> isn't specified, all calendars names are returned. If <code>calendarType</code> has an invalid value, then an empty collection is returned.

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>
```

17.2 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[-pro]/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>
```

17.3 Deleting an Exclusion Calendar

Use the following method to delete a calendar by name.

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

17.4 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[-pro]/rest_v2/jobs/calendars/<calendarName>?<arguments>	
Argument	Type/Value	Description
replace?	true / false	If true, any existing calendar 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 – The calendars are updates.		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:

- 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>
```

```

</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>
```


CHAPTER 18 THE queryExecutor SERVICE

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

Use the GET method to specify the query string in the request as an argument.

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>
      <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 time zone as stored in the database. These dates and times are converted to the server's time zone.

CHAPTER 19 THE caches SERVICE

The rest_v2/caches service allows you to clear the caches used by virtual data sources. Virtual data sources use the Teiid engine that lets you combine data from several data sources such as JDBC, JNDI, and several flavors of big data. In order to join the data, the Teiid engine uses an internal cache to store data. You can use this service to clear this cache, for example after updating your data sources.

For now this service provides only cache deletion for virtual data sources.

Method	URL	
DELETE	http://<host>:<port>/jasperserver-pro/rest_v2/caches/vds/	
Return Value on Success		Typical Return Values on Failure
204 No Content – There is nothing to return.		404 Not Found – When the specified cache does not exist.

CHAPTER 20 THE 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.

The `rest_v2/organizations` service provides methods that allow you to list, view, create, modify, and delete organizations (also known as tenants). Search functionality allows you to find 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.

Only administrative users may access the organizations service. System admins (`superuser`) can operate on top-level organizations, and organization admins (`jasperadmin`) can operate on their own organization or any sub-organizations.

This chapter includes the following sections:

- [Searching for Organizations](#)
- [Viewing an Organization](#)
- [Creating an Organization](#)
- [Modifying Organization Properties](#)
- [Setting the Theme of an Organization](#)
- [Deleting an Organization](#)

20.1 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	<code>http://<host>:<port>/jasperserver-pro/rest_v2/organizations?<arguments></code>

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.
sortBy	Optional String	Specifies a sort order for results. When not specified, lists of organizations are in the order that they were created. The possible values are: name – Sort results alphabetically by organization name. alias – Sort results alphabetically by organization alias. id – Sort results alphabetically by organization ID.
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>

```

20.2 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"
}

```

20.3 Creating an Organization

To create an organization, put all information in an organization descriptor, and include it in a POST request to the 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 (<code>joeuser</code> , <code>jasperadmin</code>) 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.		<p>404 Not Found – When the ID of the parent organization cannot be resolved.</p> <p>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:</p> <p>id and alias: <code>~!+-#\$\$%^ </code></p> <p>tenantName: <code> &*?<>/\</code></p>

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"
}
```

20.4 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		<p>A partial organization descriptor that includes the properties to change. Do not specify the following properties:</p> <ul style="list-style-type: none"> <code>id</code> – The organization ID is permanent and can never be modified. <code>parentId</code> – Organizations cannot change parents. <code>tenantUri</code> – Organizations cannot change the organization hierarchy. <code>tenantFolderUri</code> – 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"
}
```

20.5 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 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.

20.6 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.

CHAPTER 21 THE users SERVICE

The `rest_v2/users` service provides methods that allow you to list, view, create, modify, and delete user accounts, including setting role membership. The 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.

Only administrative users may access the users service. System admins (`superuser`) can define and modify users anywhere in the server, and organization admins (`jasperadmin`) can define and modify users within their own organization or any sub-organizations.

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.

This chapter includes the following sections:

- **Searching for Users**
- **Viewing a User**
- **Creating a User**
- **Modifying User Properties**
- **Deleting a User**

21.1 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. 204 No Content – The search did not return any users.		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 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>

```

21.2 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	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID">http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/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}
  ]
}
```

21.3 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 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 [21.4, “Modifying User Properties,” on page 156](#).

Method	URL
PUT	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID">http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/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 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: <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 that includes the name and organization of each role. The following example shows the descriptor in JSON format:

```
{
  "fullName":"Joe User",
  "emailAddress":"juser@example.com",
  "enabled":false,
  "password":"mySecretPassword",
  "roles":[
    {"name":"ROLE_MANAGER", "tenantId":"organization_1"}]
}
```

21.4 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 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 [21.3, "Creating a User," on page 155](#).

Method	URL
PUT	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID">http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/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: 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", "tenantId":"organization_1"}
  ]
}
```

21.5 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.

CHAPTER 22 THE roles SERVICE

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

Only administrative users may access the roles service. System admins (`superuser`) can define and set roles anywhere in the server, and organization admins (`jasperadmin`) can define and set roles within their own organization or any sub-organizations.

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.

This chapter includes the following sections:

- **Searching for Roles**
- **Viewing a Role**
- **Creating a Role**
- **Modifying a Role**
- **Setting Role Membership**
- **Deleting a Role**

22.1 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	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/roles?<arguments>">http://<host>:<port>/jasperserver[-pro]/rest_v2/roles?<arguments> <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/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 (name) 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 all users' roles). When false or not specified, all roles of all specified users are found (union of all users' roles).
include SubOrgs	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 External Authentication Cookbook*.

22.2 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"
}
```

22.3 Creating a Role

To create a role, send the PUT request to the 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	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/roles/roleID">http://<host>:<port>/jasperserver[-pro]/rest_v2/roles/roleID <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/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 <code><role></role></code> or <code>{}</code> . Do <i>not</i> specify the following properties: <code>name</code> – Specified in the URL and should not 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.
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.

22.4 Modifying a Role

To change the name of a role, send a PUT request to the 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: name – 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.

22.5 Setting Role Membership

To assign role membership to a user, set the roles property on the user account with the PUT method of the users service. For details, see [21.4, “Modifying User Properties,” on page 156](#).

22.6 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	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/roles/roleID">http://<host>:<port>/jasperserver[-pro]/rest_v2/roles/roleID <a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/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 23 THE attributes SERVICE

Attributes are name-value pairs that are associated with users, organizations, or the server. Unlike roles, attributes are not predefined, and thus any attribute name can be assigned any value at any time. When running dashboards, views, or reports, certain advanced features of the server will reference attribute values of the currently logged-in user (or of the organization of the currently logged-in user), so that behavior is customized for that user.

For example, Domain security files and OLAP access grants may reference attributes in addition to roles to grant certain permissions. Attributes may also be referenced when defining the fields of a data source, thereby making database access customized for each user or organization. Finally, application developers may use the attributes service in order access or store information that can enhance their embedded BI solutions.

The `rest_v2/attributes` service provides methods for reading, writing, and deleting attributes at the server, organization, or user level. Only administrative users may access the attributes service. System admins (`superuser`) can set attributes anywhere in the server, and organization admins (`jasperadmin`) can set attributes within their own organization or any sub-organizations. Because of the nature of attributes, organization admins may see attributes from parent organizations and override them if allowed to do so by the parent administrator.

Attributes used to be called profile attributes because they were associated only with users. As of JasperReports Server 6.0, the attributes service applies to users, organization, and the root organization representing the server.

This chapter includes the following sections:

- **Attribute Descriptors**
- **Secure Attributes**
- **Entities with Attributes**
- **Permissions for Accessing Attributes**
- **Referencing Attributes**
- **Attribute Limitations**
- **Viewing Attributes**
- **Setting Attributes**
- **Deleting Attributes**

23.1 Attribute Descriptors

Attributes are represented as a pair of string fields, one for the name of the attribute, the other for its value. For example, the following JSON structure defines an attribute:

```
{
  "name": "Attr1",
  "value": "Value1"
}
```

Each attribute may only have one value, however that value may contain a comma-separated list that, in certain uses, is interpreted by the server as being multi-valued. Such attributes can be used in Domain security filters that match against a collection of values.

```
{
  "name": "Attr2",
  "value": "Value2a, Value2b, Value2c"
}
```

Attributes with the same name may be defined on different entities. For example, a user has a specific value for an attribute, the organization he belongs to has a default value for the same attribute, and the server level has yet another value for it. In this example, three separate attributes are defined, but they have the same name because they occur on different entities. The mechanisms described in [23.5, “Referencing Attributes,” on page 167](#) can take advantage of this to implement default values.

23.2 Secure Attributes

JasperReports Server 6.0 also introduces the notion of secure attribute that can be used to store sensitive information such as a password. Secure attributes have the following properties:

- Their values are stored in encrypted form in the server's internal database.
- Their values are write-only through the REST service; their value is never returned.
- Their values are never displayed in the user interface; only ●●● or *** symbols are shown.
- Their value is decrypted only when referenced internally, for example as the password field in a data source.

When reading the value of a secure attribute, the server returns the field `"secure": "true"` instead of the `"value"` field. Applications that read attributes must test for this case:

```
{
  "name": "Attr3",
  "secure": "true"
}
```

When setting the value of a secure attribute, your application should specify both the secure field and the value field.

```
{
  "name": "Attr3"
  "value": "SecureValue3"
  "secure": "true"
}
```

Applications that set secure attributes should consider enabling HTTPS so that the clear-text value of the attribute is encrypted in all communication with the server.

23.3 Entities with Attributes

The entities that may have attributes are user accounts, organizations, and the server itself, represented by the root organization. The entity is specified in the URL invoking the attributes service. The URL has the following form:

`http://<host>:<port>/jasperserver[-pro]/rest_v2/<entity>attributes<parameters>`

The syntax of `<entity>` depends on the target entity for the operation and the type of server.

Commercial Edition	Syntax of <entity>
User	organizations/organizationID/users/userID/
Organization-level	organizations/organizationID/
Server Admin	users/userID/
Server-level	<blank> (the attributes apply to the "root")
Community Edition	Syntax of <entity>
User	users/userID/
Server-level	<blank> (the attributes apply to the "root")



When specifying the organization, use its unique ID, not its path. In commercial edition servers that use the single default organization, you must specify `organization_1`.

23.4 Permissions for Accessing Attributes

Only API calls that include administrator credentials may view, set, or delete attributes on users, organizations, or the server. Non-administrative users can't view or edit attributes, even on their own user account.

In commercial editions of the server, operations on attributes follow the visibility rules for organizations:

- Organization admins (`jasperadmin` by default) can view and edit attributes on their own organization, their users, any of their sub-organizations, and the users in any sub-organizations.
- Organization admins can't view or edit attributes in any parent or sibling organizations.
- Only the server admin (`superuser` by default) can view and edit attributes at the server level, represented as the root organization.
- Server admins can view and edit attributes on any organization or sub-organization in the server, as well as on any user account in any organization.
- Only a server admin can view and edit attributes on other server admins (users of the root organization).

23.5 Referencing Attributes

As mentioned, several internal mechanisms of the server read attributes on users and organizations and make use of their values in some way:

- Domain security files: you can reference attribute values associated with the logged-in user (or his organization) to create rules to access data in the Domain. For more information, see the chapter "Advanced Domains Features" in the *JasperReports Server User Guide*.
- Data source definitions: the fields that define a data source, such as its server, port number, database, and user credentials, can all reference attributes of the logged-in user's organization (or a server-specific attribute). In this way, different organizations or different servers can share the same data source yet still access a different database. For more information, see the chapter "Data Sources" in the *JasperReports Server Administrator Guide*.

The server provides two different methods to reference attributes:

- Categorical reference: requests the value of a named attribute from a specific entity, either the logged-in user's profile, the logged-in user's organization, or from the server-wide set of attributes. If the named attribute is not defined in the specified entity, an error is returned.
- Hierarchical reference: searches for the value of a named attribute first in the logged-in user's account, and if not found, then in the logged-in user's organization, and if still not found, then at the server level. This allows attributes to be defined at several levels, with the definition at a lower level (the user profile) having higher priority, and the definition at a higher level (the organization or server level) providing a default value. If the named attribute is not defined at any level, an error is returned.

The methods you use to reference attributes will then determine the entities where you need to create attributes and the values of those attributes.

23.6 Attribute Limitations

Attributes have the following limitations in the attributes service:

- The user ID and organization ID are specified in the URL, and therefore must be less than 100 characters long and not contain spaces or special symbols.
- Attribute names and attribute values being written with this service are limited to 255 characters and may not be empty (null) nor contain only whitespace characters.

The attributes service detects these conditions and returns errors accordingly:

Error Code	Description
too_long_name	Attribute's name is longer than 255 characters.
too_long_value	Attribute's value is longer than 255 characters.
empty_name	Attribute's name is empty or contains only whitespaces.
empty_value	Attribute's value is empty or contains only whitespaces.

Some methods of the attributes service operate on multiple attributes on a given entity. Such batch operations are not transactional, meaning the operation terminates with no rollback functionality when encountering an error. Attributes that have been processed (modified or deleted) before the error remain so, and attributes after the error are not processed.

All attribute operations apply to a single specific entity; there are no operations for reading or setting attributes on multiple entities.

23.7 Viewing Attributes

The GET method of the attributes service retrieves the list of attributes, if any, defined for the specified entity (a user, an organization, or the server-level). For possible values of <entity> in the URL, see [23.3, “Entities with Attributes,” on page 167](#).

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

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/<entity>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 entity.
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 entity. 204 No Content – The search did not return any attributes or the entity 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. The following example shows user-level attributes in JSON format:

GET http://localhost:8080/jasperserver-pro/rest_v2/organizations/organization_1/users/joeuser/attributes

```
{
  "attribute":[
    {
      "name": "Attr1",
      "value":"Value1"
    },
    ...
    {
      "name": "AttrN",
      "value":"ValueN"
    }
  ]
}
```

The second syntax reads a single attribute by specifying its name in the URL:

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/<entity>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 attribute for the given entity.	404 Not Found – When the user ID, organization ID, or attribute name does not match any user, organization, or attribute. The content includes an error message.

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

GET http://localhost:8080/jasperserver-pro/rest_v2/organizations/organization_1/attributes/Attr2

```
{
  "name": "Attr2",
  "value": "Value2a, Value2b, Value2c"
}
```

23.8 Setting Attributes

The PUT method of the attributes service adds or replaces attributes on the specified entity (a user, an organization, or the server-level). For possible values of <entity> in the URL, see [23.3, “Entities with Attributes,” on page 167](#).

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

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/<entity>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
<p>201 Created – When the attributes were successfully created on the given entity.</p> <p>200 OK – When the attributes were successfully updated.</p>	<p>404 Not Found – When the user ID or organization ID does not match any user or organization. The content includes an error message.</p> <p>400 Bad Request – When an attribute name or value is null, blank, or too long (see 23.6, “Attribute Limitations,” on page 168). If one attribute causes an error, the operation stops and returns an error, but attributes that were already set remain.</p>

The following example shows how to set all attributes on an organization. The list of attributes in JSON format defines the name and value of each attribute.

PUT http://localhost:8080/jasperserver-pro/rest_v2/organizations/organization_1/attributes

```
{
  "attribute": [
    {
      "name": "Attr1",
      "value": "newValue1"
    },
    {
      "name": "Attr2",
      "value": "newValue2a, newValue2b"
    },
    {
      "name": "Attr3"
      "value": "SecureValue3"
      "secure": "true"
    }
  ]
}
```

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

Method	URL
PUT	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/<entity>attributes/attrName">http://<host>:<port>/jasperserver[-pro]/rest_v2/<entity>attributes/attrName
Content-Type	Content
<p>application/xml</p> <p>application/json</p>	<p>A single attribute name-value pair. The attribute name must match the attrName 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.</p>

Return Value on Success	Typical Return Values on Failure
201 Created – When the attribute was successfully created on the given entity. 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-pro/rest_v2/organizations/organization_1/users/joeuser/attributes/Attr2

```
{
  "name": "Attr2",
  "value": "NewValue2"
}
```

23.9 Deleting Attributes

The DELETE method of the attributes service removes attributes from the specified entity (a user, an organization, or the server-level). When attributes are removed, both the name and the value of the attribute are removed, not only the value. For possible values of <entity> in the URL, see [23.3, “Entities with Attributes,” on page 167](#).

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

Method	URL	
DELETE	<a href="http://<host>:<port>/jasperserver[-pro]/rest_v2/<entity>attributes?<arguments>">http://<host>:<port>/jasperserver[-pro]/rest_v2/<entity>attributes?<arguments>	
Argument	Type	Description
name	Optional String	Specify an attribute name to remove that attribute. Repeat this argument to delete multiple attributes. When this argument is omitted, all attributes are deleted from the given entity.
Return Value on Success		Typical Return Values on Failure
204 No Content – The attributes were successfully removed from the given entity.		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 (see 23.6, “Attribute Limitations,” on page 168). 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 entity.

Method	URL	
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/<entity>attributes/attrName	
Return Value on Success		Typical Return Values on Failure
204 No Content – The attribute was successfully removed from the given entity.		<p>404 Not Found – When the user ID, organization ID, or attribute name does not match any user, organization, or attribute. The content includes an error message.</p> <p>400 Bad Request – When an attribute name is null, blank, or too long (see 23.6, “Attribute Limitations,” on page 168).</p>

