JasperReports® Server

REST API Reference

Version 9.0.0 | January 2024

Contents

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 The first REST API with the rest/ prefix and the earlier SOAP API (Simple Object Access Protocol) are deprecated and no longer maintained.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 garanteed 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

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:

REST API Services and URLs

Web Service	URLs
Login (optional)	http://<host>:<port>/jasperserver[-pro]/rest_v2/login http://<host>:<port>/jasperserver[-pro]/logout.html http://<host>:<port>/jasperserver[-pro]/rest_v2/serverInfo

Web Service	URLs
Repository	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources
	http://<host>:<port>/jasperserver-pro/rest_v2/domains/.../metadata *
	http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions
	http://<host>:<port>/jasperserver[-pro]/rest_v2/export
	http://<host>:<port>/jasperserver[-pro]/rest_v2/import
	http://<host>:<port>/jasperserver[-pro]/rest_v2/keys
Reports	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports
	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions
	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/.../inputControls
	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/.../options
	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs
	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts
	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts/{alertId}
	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts?id={ID_1}&id={ID_2}&...&id={ID_n)
	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts/pause/
	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts/resume/
	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts/restart/
	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts/calendars
	http://<host>:<port>/jasperserver-pro/rest_v2/queryExecutor *
	http://<host>:<port>/jasperserver-pro/rest_v2/caches/vds *
Administration without organizations	http://<host>:<port>/jasperserver[-pro]/rest_v2/users
	http://<host>:<port>/jasperserver[-pro]/rest_v2/users/.../attributes
	http://<host>:<port>/jasperserver[-pro]/rest_v2/roles

Web Service	URLs
	http://<host>:<port>/jasperserver[-pro]/rest_v2/attributes
Administration with organizations *	http://<host>:<port>/jasperserver-pro/rest_v2/organizations http://<host>:<port>/jasperserver-pro/rest_v2/organizations/.../attributes http://<host>:<port>/jasperserver-pro/rest_v2/organizations/.../users http://<host>:<port>/jasperserver-pro/rest_v2/organizations/.../users/.../attributes http://<host>:<port>/jasperserver-pro/rest_v2/organizations/.../roles http://<host>:<port>/jasperserver-pro/rest_v2/attributes
* Available only in commercial editions of JasperReports Server.	

For progammers 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

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
```

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.

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

Success Messages		Client Error		Server Errors	
Code	**Message**	**Code**	**Message**	**Code**	**Message**
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		

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.

Deprecated REST v1 Services

Web Service	URLs
Login	http://<host>:<port>/jasperserver[-pro]/rest/login http://<host>:<port>/jasperserver[-pro]/j_spring_security_check http://<host>:<port>/jasperserver[-pro]/GetEncryptionKey
Repository	http://<host>:<port>/jasperserver[-pro]/rest/resources http://<host>:<port>/jasperserver[-pro]/rest/resource http://<host>:<port>/jasperserver[-pro]/rest/permission
Reports	http://<host>:<port>/jasperserver[-pro]/rest/report http://<host>:<port>/jasperserver[-pro]/rest/jobsummary http://<host>:<port>/jasperserver[-pro]/rest/job
Administration without organizations	http://<host>:<port>/jasperserver[-pro]/rest/user http://<host>:<port>/jasperserver[-pro]/rest/attribute http://<host>:<port>/jasperserver[-pro]/rest/role
Administration with organizations *	http://<host>:<port>/jasperserver-pro/rest/organization http://<host>:<port>/jasperserver-pro/rest/user http://<host>:<port>/jasperserver-pro/rest/attribute 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.

Deprecated SOAP Web Services

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

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

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

```
<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.

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

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:

- Stateless authentication - Your client sends user credentials with every API request. This is the traditional RESTful behavior and fully supported by JasperReports Server. Clients may send credentials using HTTP Basic Authentication, where the user ID and password are sent in the header with every request, or argument-based authentication, where the user ID and password are included in URL arguments.
- User session management - Your client performs a login operation first, and then sends a session cookie with every API request. As with a user interface, your client performs a log out operation when done. Use of the login and log out services is optional, but it can improve performance under heavy user loads.

Normally, RESTful implementations do not rely on the persistent user 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 method takes advantage of this. There are several use cases for either type of authentication.

- If your client makes sporadic requests, for example running a report every hour, it is easier to use basic authentication and send the credentials with each request. See HTTP Basic Authentication.
- If a username or password contains UTF-8 characters, it may be corrupted by basic authentication and the service will always return an error. In this case, you can send the username and password in URL arguments with each request. See Argument-based Authentication.
- If your client applications perform many requests in a short time, you can avoid the overhead of stateless authentication by using the login service once and passing the session ID cookie instead with each request. For more information, see The login Service.
- 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 is 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 log out call to make sure the memory is freed. For more information, see Logout.

As with logging in from the web UI, you can send a user-specific locale and time zone during REST API authentication. To specify a locale and timezone, choose from the following possibilities:

- Use locale and time zone arguments on any REST API to specify the language and time in the response, for example to localize a report. It is also possible for the same user to make several requests with different locales or time zones. Once you specify a locale or time zone for a given user, the server sets a cookie so that it applies to all requests. See Argument-based Authentication.
- When doing many requests with the same locale and time zone, you can also specify the locale and time zone arguments with the login service. The language and time will be set with a cookie for all future requests. See The login Service.
- If you never specify any locale or time zone arguments, the default locale and default time zone on the server will be used for all operations.

In the case of external authentication, how you perform REST authentication depends on the type of mechanism:

- If your server is configured with an external authentication that requires a username and password, such as LDAP, then you can use any authentication method that submits those values: HTTP basic authentication, argument-based authentication, or the login service with credentials in arguments or the request body. However, repeatedly verifying external credentials might cause performance issue, in which case you should use the login service and the session cookie it returns.

- If your server is configured with SSO (Single Sign-On), use the updated v2 login service to send the token. For more information, see The login Service.
- If your server is configured with Pre-Authentication, specify the pp argument in every API request, as shown in Argument-based Authentication.

None of these authentication methods provide privacy, meaning that passwords are sent in plain text or easily reversed encodings. Jaspersoft recommends that you configure your server and clients to use HTTPS to provide end-to-end privacy and security. Alternatively, JasperReports Server has a login encryption feature that hides passwords. If this feature is enabled on your server, you must encrypt your passwords before sending them in REST requests. For more information, see Login Encryption (Deprecated).

HTTP Basic Authentication

HTTP basic authentication is stateless, meaning that your client application must supply a valid user and password in every API 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 REST API 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. When the organization ID or alias is added to an argument in the URL, you should use the encoded form: jasperadmin%7Corganization_1

When your server implements external authentication, such as using LDAP, you can submit the username and password with basic HTTP authentication as well.

If log in encryption is enabled in your server, then you must encrypt the password before base64-encoding it with the username. For more information about encryption, see Login Encryption (Deprecated).

Argument-based Authentication

Some UTF-8 characters in usernames and passwords are not properly handled by the encoding in HTTP basic authentication, and such requests will return an error. To get around this, all services of the REST API accept arguments for the username and password in the URL. This method also works when your server is configured to check username and passwords with external authentication, for example using LDAP. All services also recognize the pp argument that you use when your server is configured for pre-authentication.

Use the following arguments as an alternate method to send user credentials in a stateless manner with each API request:

Method	**URL**
any	http[s]://<host>:<port>/jasperserver[-pro]/rest_v2/<service/and/path>[?<arguments>]

Argument	**Type/Value**	**Description**
j_username	Text	The user ID. In commercial editions of the server that implement multiple organizations, the argument must specify the organization ID or alias in the following format: userID%7CorgID (%7C is the encoding for the | character).
j_password	Text	The user's password. The argument is optional but authentication fails without the password. If the server has login encryption enabled, the password must be encrypted as explained in Login Encryption (Deprecated).
pp	Text	The token for your pre-authentication mechanism. The default parameter name for a pre authentication token is pp. This parameter name can be changed in the configuration file .../WEB-INF/applicationContext-externalAuth-preAuth.xml.

Method	URL	
userLocale	Java locale string	An optional argument to set the locale for this user. The locale can affect both server strings such as messages and report content if localized by Domains. The server sets a cookie with this value so that it is used in every subsequent request until changed. If this argument is never specified for a given user, the server's default locale is used. Specify a Java locale string such as fr (French) or de (German).
userTime zone	Java time zone	An optional argument to set the time zone for this user. The server sets a cookie with this value so that it is used in every subsequent request until changed. If this argument is never specified for a given user, the server's default time zone is used. The time zone names are those supported by java.time.ZoneID, which are defined in the tz database.

Return Value on Success	Typical Return Values on Failure
The normal response for the requested operation.	401 Unauthorized - Login failed or j_username or j_password was missing, body of response is empty. 403 Forbidden - License expired or otherwise not valid.

For example, the following request returns all repository resources in the Public folder that the sample user `joeuser` has permission to read:

```
http[s]://<host>:<port>/jasperserver[-pro]/rest_v2/resources/Public?j_
username=joeuser%7Corganization_1&j_password=<password>
```

When using pre-authentication on the server, specify only the pp argument, for example (%3D is the encoding for =, and %7C for |):

```
http[s]://<host>:<port>/jasperserver[-pro]/rest_
v2/resources/Public?pp=u%3Djoeuser
%7Cr%3DUSER,SALES%7Co%3DHeadquarters%7Cpa1%3DUSA%7Cpa2%3DLosAngeles
```

Even if you implement HTTPS, you should be aware that plain-text passwords in URLs may appear in your app server's logs, and you should protect such log files. To prevent this security issue, change your logging rules or implement login encryption as described in Login Encryption (Deprecated).

The login Service

The login service allows your client to send user credentials to the server, verify the credentials, and receive a session cookie. By explicitly creating and maintaining a user session, your client can manage the user session and optimize the resources it uses. For more information, see Overview of REST Authentication.

As of JasperReports Server 8.2, reports executions and Input Controls are no longer session dependent. The new behavior is as follows:

1. The user logs in as *joeuser|organization_1*; the user gets a session ID (JSESSIONID in cookies).
2. That user runs a report; report results are stored in the cache.
3. The user does not log out; the session is still alive.
4. From another browser or other client, the user logs in as the same user *joeuser|organization_1*; the user gets a different session ID.
5. If a user with a different session ID tries to access report results from the cache from step 2, the user will be able to get those results because now the service is session independent, it passes through the standard security layer (superuser, tenant admins, users); and if the user has the same name|tenant or tenant admin, the user will be able to access it.

In case at step 3, if the user logs out, then the associated cache is cleared; furthermore, the user at step 5 cannot see the results.

As of JasperReports Server 7.1, the REST v1 login service (rest/login) was deprecated and removed from the API. It is replaced with the similar REST v2 login service (rest_v2/login). This section documents the use of the new rest_v2/login API.

The rest_v2/login service allows REST clients to submit authentication credentials in several ways and receive a server cookie that can be used to identify the user session in subsequent API operations. The supported authentication methods are:

- Login with the username and password in the URL arguments.
- Login with username and password in the request body.
- Login with a ticket for servers configured for Single Sign-On (SSO).

When external authentication such as LDAP is configured in the server, clients are still required to submit the username and password in one of the first two methods above.

Sending passwords in plain text is strongly discouraged, therefore Jaspersoft recommends that you configure your server and clients to use HTTPS, or that you use the login encryption feature. For more information, see Login Encryption (Deprecated).

Method	URL
POST	http[s]://<host>:<port>/jasperserver[-pro]/rest_v2/login[?<arguments>]
GET (config.)	http[s]://<host>:<port>/jasperserver[-pro]/rest_v2/login?<arguments>

Argument	Type/Value	Description
j_username	Text	The user ID. In commercial editions of the server that implement multiple organizations, the argument must specify the organization ID or alias in the following format: j_username%7CorgID (%7C is the encoding for the | character).
j_password	Text	The user's password. The argument is optional but authentication fails without the password. If the server has login encryption enabled, the password must be encrypted as explained in Login Encryption (Deprecated).
ticket	Text	The user's ticket for your SSO mechanism, when enabled. This argument is not valid when j_username and j_password are specified. For example: ticket=ST-40-CZeUUnGPxEqgScNbxh9l-sso-cas.example.com The default parameter name for an SSO authentication token is ticket. This parameter name can be changed in the

Method	URL	
		configuration file WEB-INF/applicationContext-externalAuth-<sso>.xml.
userLocale	Java locale string	An optional argument to set the locale for this user session. The locale can affect both server strings such as messages and report content if localized by Domains. The server sets a cookie with this value so that it is used in every subsequent request until changed. When omitted, the server's default locale is used during this session. Specify a Java locale string such as fr (French) or de (German).
userTime zone	Java time zone	An optional argument to set the time zone for this user. The server sets a cookie with this value so that it is used in every subsequent request until changed. When omitted, the server's default time zone is used during this session. The time zone names are those supported by java.time.ZoneID, which are defined in the tz database.
Content-Type		Content
application/x-www-form-urlencoded		j_username=<userID>[%7C<organizationID>]&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, body of response is empty.	400 Bad Request - Missing j_username or j_password. 401 Unauthorized - Login failed, body of response is empty. 403 Forbidden - License

Method	URL	
		expired or otherwise not valid.

Because browsers submit URLs with the GET method, you can test the login service and test credentials by submitting requests from a web browser. With developer tools in your browser, you can see the server's response, and when successful, the session cookie it contains. Credentials must be passed as arguments in the URL, as shown in the following example:

```
http[s]://<host>:<port>/jasperserver[-pro]/rest_v2/login?j_
username=<userID>[%7C<orgID>]&
j_password=<password>
```

Client applications typically use the POST method, and they gather the session cookie from the response to use in future requests. Credentials can be sent either in the URL arguments, as shown above, or in the content of the request, as shown in the following example:

```
POST /jasperserver/rest_v2/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%7Corganization_1&j_password=jasperadmin
```

When the login is successful, the server sends the "200 OK" response containing a cookie for the session ID of the now-logged-in user:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=52E79BCEE51381DF32637EC69AD698AE;
Path=/jasperserver
Content-Length: 0
Date: Fri, 3 Aug 2018 01:52:48 GMT
```

For optimal performance, the session ID from the cookie should be used to keep the session open. Usually, your REST library will automatically 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
```

By default, the session timeout on the server is 20 minutes of inactivity. Beyond that time, requests using the session cookie fails due to lack of authentication. Your client will need to authenticate again using any of the methods described in this chapter.

Maintaining a session with cookies is not mandatory, and your application can use any combination of session cookie, stateless authentication, or both. However, if you use the session ID, it is good practice to close the session as described in Logout. Closing the session frees up any associated resources in memory.

Login Encryption (Deprecated)

As of release 7.5, the HTTP parameter encryption described in this section is deprecated. This feature is no longer supported because the Javascript libraries it uses are no longer supported. Jaspersoft recommends using TLS (Transport Level Security) to implement HTTPS and secure communication between your users and the server.

JasperReports Server supports 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 request. The client must request the new key and re-encrypt the password before every request using HTTP Basic Authentication including the login service.

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

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

Return Value on Success	Typical Return Values on Failure
200 OK – Body contains a JSON representation of public key: `{ "maxdigits":"131", "e":"10001", "n":"9f8a2dc4baa260a5835fa33ef94c..." }`	200 OK – Body contains {Error: Key generation is off}

```
{
  "maxdigits":"131",
  "e":"10001",
  "n":"9f8a2dc4baa260a5835fa33ef94c..."
}
```

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.

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. The request must include the JSESSIONID cookie, which your REST client libraries should do automatically.

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

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:

- Resource Descriptors for a reference to every type of resource and its attributes.
- The resources Service for methods to operate on resources in the repository.

This chapter includes the following sections:

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.

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 Resource Descriptors.
- <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.

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.

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

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.

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.

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 Referenced Resources 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

Resource Type	Reference Attribute(s)	Expanded Name and Descriptor
	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

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 Nested Resources 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.

Optimistic Locking

The resources service supports optimistic locking on all write and update operations (PUT and POST). 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.

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.

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:

- Working With Resources for general guidelines about using descriptors.
- The resources Service 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:

- Domain Topic
- XML/A Connection
- Mondrian Connection
- Secure Mondrian Connection
- OLAP Unit
- Mondrian XML/A Definition
- Other Types

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

```
{
    "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"
    ...
}
```

application/repository.{resourceType}+xml

```
<?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}>
```

Only the label, description, and permission fields are writable. The other fields are generated by the server.

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

Folder

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

application/repository.folder+json

```
{
    "uri" :"<resourceURI>",
    "label":"Sample Label",
    "description":"Sample
Description",
    "permissionMask":"0",
    "creationDate": "2013-07-
04T12:18:47",
    "updateDate": "2013-07-
04T12:18:47",
    "version":"0"
}
```

application/repository.folder+xml

```
<folder>
    <uri>{resourceURI}</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>
```

Only the label and description fields are writable.

JNDI Data Source

application/repository.jndiJdbcDataSource+json

```
{
    <commonAttributes>,
    "jndiName":"<jndiName>",
    "timezone":"<timezone>"
}
```

application/repository.jndiJdbcDataSource+xml

```
<jndiDataSource>
    {commonAttributes}
    <jndiName>
{jndiName}</jndiName>
    <timezone>
{timezone}</timezone>
</jndiDataSource>
```

JDBC Data Source

application/repository.jdbcDataSource+json

```
{
    <commonAttributes>,
    "driverClass":"<driverClass>",
    "password":"<password>",
    "username":"<username>",

"connectionUrl":"<connectionURL>",
    "timezone":"<timezone>"
}
```

application/repository.jdbcDataSource+xml

```
<jdbcDataSource>
    {commonAttributes}
    <driverClass>
{driverClass}</driverClass>
    <password>
{password}</password>
    <username>
{username}</username>
    <connectionUrl>
         {connectionURL}
    </connectionUrl>
    <timezone>
{timezone}</timezone>
</jdbcDataSource>
```

AWS Data Source

application/repository.awsDataSource+json	application/repository.awsDataSource+xml

```
{
    <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>"
}
```
	```
	<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>
	```

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 .../WEB-INF/application-context.xml	Labels for AWS regions in .../WEB-INF/bundles/jasperserver_messages.properties
`us-east-1.amazonaws.com`	`US East (Northern Virginia) Region`

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

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.

application/repository.virtualDataSource+json

```
{
    <commonAttributes>,
    "subDataSources":[
        {

"id":"<subDataSourceID>",

"uri":"<subDataSourceURI>"
        },
        ...
```

application/repository.virtualDataSource+xml

```
<virtualDataSource>
    {commonAttributes}
    <subDataSources>
        <subDataSource>
            <id>
{subDataSourceID}</id>
            <uri>
{subDataSourceURI}</uri>
        </subDataSource>
        ...
```

application/repository.virtualDataSource+json

```
    ]
}
```

application/repository.virtualDataSource+xml

```
    </subDataSources>
</virtualDataSource>
```

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

```
{
    <commonAttributes>,

"serviceClass":"<serviceClass>",

"dataSourceName":"<dataSourceNam
e>",
    "properties":[
        {
            "key":"<key>",
            "value":"<value>"
        },
        ...
    ]
}
```

application/repository.customDataSource+xml

```
<customDataSource>
    {commonAttributes}
    <serviceClass>
        {serviceClass}
    </serviceClass>
    <dataSourceName>
        {dataSourceName}
    </dataSourceName>
    <properties>
        <property>
            <key>{key}</key>
            <value>{value}</value>
        </property>
        ...
    </properties>
</customDataSource>
```

Bean Data Source

application/repository.beanDataSource+json

```
{
    <commonAttributes>,
    "beanName":"<beanName>",
    "beanMethod":"<beanMethod>"
}
```

application/repository.beanDataSource+xml

```
<beanDataSource>
    {commonAttributes}
    <beanName>{beanName}<beanName>
    <beanMethod>
{beanMethod}</beanMethod>
</beanDataSource>
```

Datatypes

application/repository.dataType+json

```
{
    <commonAttributes>,

"type":"text|number|date|dateTim
e|time",
    "pattern":"<pattern>",
    "maxValue":"<maxValue>",
    "strictMax":"true|false",
    "minValue":"<minValue>",
    "strictMin":"true|false"

"maxLength":"<maxLengthInteger>"
}
```

application/repository.dataType+xml

```
<dataType>
    {commonAttributes}

<type>text|number|date|dateTime|tim
e</type>
    <pattern>{pattern}</pattern>
    <maxValue>{maxValue}</maxValue>

<strictMax>true|false</strictMax>
    <minValue>{minValue}</minValue>

<strictMin>true|false</strictMin>
    <maxLength>
{maxLengthInteger}</maxLength>
</dataType>
```

List of Values

application/repository.listOfValues+json

```
{
    <commonAttributes>,
    "items":[
        {
            "label":"<label>",
            "value":"<value>"
        },
        ...
    ]
}
```

application/repository.listOfValues+xml

```
<listOfValues>
    {commonAttributes}
    <items>
        <item>
            <label>
{label}</label>
            <value>
{value}</value>
        </item>
        ...
    </items>
</listOfValues>
```

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

```
{
    <commonAttributes>,
    "value":"<query>",
    "language":"<language>",
    "dataSource":{
        "dataSourceReference":
{

"uri":"<dataSourceURI>"
        }
    }
}
```

application/repository.query+xml

```
<query>
    {commonAttributes}
    <value>{query}</value>
    <language>
{language}</language>
    <dataSourceReference>
        <uri>
{dataSourceURI}</uri>
    </dataSourceReference>
</query>
```

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

```
{
    <commonAttributes>,
    "mandatory":"true|false",
    "readOnly":"true|false",
    "visible":"true|false",

"type":"<inputControlTypeByteValu
e>",

"usedFields":"<field1;field2;...>",
    "dataType": {
        "dataTypeReference": {
            "uri":
"<dataTypeResourceURI>"
        }
    },
    "listOfValues": {
        "listOfValuesReference": {
            "uri":
"<listOfValuesResourceURI>"
        }
    }
    "visibleColumns":["column1",
"colum2", ...],
    "valueColumn":"<valueColumn>",
    "query": {
        "queryReference": {
            "uri":
"<queryResourceURI>"
        }
    }
}
```

application/repository.inputControl+xml

```
<inputControl>
    {commonAttributes}

<mandatory>true|false</mandator
y>

<readOnly>true|false</readOnly>

<visible>true|false</visible>
    <type>
{inputControlTypeByteValue}</typ
e>
    <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>
```

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	Not used	
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
10	Multi-select list of values check boxes	listOfValues
11	Multi-select query check boxes	query; queryValueColumn

File

The repository.file+<format> descriptor is used to identify the file type.

application/repository.file+json	application/repository.file+xml
`{`	`<file>`

application/repository.file+json	application/repository.file+xml

<commonAttributes>,
 	
"type":"pdf|html|rtf|csv|odt|txt
 	
|docx|ods|xlsx|img|font|jrxml
            |jar|prop|jrtx|xml|css
            |olapMondrianSchema
            |accessGrantSchema
            |unspecified",
 	
"content":"<base64EncodedContent>"
}
	{commonAttributes}
	
	<type>pdf|html|rtf|csv|odt|txt
	
	|docx|ods|xlsx|img|font|jrxml
	        |jar|prop|jrtx|xml|css
	        |olapMondrianSchema
	
	|accessGrantSchema|unspecified
	    </type>
	    <content>
	{base64EncodedContent}</content>
	</file>

The content field is write-only: it is absent when requesting the file resource descriptor and used only when uploading a file resource as base-64 encoded content. For other ways to upload file contents, see Uploading File Resources. To download file contents, see Downloading File Resources.

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 (see Nested 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

The default value for the controlsLayout is popupScreen. The reportRenderingView and the inputControlRenderingView can be left as empty strings (""), while the query can be null.

application/repository.reportUnit+json

```
{
    <commonAttributes>,

"controlsLayout":"<popupScreen|sepa
ratePage
         |topOfPage|inPage>",
```

```

"alwaysPromptControls":"true|fals
e",
    "inputControlRenderingView":

"<inputControlRenderingView>",
    "reportRenderingView":
        "<reportRenderingView>",
    "dataSource":{
        "dataSourceReference": {
            "uri":"<dataSourceURI>"
        }
    },
    "query:" {
        "queryReference": {
            uri:
"<queryResourceURI>"
        }
    },
    "jrxml": {
        "jrxmlFileReference": {
            "uri":
"<jrxmlFileResourceURI>"
        } or
        "jrxmlFile": {
            "type": "jrxml",
            "label": "Main jrxml",
            "content":
"<base64Encoded>
```

application/repository.reportUnit+xml

```
<reportUnit>
    {commonAttributes}

<controlsLayout>popupScreen|sepa
ratePage

|topOfPage|inPage</controlsLayou
t>
```

```

<alwaysPromptControls>true|false
        </alwaysPromptControls>
    <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>
        or <jrxmlFile>
               <type>jrxml</type>
```

application/repository.reportUnit+json	application/repository.reportUnit+xml

}
    },
    "inputControls": [
        {
 	
"inputControlReference": {
                "uri":
"<inputControlURI>"
            }
        },
        ...
    ],
    "resources": {
        "resource": [{
            "name":
"<resourceName>",
            "fileReference": {
                "uri":
"<fileResourceURI>"
            }
        }],
        "resource": [{
            "name": "Logo",
            "file": {
                "fileResource": {
                    "type": "img",
                    "label":
"Logo.png",
 	
"content":"<base64Encoded>"
                }
            }
        }]
        ...
    }
}
	<label>Main
	report</label>
	                <content>
	{base64Encoded}
	                </content>
	            </jrxmlFile>
	    </jrxml>
	    <inputControls>
	        <inputControlReference>
	            <uri>
	{inputControlURI}</uri>
	        </inputControlReference>
	        ...
	    </inputControls>
	    <resources>
	        <resource>
	            <name>
	{resourceName}</name>
	            <fileReference>
	                <uri>
	{fileResourceURI}</uri>
	            </fileReference>
	        </resource>
	        <resource>
	            <name>Logo</name>
	            <file>
	                <type>img</type>
	
	<label>Logo.png</label>
	                <content>
	{base64Encoded}
	                </content>
	            </file>
	        </resource>
	        ...
	    </resources>
	</reportUnit>

Report Options

application/repository.reportOptions+json

```
{
    <commonAttributes>,
    "reportUri":"<reportURI>",
    "reportParameters":[
        {
"name":"<parameterName>",
            "value":[
                "value_1",
                "value_2",
                ...
            ]
        },
        ...
    ]
}
```

application/repository.reportOptions+xml

```
<reportOptions>
    {commonAttributes}
    <reportUri>
{reportURI}</reportUri>
    <reportParameters>
        <reportParameter>
            <name>
{parameterName}</name>
            <value>value_1</value>
            <value>value_2</value>
            ...
        </reportParameter>
        ...
    </reportParameters>
</reportOptions>
```

Domain (semanticLayerDataSource)

For more information about accessing the schema of a Domain, see Working With Domains.

When the locale property is left empty, the default locale bundle is used.

application/repository.semanticLayerDataSource+json

```
{
    <commonAttributes>,
    "dataSource":{
        "dataSourceReference": {
"uri":"<dataSourceURI>"
```

application/repository.semanticLayerDataSource+xml

```
<semanticLayerDataSource>
    {commonAttributes}
    <dataSourceReference>
        <uri>{dataSourceURI}</uri>
    </dataSourceReference>
    <schemaFileReference>
```

application/repository.semanticLayerDataSource+json	application/repository.semanticLayerDataSource+xml

```
    }   },
    "schema": {
        "schemaFileReference": {
            "uri":
"<schemaFileURI>"
    }   },
    "bundles": [{
        "locale":
"<localeString>",
        "file": {
            "fileReference":
{"uri":

"<propertiesFileURI>"
        }   }   },
        ...
    ],
    "securityFile": {
        "securityFileReference": {
            "uri":
"<securityFileURI>"
}   }   }
```
	```
	        <uri>{schemaFileURI}</uri>
	    </schemaFileReference>
	    <bundles>
	        <bundle>
	            <locale>
	{localeString}</locale>
	            <fileReference>
	    <uri>{propertiesFileURI}</uri>
	            </fileReference>
	        </bundle>
	        ...
	    </bundles>
	    <securityFileReference>
	        <uri>
	{securityFileURI}</uri>
	    </securityFileReference>
	</semanticLayerDataSource>
	```

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 Report Unit (JRXML Report)). 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

XML/A Connection

application/repository.xmlaConnection+json

```
{
    <commonAttributes>,
    "url":"<xmlaServiceURL>",

"xmlaDataSource":"<xmlaDataSourc
e>",
    "catalog":"<catalog>",
    "username":"<username>",
    "password":"<password>"
}
```

application/repository.xmlaConnection+xml

```
<xmlaConnection>
    {commonAttributes}
    <url>{xmlaServiceURL}</url>
    <xmlaDataSource>
        {xmlaDataSource}
    </xmlaDataSource>
    <catalog>{catalog}</catalog>
    <username>
{username}</username>
    <password>
{password}</password>
</xmlaConnection>
```

Mondrian Connection

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

application/repository.mondrianConnection+json

```
{
    <commonAttributes>,
    "dataSource":{
        "dataSourceReference": {

"uri":"<dataSourceURI>"
        }
    },
    "schema": {
        "schemaReference": {
```

application/repository.mondrianConnection+xml

```
<mondrianConnection>
    {commonAttributes}
    <dataSourceReference>
        <uri>{dataSourceURI}</uri>
    </dataSourceReference>
    <schemaReference>
        <uri>
{schemaFileResourceURI}</uri>
    </schemaReference>
</mondrianConnection>
```

application/repository.mondrianConnection+json	application/repository.mondrianConnection+xml
`"uri": "<schemaFileResourceURI>" } } }`	

```
            "uri":
"<schemaFileResourceURI>"
        }
    }
}
```

Secure Mondrian Connection

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

application/repository.secureMondrianConnection+json	application/repository.secureMondrianConnection+xml
see JSON block below	see XML block below

```
{
    <commonAttributes>,
    "dataSource":{
        "dataSourceReference": {

"uri":"<dataSourceURI>"
        }
    },
    "schema": {
        "schemaReference": {
            "uri":
"<schemaFileResourceURI>"
        }
    },
    "accessGrantSchemas": [
        {

"accessGrantSchemaReference": {
                "uri":
"<accessGrantSchemaFileResourceUR
I>"
```

```
<secureMondrianConnection>
    {commonAttributes}
    <dataSourceReference>
        <uri>{dataSourceURI}</uri>
    </dataSourceReference>
    <schemaReference>
        <uri>
{schemaFileResourceURI}</uri>
    </schemaReference>
    <accessGrantSchemas>

<accessGrantSchemaReference>
            <uri>
{accessGrantSchemaFileResourceURI
}</uri>

</accessGrantSchemaReference>
    </accessGrantSchemas>
</secureMondrianConnection>
```

application/repository.secureMondrianConnection+json	application/repository.secureMondrianConnection+xml
(see below)	

```
            }
        },
        ...
    ]
}
```

OLAP Unit

application/repository.olapUnit+json	application/repository.olapUnit+xml
(see below)	(see below)

```
{
    <commonAttributes>,
    "mdxQuery":"<mdxQuery>",
    "olapConnection": {

"olapConnectionReference": {

            "uri":
"<olapConnectionReferenceURI>"

        }
    }
}
```

```
<olapUnit>
    {commonAttributes}
    <mdxQuery>{mdxQuery}</mdxQuery>
    <olapConnectionReference>
      <uri>
{olapConnectionReferenceURI}</uri>
    </olapConnectionReference>
</olapUnit>
```

Mondrian XML/A Definition

application/repository.mondrianXmlaDefinition+json	application/repository.mondrianXmlaDefinition+xml

```
{
    <commonAttributes>,
    "catalog":"<catalog>",
    "mondrianConnection": {

"mondrianConnectionReference": {
            "uri":
"<mondrianConnectionResourceURI>"
        }
    }
}
```

```
<mondrianXmlaDefinition>
    {commonAttributes}
    <catalog>{catalog}</catalog>
    <mondrianConnectionReference>
        <uri>
{mondrianConnectionResourceURI}</
uri>
    </mondrianConnectionReference>
</mondrianXmlaDefinition>
```

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 Common Attributes.

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.

The resources Service

The rest_v2/resources service searches the repository and accesses 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 format takes search parameters to find resources, and the other format takes a repository URI to access resource descriptors and file contents.

For further information, see:

- Working With Resources for general guidelines about using descriptors.
- Resource Descriptors for a reference to every type of resource and its attributes.
- Working With File Resources to download and upload file resources.
- Working With Domains 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

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	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources?<arguments>	
Argument	**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 the search should include all subfolders recursively. When omitted, the default behavior is recursive (true).
excludeFolder	String	A folder to exclude from the results, for example excludeFolder=/public.
type	String	Match only resources of the given type. Valid types are listed in Resource Descriptors, 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 the 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.
showHidden	true|false	When set to true, the results include nested

Method	URL	
Items		local resources (in _files) as if they were in the repository. For more information, see Local Resources. By default, hidden items are not shown (false).
favorites	true|false	When it is set to true, the service lists the resources that are added to Favorites. By default favorites is 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		Pagination is enabled by default, and the default limit is 100 results. By default, permissions are applied after raw results, such that the search returns fewer than 100 items but there are more pages of results. If you work with large repositories, you must handle pagination issues. These parameters are described in Paginating Search Results.
Options		
accept: application/json (default) accept: application/xml		
Return Value on Success		
200 OK - The body contains a list of resourceLookup descriptors representing the results of the search.		

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 receive a list of resources for display or further processing quickly.

application/json

```
[
    {
        "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"
    },
    ...
]
```

application/xml

```
<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>
```

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 are 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 the 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 of 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 is up to your client app to use the appropriate strategy and process the results accordingly.

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 pages, all of differing lengths. In this case, you may prefer to use Full Page Pagination.

Arguments to resources for Default Pagination

Argument	Type/Value	Description
limit	integer default is 100	This defines the page size, which is the maximum number of resources to return in each response. However, with default pagination, the response likely had 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 the pagination controls:

Headers in Responses for Default Pagination

Header	Description
Result-	This is the number of results that are contained in the current response. It can be

Headers in Responses for Default Pagination	
Header	**Description**
Count	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 is 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 the 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	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. As described in the previous table, this header only appears on the first response, unless forceTotalCount=true.

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:

Arguments of resources for Full Page Pagination		
Argument	**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

Arguments of resources for Full Page Pagination		
Argument	**Type/Value**	**Description**
		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.
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 that it performed to fill the page with permission-filtered results. 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

Headers in Responses for Full Page Pagination	
Header	**Description**
	indicates that 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.

No Pagination

In certain cases, you can turn off pagination. Use this for the 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:

Arguments to resources for No Pagination		
Argument	**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.
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.

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?<argument>	
Argument	**Type/Value**	**Description**
expanded	true|false	When true, all nested resources are given as full descriptors. The default behavior, false, has all nested resources given as references. For more information, see Local Resources.
Options		
accept: application/json (default) accept: application/xml accept: application/repository.folder+<format> (specifically to view the folder resource)		

Method	URL

Return Value on Success	Typical Return Values on Failure
200 OK - The response indicates 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.

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?<argument>	
Argument	**Type/Value**	**Description**
create Folders	true|false	By default, this is true, and the service will create all parent folders if they do not 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 Resource Descriptors

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 Resource URI.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/resource ?<arguments>	
Argument	**Type/Value**	**Description**
create Folders	true|false	True by default, and the service will create all parent folders if they do not 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 Resource Descriptors

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	400 Bad Request - Mismatch between the content-type and

resource that was just created.	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	Root resource multipart item name: resource Root resource multipart Content-type and corresponding item names: • mondrianConnection • schema - Mondrian schema XML file • secureMondrianConnection • schema - Mondrian schema XML file • accessGrantSchemas.accessGrantSchema [{itemIndex}] - XML file • semanticLayerDataSource • schema - Domain schema XML file • securityFile - XML security file • bundles.bundle[{bundleIndex}] - Properties file for internationalization • reportUnit • jrxml - Report unit JRXML file • files.{fileName} - Report unit attached resource file (for example, 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.

Modifying a Resource

Use the PUT method to overwrite an entire resource. PUT sends the entire descriptor for the resource. Specify the path of the target resource in the URL, and specify a resource of the same type in the descriptor. If you want to replace a resource of a different type, specify the overwrite=true argument. The createFolders argument isn't used for updates because the resource and the folders in its path must exist already.

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. To update a local resource, use the PUT method with the hidden folder _file in the path, and send a complete descriptor for the updated resource. For more information, see Local Resources.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/resource?<arguments>	
Argument	**Type/Value**	**Description**
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.		A well defined descriptor of the specified type and format. See Resource Descriptors.

<resourceType>+xml	
Return Value on Success	**Typical Return Values on Failure**
201 Created - The resource was replaced and the response contains the full descriptor of the updated resource.	400 Bad Request - Mismatch between the content-type and the fields or syntax of the actual descriptor.

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	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources/path/to/folder ?<arguments>	
Argument	**Type/Value**	**Description**
create Folders	true|false	True by default, and the service will create all parent folders if they do not 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.

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 ?<arguments>	
Argument	**Type/Value**	**Description**
create Folders	true|false	True by default, and the service will create all parent folders if they do not 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.

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 always returns 204. If the resource existed, it has been deleted.	204 No Content - The request always returns 204, even if the resource path is invalid.

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}&...

Argument	Type/Value	Description
resourceUri	string	Specifies a resource to delete. You may need to encode the / characters in the URI with %2F. Repeat this parameter to delete multiple resources.

Return Value on Success	Typical Return Values on Failure
204 No Content - The request always returns 204. The resources with valid URIs are deleted.	204 No Content - The request always returns 204. No action is taken for invalid URIs.

Working With File Resources

This chapter includes the following sections:

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.

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

File Types	Corresponding MIME Types
odt	application/vnd.oasis.opendocument.text
txt	text/plain
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

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 contains 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 indicates the MIME type of the binary contents. See MIME Types for File Contents for the list of MIME types that correspond to file resource types.	404 Not Found - The specified resource is not found in the repository.

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?<argument>	
Argument	**Type/Value**	**Description**
create Folders	true|false	True by default, and the service creates all parent folders if they do not 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 File. The contents of the file are base64-encoded in the content 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	The request should include the following parameters: • label - contains the name of the file resource

	• description - contains a description for the resource • type - contains a file type shown in MIME Types for File Contents • 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 the 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 MIME Types for File Contents 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.

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 File. The new contents of the file are base64-encoded in the content 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 MIME Types for File Contents 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.

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

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

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 errorDescriptor 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"
                },
                "items":[
                    {
                        "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>
```

If the Domain metadata service encounters one or more issues, the response includes either a list or an object, depending on the number of errors returned. If a single error is returned, the response includes an object. If multiple errors are returned, it includes a list.

Fetching a Domain Schema

The metadata service returns only the displaying 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>
        <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_security</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 Uploading File Resources.

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 Uploading File Resources.

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

Working With Favorites

The rest_v2/favorites service provides methods that allow you to add the resources to Favorites for quick access, remove the resources from Favorites and see the starred resources in your list of Favorites.

This chapter includes the following sections:

- Adding Resources to Favorites
- Removing Resources from Favorites
- Accessing Resources in Favorites

Adding Resources to Favorites

Use the following method to add the resources to Favorites.

Method	**URL**
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/favorites

Content-Type	**Content**
application/json	A JSON object that contains the list of resource URIs to be added to Favorites.

Return Value on Success	**Typical Return Values on Failure**
201 Created - The request was successful.	400 Bad Request - Invalid request syntax. 403 Forbidden - When the logged-in user does not have permission to access the resource. 404 Not Found - When the resource specified in the request does not exist.

The following is an example of a sample payload.

Sample Request Payload:

```
{
    "favorites":[
     {
        "uri":"/public/audit/datasources/AuditDataSource_1"
     },
     {
        "uri":"/public/audit/datasources/AuditVirtualDataSource_1"
     }
   ]
}
```

Sample Response Payload:

```
{
    "favorites":[
     {
        "uri":"/public/audit/datasources/AuditDataSource_1"
     },
     {
        "uri":"/public/audit/datasources/AuditVirtualDataSource_1"
     }
   ]
}
```

If a user adds a resource to the Favorites and later an admin removes the user's access to resources, the entry stays in the jifavoriteresource table, but the user will not be able to view or remove the resources from the Favorites.

Local resources cannot be added to Favorites.

Removing Resources from Favorites

The following method is used to remove the starred resources from the Favorites.

Method	URL

POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/favorites/delete

Content-Type	Content
application/json	A JSON object that contains the list of resource URIs to be deleted from Favorites.

Return Value on Success	Typical Return Values on Failure
204 No Content - The request was successful.	400 Bad Request - Invalid request syntax. 403 Forbidden - When the logged-in user does not have permission to access the resource. 404 Not Found - When the resource specified in the request does not exist.

Accessing Resources in Favorites

Use the following method to get the list of favorites. By default, favorites is false. For more information, see Searching the Repository.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/resources?favorites=true

Content-Type	Content
application/json	A JSON object that lists the resources added to Favorites.

Return Value on Success	Typical Return Values on Failure
200 OK - The body contains a list of Favorites representing the results of the search.	404 Not Found - When the resource specified in the request does not exist. 204 No content - When the resources added to Favorites are not found.

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

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 do not need to send a resource descriptor to describe the permissions. 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.

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
effective	Boolean	When set to true, the effective permissions are returned.

Permissions	optional	By default, this argument is false and only assigned permissions are returned.
recipientType	String optional	Either user or role. 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>
```

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 the user or role, 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).

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:

```
{
  "permission" :[
    {
    "uri":"/properties",
    "recipient":"role:/ROLE_USER",
    "mask":"1"
    },
    {
    "uri":"/properties",
    "recipient":"role:/ROLE_ADMINISTRATOR",
    "mask":"32"
    }
]}
```

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	A JSON object that describes a set of permissions. Because a single resource is specified in the URL, all permissions apply to the same resource, and the server ignores the uri field in the JSON object.

```
{
  "permission" :[
    {
    "uri":"/foo",
    "recipient":"role:/organization_1/ROLE_
MANAGER",
    "mask":"30"
    },
    {
    "uri":"/bar",
    "recipient":"user:/organization_
1/joeuser",
    "mask":"32"
    }
]}
```

Return Value on Success	Typical Return Values on Failure
200 OK - The request was successful.	400 Bad Request - If a recipient or mask is invalid. 404 Not Found - If the resource in the URL is invalid.

Setting a Single Permission

The POST method accepts a single permission descriptor.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions

Content-Type	Content
application/json	A JSON object that describes a single permission on a single resource, for example: `{ "uri":"/properties", "recipient":"role:/ROLE_USER", "mask":"1" }`

```
{
  "uri":"/properties",
  "recipient":"role:/ROLE_USER",
  "mask":"1"
}
```

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 the user or role, 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:

```
{
  "uri": null,
  "recipient": null,
  "mask":"2"
}
```

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.

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.

Deleting a Single Permission

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

Method	URL

DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/permissions/path/to/resource;recipient=<recipient>

Argument	Type/Value	Description
recipient	string required	The recipient format specifies the user or role, 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.

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

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 (superuser).

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

Export Options	Description
	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.
keyAlias	Specify the alias of the key (for example "productionServerKey") to use when encrypting passwords in the export catalog. The alias must correspond to a custom key in the importing server's keystore. When not specified, the server uses its own import-export key, and unless this key is shared with another server, the catalog may only be imported back into the same server. For a list of available keys, see The keys Service. This key must also be available on the server that imports the catalog. For more information about import and export keys, see the JasperReports Server Security Guide.
scheduledAlerts	A list of report URIs for which all scheduled alerts are exported. If you specify a folder URIs, the scheduled alerts for all reports in the folder, recursively, are exported.

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 or resource types are exported, no dependent resources such as data sources, queries, or files included by reference are exported. For example, you can use this parameter to export a single report. The export catalog created with this parameter will cause 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.
skip-favorite-resources	When specified, the resources added to Favorites are not exported.
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.

Export Parameters	Description
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"]
    }, ...
  ]
}
```

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"
}
```

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.

Canceling an Export Operation

To cancel an 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.

The import Service

Use the rest_v2/import service to upload a catalog as a zip file and import it into the repository with the given options. The service has two forms, depending on whether called from an application or from a web page. This operation is Asynchronous. You must poll the state of the import to make sure it succeeds, or otherwise read an error code and retry the operation with different options.

This chapter includes the following sections:

Launching an Import Operation

Typically, an application uses the rest_v2/import service to upload a catalog zip file as an attachment. Your application can specify import options as URL arguments in the format <argument>=true. Options that are omitted are assumed to be false. To import into root, you must be authenticated as the system admin (superuser), but organization admins (jasperadmin) may import into their organizations or suborganizations.

As of JasperReports Server 7.5, import operations must specify a key to decrypt any passwords in the import catalog. Use either the secret-key or the secretUri parameter. For more information about import and export keys, see the JasperReports Server Security Guide.

The import operation is asynchronous. Your application should poll the status of the operation to determine when it finishes or has an error. In case of an error, you can restart the operation with new options or cancel it. The next sections of this chapter explain how to do this.

It is also possible to invoke the import service from a web page, as explained in Importing from a Web Form.

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 update=true, 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. • skip –The resource with broken dependency is not imported, but the import operation continues. • include – Attempts to import the resource by resolving dependencies with local resources. If unsuccessful, this resource is skipped. • fail - The import operation stops and shows 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 the

		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 other JasperReports Server versions that used themes incompatible with your version.
includeAccess Events?	true	Restores the date, time, and user name of the 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 monitoring events if they are included in the catalog.
includeServer Setting?	true	Imports server settings if they are included in the catalog.
keyAlias	key	Specify the alias of the key (for example "productionServerKey") associated with the import catalog. This is the key that was used to encrypt any passwords in the catalog when it was exported. The alias must correspond to a custom key in the importing server's keystore. When not specified, the server uses its own import-export key. In this case, the catalog must have been exported from this server, unless this key has been shared with another server. For a list of available keys, see The keys Service. For more information about import and export keys, see the JasperReports Server Security Guide.
secret-key secretUri		Deprecated for security reasons. See the Content-Type below.

Content-Type	Content
multipart/form-data	You must send the secret-key or secret-uri as form-data. See Importing from a Web Form: • secret-key: Specify the encryption key in hexadecimal format (for example "0x1c 0x40 0xb9 0xf6 0xe2 0xd3 0xf9 0xd0 0x5a 0xab 0x84 0xe6 0xd4 0xe8 0x5f 0xed") associated with the import catalog. You can obtain the key in hexadecimal format when exporting the catalog from the source server. • secret-uri: Specify the encryption key as the URI of a secure file resource in the repository. This must be the same key used when exporting the catalog from the source server.
application/zip	The catalog file to import. Jaspersoft does not recommend uploading files greater than 2 gigabytes.

Return Value on Success	Typical Return Values on Failure
200 OK - Returns a JSON object that indicates the import has been started. See sections on polling and error messages below.	401 Unauthorized - Import is available only to administrators (superuser or jasperadmin).

The body of the response contains the ID of the import operation needed to check its status:

```
{
    id:"aad78989-dasds32-dasdsd"
    phase: "inprogress",
    message: "Import in progress"
}
```

See the following sections to manage the asynchronous import operation.

Polling the Import Status

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.

As with the initial import request, the body of the response contains the state of the import operation, including its current phase and corresponding message:

```
{
    id:"aad78989-dasds32-dasdsd"
    phase: "inprogress",
    message: "Import in progress"
}
```

The following table describes the possible phases of the import operation:

Import Phase	Description
in progress	Import has begun and is still running.
finished	The import has been completed.
failed	The import had an error and was not completed.
pending	The import cannot run because of an error, but it can be restarted with new options. Pending happens if the import operation stopped with the following error codes: • import.organizations.not.match • import.broken.dependencies

Import Errors

In the case of warnings or errors, the GET method returns a JSON structure that includes an error message and code. Some errors also have parameters given as a list of values, for example a list of resource URIs with broken dependencies.

```
{
    id:"aad78989-dasds32-dasdsd"
    phase: "pending",
    message: "Import is pending",
    error: {
       code: "import.broken.dependencies",
       parameters: [errorParams]
    }
}
```

The following tables list the most common warnings and errors, along with an array of parameters, if any. When there is more than one parameter, its position in the array determines its meaning. Some warnings have more than one form with different numbers of parameters:

Warning Code	Parameters	Description
import.resource.uri.too.long	0=resourceURI	The URI given by the parameter is too long.
import.resource.uri.too.long	0=resourceURI 1=length	The URI given by the first parameter is too long. The second parameter is the maximum length.
import.access.denied	0=resourceURI	Access was denied when trying to import the resource with the given URI.
import.resource.not.found	0=resourceURI	The resource with the given URI cannot be found.
import.resource.different.type.already.exists	0=resourceURI	The target of the given resource URI has a different

Warning Code	Parameters	Description
		type than the one being imported. The resource is not updated.
import.resource.uri.not.valid	0=resourceURI	The resource with the given URI is attached to an organization that is not valid in the target.
import.resource.data.missing	0=resourceURI	The resource with the given URI is missing from the catalog and is skipped.
import.reference.resource.not.found	0=resourceURI	The resource with the given URI has dependent resources that are not in the import catalog.
import.reference.resource.not.found	0=resourceURI 1=dependentURI	The resource with the first URI has a dependent resource with the second URI that is not in the import catalog.

Error Code	Parameters	Description
import.organizations.not.match	0=catalogOrganization 1=targetOrganization	The organization ID contained in the import catalog does not match the target organization. Use the mergeOrganizations option.
import.broken.dependencies	0=resourceURI 1=resourceURI ...	The resources in the list have broken dependencies in the import catalog.

Error Code	Parameters	Description
import.organization.into.root.not.allowed	0=catalogOrganization	You cannot import the organization into the root.
import.root.into.organization.not.allowed	0=targetOrganization	The import catalog contains root resources that cannot be imported into the target organization.
import.failed	0=message	The import failed for the reason in the message.
import.failed.zip.error	none	The server cannot read the zip file.
import.failed.content.error	none	The zip file is not a valid import catalog.

Restarting an Import Operation

When an import is in the pending state, you can try to restart it. To see the import options that led to the pending state, use the GET method with the import ID.

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

Return Value on Success	Typical Return Values on Failure
200 OK - Returns a JSON object that contains the options of the import operation.	404 Not Found – When the specified import ID is not found.

The response contains a JSON structure that lists all options specified for this import operation:

```
{
   "brokenDependencies": "fail",
   "organization" : "organization_1",
   "parameters" : ["role-users", "repository-permissions"]
}
```

Once you know which options blocked the import operation, use the PUT method of the import service to send new options and restart the operation.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/import/<import-id>

Content-Type	Content
application/json	A JSON object that contains the new import options, for example: `{ "brokenDependencies": "include", "organization" : "organization_1", "parameters" : ["role-users", "repository-permissions"] }`

Return Value on Success	Typical Return Values on Failure
200 OK - The body of the response is shown below.	404 Not Found - When the specified import ID is not found.

The body of the response shows the import options that were applied:

```
{
   "brokenDependencies": "include",
   "organization" : "organization_1",
   "parameters" : ["role-users", "repository-permissions"]
}
```

Canceling an Import Operation

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

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/import/<import-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 import ID is not found.

Importing from a Web Form

Alternatively, you can call the import service directly from a web page with form and input tags. Use inputs from the Import options checkbox to submit the import options and a file input to upload the catalog zip file.

Submitting an import catalog through an HTML form is also an Asynchronous operation. However, web pages are not practical for receiving the ID and polling the status of the import operation. Therefore, you are limited in knowing whether the import succeeded as you cannot restart the import service if needed.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/import

Content-Type	Content
multipart/form-data	The form data is sent by the browser when you submit a page with input tags. For example: `form-data; name="file-name",` `form-data; name="include-access-events",`

	`form-data; name="update",`
	`...`

application/zip	The catalog file to import. Jaspersoft does not recommend uploading files greater than 2 gigabytes.

Return Value on Success	Typical Return Values on Failure
200 OK – Returns a JSON object that indicates the import has been started.	401 Unauthorized – Import is available only to administrators (superuser or jasperadmin).

The form data options are similar to the arguments in the other import format. When you select a form data option, the option is set to true. Otherwise, the option is set to false by default.

When an option name is submitted in the form data with any value, then the form data options are considered true for the operation.

If you do not want to enable any of the import options, then do not specify those option names in the import request when submitting the form.

The following table describes the options that you can submit in the import request:

Import Web Form Options	Description
update	When the catalog contains resources with the same path and type as existing resources in the repository, then those in the repository is overwritten. Roles and users in an organization are also overwritten with any in the catalog.
skip-user-update	When update is specified, you can also specify this option to avoid overwriting any user profiles.
merge-organization	In commercial releases with organizations, specify this option if the catalog is exported from one organization and imported into a different one. If this option is not

Import Web Form Options	Description
	specified, and the catalog is sourced from a different organization than the destination, and the import fails.
skip-themes	In commercial releases, specify this option to ignore any themes in the catalog. Otherwise, any themes in the catalog are imported into the repository.
include-access-events	When specified, the timestamps for resource creation and modification of each resource are imported into the repository.
include-audit-events	When this option is specified, any audit event logs in the catalog are imported into the server's audit event logs.
include-alerts	Includes data alert when importing the report.
include-monitoring-events	When this option is specified, any monitoring event logs in the catalog are imported into the server's monitoring event logs.
include-server-settings	When this option is specified, any global server settings in the catalog are imported into the server.

The following HTML example shows how the import service can be invoked from a web page:

```
<form method="post"
      action="http://example.com:8090/jasperserver-pro/rest_v2/import"
      enctype="multipart/form-data">
    Import a catalog file to JasperReports Server:
    <input type="file" name="file-name" required="true"
accept="application/zip">
    <fieldset>
        <legend>Options:</legend>
        <input type="checkbox" name="update">Overwrite resources of the
same name<br>
          <input type="checkbox" name="skip-user-update">But do not
overwrite users<br>
        <input type="checkbox" name="merge-organization">Import into a
different organization<br>
```

```
        <input type="checkbox" name="skip-themes">Do not import
themes<br>
        <input type="checkbox" name="include-access-events">Import
created/modified timestamps<br>
        <input type="checkbox" name="include-audit-events">Import audit
event logs<br>
        <input type="checkbox" name="include-monitoring-events">Import
monitoring event logs<br>
        <input type="checkbox" name="include-server-settings">Import
global server settings<br>
    </fieldset>
    <input type="submit" value="Submit">
</form>
```

The keys Service

The rest_v2/keys service allows you to list the cryptographic keys that have been added to the server's keystore. The keys in the list are identified by their key alias, the keys themselves are not given. The response never includes the server's own keys that it creates at installation time, only custom keys added to keystore by administrators using the js-import or keytool commands.

For more information about cryptographic keys and how to add them to the keystore, see the JasperReports Server Security Guide.

This service requires system administrator priviliges on the server (jasperadmin for Community Project, superuser for Professional Edition).

Method	URL
GET	http://<host>:<port>/jasperserver-pro/rest_v2/keys/

Options	Response
accept:application/json	A JSON object that lists custom keys, for example: `[ {"alias": "myCustomKeyAlias", "algorithm": "AES", "label": "My Custom Key" }, {"alias": "productionServerKey", "algorithm": "AES"}, {"alias": "testServerKey", "algorithm": "RSA"} ]`

```
[
    {"alias": "myCustomKeyAlias", "algorithm":
"AES",
     "label": "My Custom Key" },
    {"alias": "productionServerKey",
"algorithm": "AES"},
    {"alias": "testServerKey", "algorithm":
"RSA"}
]
```

Return Value on Success	Typical Return Values on Failure
200 OK – The response contains the custom keys. 204 No Content – When there are no custom keys.	401 Unauthorized – When system administrator credentials are not provided.

You can use the response to create a list of keys available for import or export operations. When present, use the label to display the keys, otherwise use the alias. When specifying keys in import and export operations, specify them by alias. For more information, see The export Service and The import Service.

The reports Service

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

This chapter includes the following sections:

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 is 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 The reportExecutions Service

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: pdf, html, 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 is used to export a specific page. Passing 0 (zero) as the page parameter is accepted, and a blank report is displayed.
ignore pagination?	Boolean	When set to true, the report is 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. By default, the input values are case sensitive. To change them to case insensitive, set inputControl.handler.values.caseSensitive=false in the jasperserver-pro/WEB-INF/js.config.properties file.
interactive?	Boolean	In the commercial edition of the server, where HighCharts are used in the report, this property determines the generation of the JavaScript required for interaction when exported 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 the value is true, each page of the report is on a separate spreadsheet. If false or omitted, the entire report is on a single spreadsheet. For long reports, set this argument to true, otherwise the report does not fit on a single spreadsheet and causes an error.
report Container Width?	Integer	This property specifies the width of the report container. A report specifying this parameter with integer values receives the current screen size width when the report is run.

baseUrl	String	Specifies the base URL that the reportusesuses 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.
attachments Prefix	attachments	For HTML output, this property specifies the URL path to use for downloading the attachment files (JavaScript and images).

Return Value on Success	**Typical Return Values on Failure**
200 OK – The content is the requested file.	400 Bad Request – When incorrect format is provided in the Get request. 404 Not Found – When the specified report URI is not found in the repository.

The following 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.

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 The reportExecutions Service.

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 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 are started by a user of the same organization or its child organizations.
- A regular user can see every running report, but can cancel only the reports that they initiated.

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 The reportExecutions Service.

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>canceled</status>.

Return Value on Success	Typical Return Values on Failure
200 OK – The content also contains: <status>canceled</status>.	204 No Content – When the specified execution ID is not found on the server, and the response body is empty.

The reportExecutions Service

As described in The reports Service , synchronous report execution blocks the client waiting for the response. Synchronous report execution slows down or uses many threads, each waiting for a report, when:

- Managing large reports that may take minutes to complete.
- Running a large number of reports simultaneously

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 to check the status of the report periodically. This also called polling. When the report is finished, the client downloads the output. Alternatively, the client can check when specific pages are finished and download available pages. 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 been completed.

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

This chapter includes the following sections:

- Stopping Running Reports and Jobs
- Removing a Report Execution

Running a Report Asynchronously

To run a report asynchronously, the reportExecutions service provides a method to specify all the parameters needed to open 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 ReportExecutionRequest 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 ReportExecution 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>
    <reportContainerWidth>900</reportContainerWidth>
    <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>
```

```
{
    "reportUnitUri":"/samples/reports/chartthemes/ChartThemesReport",
    "async":true,
    "interactive":true,
    "pages":"1-5",
    "attachmentsPrefix":"/jrio/rest_v2/reportExecutions/
        {reportExecutionId}/exports/{exportExecutionId}/attachments/",
    "baseUrl":"/jrio",
    "parameters":
    {
        "reportParameter":
        [
            {"name":"chartTheme","value":["aegean"]},
            {"name":"anotherParamName","value":["value 1","value 2"]}
        ]
    }
}
```

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

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.
outputFormat	Required	Specifies the desired output format: pdf, html, xlsx, rtf,

Property	Required or Default	Description
		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, it specifies whether the report should get fresh data by querying the data source. If false, use a previously saved data snapshot (if any). By default, if a saved data snapshot exists for the report it is used when running the report.
saveDataSnapshot	false	When data snapshots are enabled, it specifies whether the data snapshot for the report should be written (or overwritten) with data from the report execution.
interactive	true	In a commercial edition of the server where Highcharts are used in the report, this property determines whether the required JavaScript is generated. It is returned as an attachment when exported 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>

Property	Required or Default	Description
async	false	Determines whether reportExecution is synchronous or asynchronous. When set to true, the response is provided immediately and the client must poll the report status. They can download the results when ready. By default, this property is false and the operation pauses until the report execution is complete. The client must wait but can download the report immediately after the response.
transformerKey	Optional	Advanced property used when requesting a report as a JasperPrint object. This property can specify a JasperReports Library generic print element transformer of the class net.sf.jasperreports.engine.export.GenericElementTransformer. 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: {contextPath}/rest_v2/reportExecutions/{reportExecutionId}/exports/{exportExecutionId}/attachments/ You can specify a different URL path using the placeholders {contextPath}, {reportExecutionId}, and {exportExecutionId}.
baseUrl	String	Specifies the base URL that the report uses 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.
parameters	See example	A list of input control parameters and their values.

Property	Required or Default	Description
		By default, the parameter values are case-sensitive. To change them to case insensitive, set inputControl.handler.values.caseSensitive=false in the jasperserver-pro/WEB-INF/js.config.properties file.
reportContainerWidth	Optional	This property specifies the width of the report container. A report specifying this parameter with integer values receives the current screen size width when the report is run.

When successful, the reply from the server contains the reportExecution descriptor. This descriptor contains the request ID and status needed 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 running (<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>
```

```
{
    "requestId":"9ecf5c6f-b70d-4170-8a3b-b305db4c2253",
    "reportURI":"/samples/reports/chartthemes/ChartThemesReport",
    "status":"queued"
}
```

The value of async in the request determines if the report output is available after receiving response. Your client should implement either synchronous or asynchronous processing of the response depending on the value you set for the async property.

Set the async property

Polling Report Execution

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

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/status/

Options	Sample Return Value
accept: application/xml (default)	`<status>ready</status>`
accept: application/status+xml	`<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>`
accept: application/json	`{ "value": "ready" }`

accept: application/status+json	
	```
	{
	    "value": "failed",
	    "errorDescriptor": {
	        "message": "Input controls validation failure",
	        "errorCode": "input.controls.validation.error",
	        "parameters": ["Specify a valid value for type Integer."]
	    }
	}
	```

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.

Requesting Page Status

When requesting reports asynchronously, you can also poll the status of a specific page during the report execution. The request ID in the URL is the one returned in the reportExecution descriptor.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/pages/pageNumber/status

Options	Sample Response Content

accept: application/status+json	
	```
	{
	    "reportStatus": "ready",
	    "pageTimestamp": "0",
	    "pageFinal": "true"
	```

```
}
```

Return Value on Success	Typical Return Values on Failure
200 OK – The content contains the page status, as shown above.	404 Not Found – When the request ID specified in the request does not exist.

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 ReportExecution 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": "cce63cba-1685-4708-ba4f-b70f277a1cd5",
    "reportURI": "/public/Samples/Reports/AllAccounts",
    "exports": [
        {
        "status": "ready",
        "outputResource": {
            "contentType": "text/html,"
            "output final": true,
            "outputTimestamp": 0
        },
        "id": "db9acf02-8add-4196-9ab5-ce86844bfc2e",
        "attachments": [

              {
                  "contentType": "image/png",
                  "fileName": "img_0_0_0"
               }
        ]
    }
```

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

When exporting a chart report to HTML, the image produced for the chart is a part of HTML, and can be in two formats - JavaScript or SVG:

- When "interactive" is set to *true*, it is embedded as JavaScript in HTML that uses Highcharts js to render the chart.
- When "interactive" is set to *false*, the chart image is embedded as SVG as part of HTML.

When the option *net.sf.jasperreports.force.html.embed.image=false in WEB-INF/classes/jasperreports.properties* in combination with *interactive=false*, this puts the SVG images into attachments instead of HTML.

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

Return Value on Success	Typical Return Values on Failure
200 OK – The content is the main output of the report. The format specified by the contentType property of the outputResource descriptor. For example: text/html	400 Bad Request – When invalid values are provided for export options in the request body. 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. 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 Running a Report Asynchronously.

Method	URL

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

Return Value on Success	Typical Return Values on Failure
200 OK – The content is the attachment in the format specified in the contentType 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

Requesting Report Bookmarks

Some reports have additional meta-information associated with them, such as bookmarks and indexes of report sections or parts. Clients can use this information to create a table of contents for the report. There are links to bookmarks and parts defined in the report in the table of contents. After running a report, you can request this information using the same request ID.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/info

Options	Sample Response Content
accept: application/json accept: application/xml	A structure that contains bookmarks and report parts, as shown below.

Return Value on Success	Typical Return Values on Failure

200 OK – The content contains the report meta-information, as shown below.	404 Not Found – When the request ID specified in the request does not exist.

Example of a request URL:

```
https://localhost:8080/jasperserver[-pro]/rest_
v2/reportExecutions/70b9b169-1c0e-431c-b8bc-a6f49328bc75/info
```

JSON:

```
{
  "bookmarks": {
    "id": "bkmrk_1058907116",
    "type": "bookmarks",
    "bookmarks": [
      {
        "label": "USA shipments",
        "pageIndex": 22,
        "elementAddress": "0",
        "bookmarks": [
          {
            "label": "Albuquerque",
            "pageIndex": 22,
            "elementAddress": "4",
            "bookmarks": null
          },
          {
            "label": "Anchorage",
            "pageIndex": 23,
            "elementAddress": "116",
            "bookmarks": null
          },
          ...
        ]
      }
    ]
  },

  "parts": {
    "id": "parts_533304192",
    "type": "reportparts",
    "parts": [
      {
        "idx": 0,
        "name": "Table of Contents"
      },
      {
        "idx": 3,
        "name": "Overview"
      },
      {
        "idx": 22,
        "name": "USA shipments"
      }
    ]
  }
}
```

Exporting a Report Asynchronously

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

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_ v2/reportExecutions/requestID/exports/

Content-Type	Content

application/xml
 	
application/json
	Send an export descriptor in either XML or JSON format to specify the format and details of your request. For example:
	`<export>`
	`    <outputFormat>html</outputFormat>`
	`    <pages>10-20</pages>`
	
	`<attachmentsPrefix>./images/</attachmentsPrefix>`
	`</export>`

Options

accept: application/xml (default)

accept: application/json

Return Value on Success	Typical Return Values on Failure
200 OK – The content contains an exportExecution 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:

```
{
    "id":"6b7ce8fa-f1d7-4d53-9af6-4569edb05d1b",
    "status":"queued"
}
```

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

Modifying Report Parameters

You can update the report parameters, also known as input controls, through a separate method before running a report execution again. For more operations with input controls, see The inputControls Service.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/parameters

Argument	Type/Value	Description
freshData	true	When data snapshots are enabled, you must set this to true to force the server to get fresh data when you change parameters. This overrides the default value of false, as explained in the table of properties in Running a Report Asynchronously.

Media-Type | **Content**

application/json

```
[
    {
        "name":"someParameterName",
        "value":["value 1", "value 2"]
    },
    {
        "name":"someAnotherParameterName",
        "value":["another value"]
    }
]
```

application/xml

```
<reportParameters>
    <reportParameter name="Country_multi_
select">
        <value>Mexico</value>
    </reportParameter>
     <reportParameter name="Cascading_state_
multi_select">
        <value>Guerrero</value>
        <value>Sinaloa</value>
    </reportParameter>
</reportParameters>
```

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.

Polling Export Execution

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

accept: application/json	`{ "value": "ready" }`

accept: application/status+json	`{`
	`    "value": "failed",`
	`    "errorDescriptor": {`

```
        "message": "Input controls validation
failure",
        "errorCode":
"input.controls.validation.error",
        "parameters": ["Specify a valid value for
type Integer."]
    }
}
```

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/195a65cb-1762-450a-be2b-1196a02bb625/status

When the status is "ready", the client can download the new export output and any attachments as described in Requesting Report Output. For example:

GET http://localhost:8080/jasperserver-pro/rest_v2/reportExecutions/912382875_1366638024956_2/exports/195a65cb-1762-450a-be2b-1196a02bb625/outputResource

GET http://localhost:8080/jasperserver-pro/rest_v2/reportExecutions/912382875_1366638024956_2/exports/195a65cb-1762-450a-be2b-1196a02bb625/images/img_0_46_0

Finding Running Reports and Jobs

The reportExecutions service provides a method to search for reports that are running on the server. It includes asynchronous reports that are still running and those that are finished but in cache and available by their request ID.

The search for reports also includes report jobs triggered by the scheduler, both running and finished but still in the cache.

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

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions?<arguments>	
Argument	**Type/Value**	**Description**
reportURI	Optional String	This string matches the repository URI of the running report, relative to 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 fireTimeTo	Optional Date/Time	For scheduler jobs, the fire time arguments define a range of time. You can check if the current job 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.
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 Requesting Report Execution Details.

For security purposes, the search for running reports 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 from the same or child organization.
- A regular user can see every running report, but can cancel only the reports that he initiated.

Stopping Running Reports and Jobs

To stop a running report and cancel its output, use the PUT method and set the status as cancelled.

Method	**URL**
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/status/

Content-Type	**Content**
application/xml application/json	Send a status descriptor in either XML or JSON format with the value cancelled. For example: XML: <status>cancelled</status> JSON: { "value": "cancelled" }

Options
accept: application/xml (default) accept: application/json

Return Value on Success	Typical Return Values on Failure
200 OK – When the report execution was successfully stopped, the server replies with the same status: XML: <status>cancelled</status> JSON: { "value": "cancelled" } 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.	404 Not Found – When the request ID specified in the request does not exist.

Removing a Report Execution

Deleting a report that has run removes it from the cache and makes its output no longer available. If the report execution is still running, it is stopped automatically then removed.

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID

Return Value on Success	Typical Return Values on Failure
204 No Content – The report execution was successfully removed.	404 Not Found – When the request ID specified in the request does not exist.

Requesting Raw Parameter Values

After returning from the drill-down report, you can restore the input control values applied to the main report. Using the following method, you can request raw parameter values.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_ v2/reportExecutions/requestID/rawParameterValues

Media-Type	Content
application/json	`{` `"someParameterName":["someValue"], "someAnotherParameterName": ["anotherValue1","anotherValue2"] }`

Return Value on Success	Typical Return Values on Failure
200 OK – The content contains the value as shown above.	404 Not Found – When the request ID specified in the request does not exist.

The inputControls Service

The reportExecutions service includes only a simple mechanism for setting input control (parameters) values in reports. The inputControls service provides a complete set of operations for reading and reordering input controls, and for updating input control values. Even though the inputControls service is accessed through a URL that includes rest_v2/reports/<resourceURI> /inputControls, the <resourceURI> can be any of the following resource types that support input controls:

- reportUnit
- reportOption
- adhocDataView

Note:
rest_v2/inputControls is mainly a read-only API (for example, for reading the list of available input controls in a report, getting values of that input control, or selecting certain values from the list of available input controls). Use rest_v2/resources for mapping or updating input controls.

This chapter includes the following sections:

- Listing Input Controls
- Input Control Structure
- Listing Input Control Values
- Changing the Order of Input Controls
- Setting Input Control Values

Listing Input Controls

The following method returns a description of the structure of the input controls for a given resource. The <resourceURI> can be any of the resource types that support input controls (reportUnit, reportOption, adhocDataView).

By default, the inputControls operation returns both the structure and the state of the input controls. The structure of an input control is its name, type, and display characteristics (such as a label). The state of an input control includes both the current value and the list of possible values, if applicable to that type. You can use the structure of each input control to create a UI for your users to enter values. The state of each input control gives you the values to display, such as the values in a dropdown selector.

Some states are small because the input control type is a single text or numeric input, and only the current value is stored. Some states may be quite large if they are a select type (select single or select multiple items) based on a list generated dynamically from your data. For example, a list of customers to select from may contain hundreds or thousands of items. The inputControls operation can take much longer to return on such large input controls that require a query on your datasource. In this case, you can specify the exclude=state argument to list only input control structures first. You can request the input control states separately later.

The inputControls service uses either XML or JSON data structures. If no Accept header is included, the response is XML by default.

Method	**URL**	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/<resourceURI> /inputControls?<argument>	
Argument	**Type/Value**	**Description**
exclude	state	When specifed as exclude=state, the input control objects in the response contain only the structure elements and none of the state elements. Use this argument if your input controls have large lists of values and may affect performance. You can fetch these values in a separate call, usually after displaying the empty input control UI. See Listing Input Control Values.
Options		
accept: application/xml (default) accept: application/json		
Return Value on Success		**Typical Return Values on Failure**

200 OK - The content is a list of XML or JSON objects that describe the structure of all input controls. See examples below. 204 NO CONTENT - The specified <resourceURI> does not have any input controls defined.	404 Not Found - When the specified <resourceURI> is not found in the repository.

The body of the response contains an object defining the structure and optionally the state of the input controls. The following examples show the same input control in both the XML and JSON formats, including values in the state objects:

```
<inputControls>
    <inputControl>
        <description>Country multi select</description>
        <id>Country_multi_select</id>
        <label>Country multi select</label>
        <mandatory>true</mandatory>
        <masterDependencies/>
        <readOnly>false</readOnly>
        <slaveDependencies>
            <controlId>Cascading_name_single_select</controlId>
            <controlId>Cascading_state_multi_select</controlId>
        </slaveDependencies>
        <state>
            <id>Country_multi_select</id>
            <options>
                <option>
                    <label>Canada</label>
                    <selected>false</selected>
                    <value>Canada</value>
                </option>
                <option>
                    <label>Mexico</label>
                    <selected>false</selected>
                    <value>Mexico</value>
                </option>
                <option>
                    <label>USA</label>
                    <selected>true</selected>
                    <value>USA</value>
                </option>
            </options>
            <uri>/adhoc/topics/Cascading_multi_select_topic_
files/Country_multi_select</uri>
        </state>
        <type>multiSelect</type>
        <uri>repo:/adhoc/topics/Cascading_multi_select_topic_
files/Country_multi_select</uri>
        <validationRules>
            <mandatoryValidationRule>
                <errorMessage>This field is mandatory so you must enter
data.</errorMessage>
            </mandatoryValidationRule>
        </validationRules>
        <visible>true</visible>
    </inputControl>
    ...
</inputControls>
```

```
{
    "inputControl": [
        {
            "id": "Country_multi_select",
            "description": "Country multi select",
            "type": "multiSelect",
            "uri": "repo:/adhoc/topics/Cascading_multi_select_topic_
files/Country_multi_select",
            "label": "Country multi select",
            "mandatory": true,
            "readOnly": false,
            "visible": true,
            "masterDependencies": [],
            "slaveDependencies": [
                "Cascading_name_single_select",
                "Cascading_state_multi_select"
            ],
            "validationRules": [
                {
                    "mandatoryValidationRule": {
                        "errorMessage": "This field is mandatory so you
must enter data."
                    }
                }
            ],
            "state": {
                "uri": "/adhoc/topics/Cascading_multi_select_topic_
files/Country_multi_select",
                "id": "Country_multi_select",
                "options": [
                    {
                        "selected": false,
                        "label": "Canada",
                        "value": "Canada"
                    },
                    {
                        "selected": false,
                        "label": "Mexico",
                        "value": "Mexico"
                    },
                    {
                        "selected": true,
                        "label": "USA",
                        "value": "USA"
                    }
                ]
            }
        },
        ...
    ]
}
```

The following example shows two more JSON objects for single value number and date types of input controls. The number data type has limits, in this example 1 ≤ number ≤ 50, that your application should enforce when users input a value. Independently of the input limits, the values of these input controls are used as limits for a comparison filter, for example "store ID that is less than or equal to" or "Opening date after". Note that the type of filter is not reflected in the input control structure other than through a judiciously named label. Your app usually needs to know the structure of a report and the use of its input controls to properly render a UI that reflects the actual filters.

```
{
    "inputControl": [
        {
            "id": "store_id_1",
            "type": "singleValueNumber",
            "uri": "repo:/public/reports/StoreReport_files/store_id_1",
            "label": "Store ID is less than or equal to",
            "mandatory": false,
            "readOnly": false,
            "visible": true,
            "masterDependencies": [],
            "slaveDependencies": [],
            "state": {
                "uri": "/public/reports/StoreReport_files/store_id_1",
                "id": "store_id_1",
                "value": "22"
            },
            "dataType": {
                "type": "number",
                "maxValue": "50",
                "strictMax": false,
                "minValue": "1",
                "strictMin": false
            }
        },
        {
            "id": "first_opened_date_1",
            "type": "singleValueDatetime",
            "uri": "repo:/public/reports/StoreReport_files/first_opened_
date_1",
            "label": "Date opened is greater than",
            "mandatory": false,
            "readOnly": false,
            "visible": true,
            "masterDependencies": [],
            "slaveDependencies": [],
            "validationRules": [
                {
                    "dateTimeFormatValidationRule": {
                        "errorMessage": "Specify a valid date/time
value.",
                        "format": "yyyy-MM-dd'T'HH:mm:ss"
                    }
                }
            ],
            "state": {
                "uri": "/public/reports/StoreReport_files/first_opened_
date_1",
                "id": "first_opened_date_1",
                "value": "1982-01-08T00:00:00"
            },
            "dataType": {
                "type": "datetime",
                "strictMax": false,
                "strictMin": false
            }
        }
    ]
}
```

Input Control Structure

The input control objects shown in the examples above contain the information needed by your application to display the input controls to your users and allow them to make a selection. The main elements are:

- ID and URI to define which input control it is.
- Mandatory, visible, and read-only flags to determine whether users should interact with this input control.
- Display characteristics such as a label and description.
- The type of input control, which also determines how it is displayed and how users interact with it, for example text box, checkboxes, radio buttons, or dropdown list. The type is one of the following values:

bool (checkbox)	singleValue
singleSelect (dropdown)	singleValueText
singleSelectRadio	singleValueNumber
multiSelectCheckbox	singleValueDate
multiSelect (list box)	singleValueDatetime
	singleValueTime

For all of the single-value types in the right-hand column, the structure includes an additional dataType object that defines limits on the data type such as maxValue or strictMax. Your app should interpret these limits and enforce them on the values that users may enter.

The input control structure also includes certain validation rules that depend on the type of input control. The presence of these rules indicates that your client should verify or validate the values it receives from your users. The rules provide messages to display when validation fails. Messages are localized if you have language bundles defined on the server and the authenticated user specifies a locale. In the current release, the following validations are possible:

- mandatoryValidationRule - This input is required (as indicated by "mandatory": true), and your client should ensure that the user enters a value.

  ```
      "mandatoryValidationRule" : {
          "errorMessage" : "This field is mandatory so you must
  enter data."
      }
  ```

- dateTimeFormatValidationRule - This input is a date or time value and your client should ensure that the user enters a valid date or time.

  ```
      "dateTimeFormatValidationRule" : {
          "errorMessage" : "Specify a valid date value.",
          "format" : "yyyy-MM-dd"
      }
  ```

The input control structure also defines cascading dependencies, if any, between the input controls. The cascading dependencies determine whether a change of values in one input control may change the possible values in another.

- masterDependencies - A list of input control IDs that this input control depends on. If one of these dependencies is modified, your application should fetch the new state of this input control.
- slaveDependencies - A list of input control IDs that depend on this input control. If this input control is modified (given a new value by your user), your application should fetch new state values for these dependencies.

The state object of an input control contains the current and possible values for this input control. The state objects are explained in the next section.

Listing Input Control Values

The following method returns only the state objects that define the current values of a resource's input controls. The state object includes the possible values of each input control, and among these values, the one that is currently selected. Your app can use these values to generate input and selection widgets in the UI for each input control.

Use this method if you have already fetched all the input control structures using the inputControls method. The <resourceURI> can be any of the resource types that support input controls (reportUnit, reportOption, adhocDataView).

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/<resourceURI>/inputControls/values?<argument>	
Argument	**Type/Value**	**Description**
freshData	true|false	When freshData=true is specified, the list of values for any selection input controls is refreshed with a database query. When this argument is omitted, its default value is false, and cached values for input controls are returned. Querying the database for thousands of input control list values may impact performance, which is why the server manages a cache of these values.

Options
accept: application/xml (default) accept: application/json

Return Value on Success	Typical Return Values on Failure
200 OK - The content is a list of XML or JSON objects that describe the values of all input controls. See examples below. 204 NO CONTENT - The specified <resourceURI> does not have any input controls defined.	404 Not Found - When the specified <resourceURI> is not found in the repository.

The body of the response contains a list of state objects for all input controls in the given resource. The contents of each state object depend on the type of the input control. Single value types will only have a value that is the current value of the input control. Selection types have a list of options, each with a value and indicator of whether it is currently selected or not.

The following examples show the same state objects in both the XML and JSON formats:

```
<inputControlStateList>
    <inputControlState>
        <id>Country_multi_select</id>
        <options>
```

```
            <option>
                <label>Canada</label>
                <selected>false</selected>
                <value>Canada</value>
            </option>
            <option>
                <label>Mexico</label>
                <selected>false</selected>
                <value>Mexico</value>
            </option>
            <option>
                <label>USA</label>
                <selected>true</selected>
                <value>USA</value>
            </option>
        </options>
        <uri>/adhoc/topics/Cascading_multi_select_topic_files/Country_
multi_select</uri>
    </inputControlState>
    ...
</inputControlStateList>
```

```
{
    "inputControlState": [
        {
            "uri": "/adhoc/topics/Cascading_multi_select_topic_
files/Country_multi_select",
            "id": "Country_multi_select",
            "options": [
                {
                    "selected": false,
                    "label": "Canada",
                    "value": "Canada"
                },
                {
                    "selected": false,
                    "label": "Mexico",
                    "value": "Mexico"
                },
                {
                    "selected": true,
                    "label": "USA",
                    "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~.

The internal structure of the inputControlState object in an inputControls/Values response is the same as that of a state object in an inputControls response.

The following example shows two more JSON inputControlState objects for single value number and date types of input controls.

```
{
    "inputControlState": [
        {
            "uri": "/public/reports/StoreReport_files/store_id_1",
            "id": "store_id_1",
            "value": "22"
        },
        {
            "uri": "/public/reports/StoreReport_files/first_opened_date_
1",
            "id": "first_opened_date_1",
            "value": "1982-01-08T00:00:00"
        }
    ]
}
```

Note that the state objects do not contain the input control type, therefore your app must determine how to read each state object based on the input control structure that it has previously fetched and stored in memory. There are two ways you can match the list of input control values to their previously fetched structure:

- Each state object has the ID and URI of its corresponding input control. The URI of an input control is equivalent to <resourceURI>_files/<inputControlID>. Use the ID or URI of each state object to match the ID or URI of each input control structure in your app.
- Input controls are positional: the order of input controls is determined when creating the resource and saved in the resource. All responses from the inputControls methods, both structure and values, contain the complete list of input controls in the same order.

Changing the Order of Input Controls

You cannot use the inputControls service to modify any input control structures, such as types, labels, visibility, or dependencies, because doing so would break the reports that rely on them. Also, input controls definitions may simply be referenced in a resource and their structure defined in other repository folders. However, you may use the following method to change the order of the input controls.

Changing the order of the input controls is persistent in the parent resource as stored in the repository, but it does not affect the running of a report or their display in a viewer.

Note that if you manage your list of input control structures and states based on the unchanging order of input controls, this operation invalidates your current order in memory. You need to update your list of stored input controls, or use IDs or URIs to match structures and states.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/<resourceURI>/inputControls/

Content-Type	Content
application/xml application/json	An XML or JSON object that lists the full structure, including the state object, of all input controls in the new order. You cannot modify any fields in the input control structures. They must be sent exactly as received from the inputControls service, except for the order of objects in the list.

Options
accept: application/xml (default) accept: application/json

Return Value on Success	Typical Return Values on Failure
200 OK - The content is an XML or JSON object that lists all input control structures, including their state objects. This should be identical to the content that was sent.	403 Forbidden - If any input control structure in the request list does not match its current structure.

	404 Not Found - When the specified <resourceURI> is not found in the repository.

Setting Input Control Values

After your app has fetched all structures and values and created a UI, you can interact with the input controls and set new values. Use the following methods to send the new values and selections to the server. The server performs validation and returns an error if certain conditions are not satisfied. Before sending new values your application should validate user input in the following ways:

- It must prevent certain input, such as accepting values for a read-only input control or making multiple selections in a single-select input control.
- It should enforce constraints, such as ensuring that a mandatory input control is not null or has at least one selection.
- It should also validate values against any input control limits, such as minimum and maximum values.

After sending new values, use the response to update any changes in selection list values. For example, if you change an input control with cascading dependencies, the server responds with the new selection lists for the dependent input controls. After all new values have been set, you can call the reportExecutions service to run the report again.

There are two forms of this operation, one that returns the full input control structures, and the other that returns only the state values.

Method	**URL**	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/<resourceURI>/inputControls?<argument>	
Argument	**Type/Value**	**Description**
freshData	true|false	When freshData=true is specified, the list of values for any selection input controls is refreshed with a database query. When this argument is omitted, its default value is false, and

	cached values for input controls are returned. Querying the database for thousands of input control list values may impact performance, which is why the server manages a cache of these values.

Content-Type	**Content**

application/xml	An XML object that lists the new value for just those input controls that are modified, for example:
	`<reportParameters>`
	`    <reportParameter name="Country_multi_select">`
	`        <value>Mexico</value>`
	`    </reportParameter>`
	`     <reportParameter name="Cascading_state_multi_select">`
	`        <value>Guerrero</value>`
	`        <value>Sinaloa</value>`
	`    </reportParameter>`
	`</reportParameters>`

application/json	A JSON object that lists the new value for just those input controls that are modified. In JSON, the value of every input control is given as an array of string values, even for numbers, single-select controls, or multi-select controls with a single value. This example is equivalent to the XML example above:
	`{`
	`    "Country_multi_select":["Mexico"],`
	`    "Cascading_state_multi_select":["Guerrero", "Sinaloa"]`
	`}`

Options
accept: application/xml (default) accept: application/json

Return Value on Success	Typical Return Values on Failure
200 OK - The content is a list of XML or JSON structure objects that describes all input controls and their values, including the newly sent values and any new list values arising from cascading dependencies.	403 Forbidden - If any input control value in the request list is invalid as determined by its type or limit validation. 404 Not Found - When the specified <resourceURI> is not found in the repository.

When sending the values shown in the table above, the JSON response is a list of input control structures that begins with the following element:

```
{
    "inputControl": [
        {
            "id": "Country_multi_select",
            "description": "Country multi select",
            "type": "multiSelect",
            "uri": "repo:/adhoc/topics/Cascading_multi_select_topic_
files/Country_multi_select",
            "label": "Country multi select",
            "mandatory": true,
            "readOnly": false,
            "visible": true,
            "masterDependencies": [],
            "slaveDependencies": [
                "Cascading_name_single_select",
                "Cascading_state_multi_select"
            ],
            "validationRules": [
                {
                    "mandatoryValidationRule": {
                        "errorMessage": "This field is mandatory so you
must enter data."
                    }
                }
            ],
            "state": {
                "uri": "/adhoc/topics/Cascading_multi_select_topic_
files/Country_multi_select",
                "id": "Country_multi_select",
                "options": [
                    {
                        "selected": false,
                        "label": "Canada",
                        "value": "Canada"
                    },
                    {
                        "selected": true,
                        "label": "Mexico",
                        "value": "Mexico"
                    },
                    {
                        "selected": false,
                        "label": "USA",
                        "value": "USA"
                    }
                ]
            }
        },
        ...
    ]
}
```

In the second form, you send the same content in the request, but the URL includes the IDs of the modified input controls and your requested values.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/<resourceURI> /inputControls/<inputControlid1>;<inputControlid2>;.../values?<argument>

Argument	Type/Value	Description
freshData	true|false	When freshData=true is specified, the list of values for any selection input controls is refreshed with a database query. When this argument is omitted, its default value is false, and cached values for input controls are returned. Querying the database for thousands of input control list values may impact performance, which is why the server manages a cache of these values.

Content-Type	Content
application/xml	An XML object that lists the new value for just those input

controls that are modified, for example:

```
<reportParameters>
    <reportParameter name="Country_multi_
select">
        <value>Mexico</value>
    </reportParameter>
     <reportParameter name="Cascading_state_
multi_select">
        <value>Guerrero</value>
        <value>Sinaloa</value>
    </reportParameter>
</reportParameters>
```

application/json

A JSON object that lists the new value for just those input controls that are modified. In JSON, the value of every input control is given as an array of string values, even for numbers, single-select controls, or multi-select controls with a single value. This example is equivalent to the XML example above:

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

Options

accept: application/xml (default)
accept: application/json

Return Value on Success	**Typical Return Values on Failure**
200 OK - The content is a list of XML or JSON state objects of all input control values, including the newly sent values and any new list values arising from cascading dependencies.	403 Forbidden - f any input control value in the request list is invalid as determined by its type or limit validation. 404 Not Found - When the specified

	<resourceURI> is not found in the repository.

When sending the values shown in the table above, the JSON response is a list of state objects that begins with the following element:

```
{
    "inputControlState": [
        {
            "uri": "/adhoc/topics/Cascading_multi_select_topic_
files/Country_multi_select",
            "id": "Country_multi_select",
            "options": [
                {
                    "selected": false,
                    "label": "Canada",
                    "value": "Canada"
                },
                {
                    "selected": true,
                    "label": "Mexico",
                    "value": "Mexico"
                },
                {
                    "selected": false,
                    "label": "USA",
                    "value": "USA"
                }
            ]
        },
        ...
    ]
}
```

Getting Total Available Values

This method is used to get the total available values and limit the count of the total available values retrieved when input controls are paginated.

Provide inputs for the following fields in the request body:

- Offset - Specify the values to be returned for each input control.
- Limit - Specify the total number of values to be returned for each input control.

- Criteria - Specify the texts or words to be returned for each input control. If specified, then it searches not only for the complete text, but also for the values provided in the texts for the specified criteria. For example, specifying "ry" in the criteria field returns all the values containing "ry" texts.
- Select - This field is newly added. Specify either "selectedValues" or "allValues" in this field.
    - If "selectedValues" is chosen, then the returned available values have the default-selected values.
    - If "allValues" is chosen, then the returned available values have all the values selected.
    - If any other value other than "selectedValues" or "allValues" is provided, then this field is ignored.

If the "values" and "select" field both are provided in the request, then the "select" field with valid values is considered over the "values" field.

Pass the following arguments to get the total available values of the input control based on selected offset, limit, criteria, and select fields.

Method	**URL**
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/{reportUnitURI}/inputControls/values/pagination

Argument	**Type/Value**	**Description**
freshData	true|false	When freshData=true is specified, then the total count of the list of values for any selected input control is refreshed with the database query. When freshData=false is specified, then the total cached values for input controls are returned. The default value is false.
includeTotalCount	true|false	When includeTotalCount=true is specified, each inputControlState includes the total count of all the values and not the total number of paginated values.

	When includeTotalCount=false is specified, then inputControlStatedoes not include any value in the response. The default value is true.

Content-Type	Content
application/xml	An XML object that lists the total available values for input control based on modified offset, limit, criteria, and select fields. For example:

```
<reportParameters>
  <reportParameter name="Country_multi_
select">
    <limit>2</limit>
    <offset>1</offset>
    <select>selectedValues</select>
  </reportParameter>
  <reportParameter name="Cascading_name_
single_select">
    <criteria>Adams
Construction</criteria>
    <select>allValues</select>
  </reportParameter>
  <reportParameter name="Cascading_
state_multi_select">
    <value>CA</value>
    </reportParameter>
</reportParameters>
```

application/json	A JSON object that lists the total available values for input control based on modified offset, limit, criteria, and select fields. For example:

```
{
 "reportParameter": [
   {
     "name": "Country_multi_select",
     "limit": 2,
     "offset": 1,
     "select": "selectedValues"
    },
   {
     "name": "Cascading_name_single_
select",
     "criteria": "Adams Construction",
     "select": "allValues"
    },
   {
     "name": "Cascading_state_multi_
select",
     "value": [
     "CA"
    ]
   }
 ]
}
```

Options	
accept: application/xml (default) accept: application/json	
Return Value on Success	**Typical Return Values on Failure**
200 OK–The content is a list of XML or JSON state objects of total available values of input control.	500 Unexpected Error–An unexpected error occurs while passing any incorrect value. 400 Serialization Error–An error occurs determined by its type. This issue occurs on sending a malformed or wrong XML/JSON object in the request. For example, missing an open/closed tag or a completely wrong

	object representation that does not match an example object. 400 Out of range Error–An error occurs when the limit or offset for the input control is specified out of range.

When sending the values shown in the table above, the JSON response has the total available values but the number of values are limited based on the offset, limit, and search criteria.

```
{
  "inputControlState": [
    {
      "uri": "/organizations/organization_1/adhoc/topics/Cascading_
multi_select_topic_files/Country_multi_select",
      "id": "Country_multi_select",
      "totalCount": "4",
      "options": [
       {
         "selected": false,
         "label": "Mexico",
         "value": "Mexico"
       },
       {
         "selected": true,
         "label": "USA",
         "value": "USA"
       }
      ]
    },
    {
       "uri": "/organizations/organization_1/adhoc/topics/Cascading_
multi_select_topic_files/Cascading_state_multi_select",
       "id": "Cascading_state_multi_select",
       "totalCount": "3",
       "options": [
         {
           "selected": true,
           "label": "USA | CA",
           "value": "CA"
```

```
          },
          {
            "selected": false,
            "label": "USA | OR",
            "value": "OR"
          },
          {
           "selected": false,
           "label": "USA | WA",
           "value": "WA"
          }
        ]
      },
      {
        "uri": "/organizations/organization_1/adhoc/topics/Cascading_
multi_select_topic_files/Cascading_name_single_select",
        "id": "Cascading_name_single_select",
        "totalCount": "2",
        "options": [
           {
             "selected": true,
             "label": "Steelman-Adams Construction Partners",
             "value": "Steelman-Adams Construction Partners"
           },
           {
             "selected": true,
             "label": "Winter-Adams Construction Associates",
             "value": "Winter-Adams Construction Associates"
           }
         ]
       }
    ]
}
```

Getting Default Selected Values

This method is used to get the default selected values when the input controls are paginated. For example, when a Report, Scheduler, or Dashboard is initially loaded, the input controls API excludes only the values defined in the "state" to avoid the values to be included in the response.

Pass the following arguments to get only the default-selected values of the input control field in the response.

Method	URL	
GET	../rest_v2/reports/{reportUnitURI}/inputControls/selectedValues	
Argument	**Type/Value**	**Description**
freshData	true|false	When freshData=true is specified, the total count of the list of selected values of input controls is refreshed with a database query. When freshData=false is specified, the total count of cached values for input controls are returned. The default value is false.
withLabel	true|false	When withLabel = true is specified, the returned options field for each input control has a label field along with the value. In this process, an additional overhead is required to fetch labels from the DB/cache. When withLabel = false is specified, the returned options field for each input control does not include any label field. In this process, values are the default values and they are fetched from JRXML. The default value is true.

Options
accept: application/xml (default) accept: application/json

Return Value on Success	Typical Return Values on Failure
200 OK–The content is a list of XML or JSON state objects of default-selected values of input control.	500 Unexpected Error–An unexpected error that has occurred while passing any value.

	400 Serialization Error–An error occurs determined by its type. This issue occurs on sending a malformed or wrong XML/JSON object in the request. For example, missing an open/closed tag or a completely wrong object representation that does not match an example object. 400 Out of range Error–An error occurs when the limit or offset for the input control is specified out of range.

The following example shows a simple use of the `select` field in JSON:

```
{

    "selectedValue": [

        {
            "id": "Country_multi_select",

            "options": [
                {
                    "label": "USA",
                    "value": "USA"
                }
            ]
        },

        {
            "id": "Cascading_state_multi_select",

            "options": [

               {
                    "label": "USA | CA",

                    "value": "CA"
                }
            ]
        },

        {
            "id": "Cascading_name_single_select",

            "options": [
                {
                    "label": "A & U Stalker Telecommunications, Inc",
                    "value": "A & U Stalker Telecommunications, Inc"
                }
            ]
        }
    ]
}
```

The following example shows a simple use of the `select` field in XML:

```
<selectedValues>

    <selectedValue id="Country_multi_select">
        <options>
            <label>USA</label>

            <value>USA</value>

        </options>
    </selectedValue>

    <selectedValue id="Cascading_state_multi_select">
        <options>
            <label>USA | CA</label>
            <value>CA</value>
        </options>

    </selectedValue>

    <selectedValue id="Cascading_name_single_select">

        <options>
            <label>A & U Stalker Telecommunications, Inc</label>
            <value>A & U Stalker Telecommunications, Inc</value>

        </options>
    </selectedValue>
</selectedValues>
```

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.

A report option contains input control values that you can read and modify with the inputControls service. Therefore, you should use the methods of the options service to create and list report option resources in the repository, and use the methods of the inputControls service to view and modify the values contained in a report option. For more information, see The inputControls Service.

This chapter includes the following sections:

- Listing Report Options
- Creating Report Options
- Updating Report Options
- Deleting Report Options

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"
  }]
}
```

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 the 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"
}
```

Updating Report Options

Use the following method to modify the values in a given report option. You can also use the methods of the inputControls service to view and modify the values contained in a report option. For more information, see The inputControls Service.

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 the 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 Creating Report Options 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"]
}
```

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.

The jobs Service

The rest_v2/jobs service provides the interface to schedule reports and dashboards and manage those schedules. When a user schedules a report or dashboard to run at a given time and with a given recurrence, the server stores this information in a job. There can be any number of jobs for any number of reports and dashboards, all created by different users. As with the server's user interface, the jobs service only manages jobs that are active, which means they have a trigger that is still pending in the future. Jobs that have finished and are no longer active are never listed or returned.

Within the job, the trigger determines when it will run. A trigger is said to fire when it reaches its scheduled time, and when it does, the server generates the output associated with the job. Various other properties define the filename of the output and where it is stored, emailed, or both. For example, you can define a job to run every Monday at 6 AM, except on company holidays, to generate a report in both PDF and Excel output and have the files written to the repository with sequential filenames, with links to the output emailed to a list of recipients, and any errors emailed to administrators.

The jobs service also uses exclusion calendars that can be defined in The jobs calendars Services.

This chapter includes the following sections:

Searching for Jobs

The GET method of the jobs service has multiple arguments to search for jobs in several ways. The credentials used for authentication determine the scope of the search: an ordinary user can search all of his or her own jobs, an organization admin (jasperadmin) can search all jobs in his or her organization, and the system admin (superuser) can search all jobs on the server. This method only returns active jobs that still have a pending trigger or are still running after their last trigger.

When used without any arguments, this method returns all scheduled jobs for any scheduled report, report option, or dashboard within the scope of the user. The various arguments let you search for jobs on a specific report or dashboard, or find all jobs created by a given user (with administrator credentials). You can also do a string search on the label, or do an advanced search such as by output type or email address. If you want to handle large numbers of results, you can 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 property 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	Optional URI (repository path) of a report, report option, or dashboard to list all of its jobs. You may need to encode the / characters in the URI with %2F. When specified, the results are only for the given resource.
example?	JSON jobModel	Searches for jobs that match the JSON jobModel,

		which is a fragment of a job descriptor containing one or more properties to be matched. This argument lets you search for any parameter of the job descriptor, for example the trigger, exclusion calendar, output format, recipient email, or FTP output. Because the JSON fragment appears as an argument in the URL, it must be properly URL-encoded (percent-encoded) as shown in the example below.
limit?	integer	Specifies the number of jobsummary descriptors in the results. The default is -1 for no limit and thus all results are returned. When used with the offset parameter, it can be used to implement pagination of results.
offset?	integer	Specifies the index of the first jobsummary to be returned. When used with the limit parameter, it can be used to implement pagination.
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. This has no effect when sortType is not specified.

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, as shown in the examples below.	204 No Content - When no matching job is found in the server. After a job finishes running its last triggered instance, it no longer shows up in search results.

For example, if your application wants to request all the jobs for a given report, it would send the following URL (%2F is the / character):

```
GET http://example.com:8090/jasperserver-pro/rest_
v2/jobs?reportUnitURI=%2Freports%2FAllAccounts
```

In the response from the server, the jobs are described in a jobsummary element such as the following example. The jobsummary contains a small subset of the properties of the job descriptor as well as the complete state descriptor:

```
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": "2022-03-15T00:00:00+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>2022-03-15T00:00:00+03:00</nextFireTime>
                    <value>NORMAL</value>
                </state>
                <owner>jasperadmin|organization_1</owner>
                <version>0</version>
            </jobsummary>
            ...
        </jobs>
```

The example parameter lets you specify a search of any property in the job descriptor, such as output formats. You can specify any property in the job descriptor or in any of its nested structures. Some properties 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 property 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
```

Viewing a Job Definition

After searching and finding the ID of a job that is still active, 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. After a job finishes running its last triggered instance, it is no longer active and its ID will return this error.

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. All properties are included, many of which may be null if not set for the chosen job. For more information, see The job Descriptor.

JSON:

```
{
    "id": 1906,
    "version": 0,
    "username": "jasperadmin|organization_1",
    "label": "Daily Report",
    "description": "Sample desctiption",
    "creationDate": "2019-04-21T14:52:04.955+03:00",
    "outputFormats": {
        "outputFormat": ["XLS",
        "PDF"]
    }

    "trigger": {
        "simpleTrigger": {
            "id": 0,
            "version": 0,
            "timezone": "America/Los_Angeles",
            "calendarName": null,
            "startType": 2,
            "startDate": "2019-04-21 10:00",
            "endDate": null,
            "misfireInstruction": 0,
            "occurrenceCount": 1,
            "recurrenceInterval": 1,
            "recurrenceIntervalUnit": "DAY"
        }
    },
    "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": "ftps",
            "protocol": null,
            "port": 990,
            "implicit": true,
            "pbsz": 0,
            "prot": null
        }
    },
}
```

XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<clientJob>
    <creationDate>2019-04-21T13:38:09.759+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>2019-04-21 10:00</startDate>
        <startType>2</startType>
        <timezone>Europe/Helsinki</timezone>
        <version>0</version>
        <occurrenceCount>1</occurrenceCount>
        <recurrenceInterval>1</recurrenceInterval>
        <recurrenceIntervalUnit>DAY</recurrenceIntervalUnit>
    </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_topic<
         /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</usingDefaultReportOutputFolder
URI>
        <version>-1</version>
    </repositoryDestination>
</clientJob>
```

The job Descriptor

The job descriptor is a complex data object with nested containers for the various properties that define a scheduled job. The job descriptor is both the output of the GET method to view job details and, with slightly different content, the input of the PUT and POST methods to create and modify jobs.

The properties of the job descriptor are defined in the following sections:

- General Properties of a Job, such as label and description, but also the output formats and base filename.
- Source and Input Controls includes the repository URL of the report, report option, or dashboard, and any input controls.
- Simple Trigger defines interval-based repetition of the job for a given number of occurrences.
- Calendar Trigger runs at specific time on specific days of the week or days of the month.
- Job Output Properties define the file name and locations where output files are written.
- FTP Output defines whether the output files are written to a remote server.
- Job Output Email defines the recipients for successful output files.
- Job Status Email defines the recipients for success or error messages.

When submitting a job descriptor to create or modify a job schedule, not all properties are needed. In the following tables, each property is one of the following:

- Required - This property must have a value for input to define a valid job.
- Optional - This property may be omitted on input, either because it is nullable, or because the server assigns a default value. The behavior is explained in the property description.

- Conditional - The property may be required or optional depending on other property values. The behavior is explained in the property description.
- Ignored - This property is for internal usage or output only, and the server ignores any value on input. Good practice is to omit these properties from your input.

General Properties of a Job

A valid job descriptor contains the following properties:

Property (as input)	**Description**
creationDate (ignored)	The time and date that the job was first created, managed internally by the server.
id (ignored)	The ID of the job assigned by the server. Use this ID to read, modify, or delete the job.
version (ignored)	An internal version number that has no outside use.
username (ignored)	The owner and creator of this job, including the organization name. Note that only administrators can view other users' jobs.
label (required)	A name for the job, like the label of a resource in the repository.
description (optional)	A description string for the job, which often describes the trigger and chosen output.
exportType (optional)	The default value is DEFAULT, and for scheduling a report no other value is accepted. In a schedule for a dashboard, DEFAULT indicates that the output will be a screenshot in the chosen format, for example an image in a Word file. When this is set to DASHBOARD_DETAILED for a dashboard schedule, the dashlets are exported as individual reports sequentially into the same file. Not all output formats are available depending on the value of exportType (see next property).
outputFormats	Contains a list of output formats, each to be generated in a separate

Property (as input)	Description
outputFormat (required)	output file when the job is triggered. The possible values depend on the resource and the exportType above: • Report (exportType="DEFAULT" is required): CSV, DATA_CSV, DOCX, HTML, ODS, ODT, PDF, PPTX, RTF, XLSX, XLSX_NOPAG, DATA_XLSX • Dashboard with exportType="DEFAULT": DOCX, ODT, PDF, PNG, PPTX • Dashboard with exportType="DASHBOARD_DETAILED": CSV, DOCX, ODS, ODT, PDF, PPTX, RTF, XLSX The XLS and XLS_NOPAG (Excel 2003) output formats are deprecated. You can use XLSX and XLSX_NOPAG (Excel 2016) instead. Older jobs that are still running a request for XLS or XLS_NOPAG outputs, generate output in the XLSX or XLSX_NOPAG format.
outputTimeZone (optional)	The time zone to use when running the report or dashboard, for example "America/Los_Angeles". By default, the scheduler runs the report or dashboard using the server's default time zone. The time zone names are those supported by java.time.ZoneID, which are defined in the tz database. The trigger time zone and the output time zone have different purposes, even if they are often set to the same value. The trigger time zone determines when the job will run, whereas the output time zone affects anytime or date calculations in the report or dashboard. The time zone specified in a data source also affects times and dates in the output.
outputLocale (optional)	The locale to use when running the report or dashboard, if it is based on a domain with locales defined. By default, the scheduler runs the report or dashboard using the server's default locale. If the report or dashboard is not based on a domain with locales, this value has no effect. Use a Java locale string that is also defined in your domain, for example fr (French) or fr_CA (French from Canada).
baseOutput Filename	The basename of the file for the generated report output. Each output

Property (as input)	Description
(required)	format appends its corresponding file extension to this name. This name may also have a time stamp appended as specified in Job Output Properties. This final output name is then used in all locations where the file is saved: repository, file system, FTP, and email attachment.
source (required)	A container for the properties that define the repository URI of the report, report option, or dashboard and its input controls. See Source and Input Controls.
trigger (required)	A container for one of the triggers specified in Simple Trigger or Calendar Trigger. The trigger determines how often the job runs and the date and time at which it runs.
repository Destination (optional)	A container for properties that define the folders and filenames for the output files that are generated each time a scheduled job runs successfully. Its properties are defined in Job Output Properties and the optional FTP Output.
mailNotification (optional)	A container for properties that define email recipients when a job runs successfully. You can customize the contents of the email and whether the report output files are sent as attachments or links. For more information, see Job Output Email.
alert (optional)	A container for properties that define email recipients for job success and errors. For more information, see Job Status Email.

Source and Input Controls

The source is the report, report option, or dashboard being scheduled by the job, along with any required input controls.

Property (as input)	Description
source (required)	A container for the following properties.
reportUnitURI (required)	A mandatory property that defines the repository URI of the report, report option, or dashboard to run for this job. The repository URI is relative to the scope of the user credentials used to authenticate the REST call.
referenceHeight (optional)	When scheduling a dashboard, the default output is a screenshot and these properties specify the dimensions in pixels of the dashboard canvas. The images in the output have these dimensions, even inside another format such as PDF. The default size is 1280 width by 800 height.
referenceWidth (optional)	
parameters parameterValues (conditional)	A container for the input controls (filters) for this job. Every required input control in the report or dashboard must have an XML entry or JSON list item within parameterValues. If there are no input controls, this property can be omitted. See the examples in Creating a Job for the syntax of these properties in JSON and XML.

Simple Trigger

The following properties define the recurrence pattern for a simple trigger.

Property (as input)	Description
simpleTrigger (conditional)	A container for the following properties, itself contained in a required trigger. For input, you must specify either the simpleTrigger or the calendarTrigger.
id (ignored)	An internal ID of the simple trigger that has no outside use.
version (ignored)	An internal version number that has no outside use.

Property (as input)	Description
timezone (optional)	The timezone for all dates and times used by the trigger, for example "America/Los_Angeles". By default, the trigger uses the same time zone as the server. The time zone names are those supported by java.time.ZoneID, which are defined in the tz database.
startType (optional)	Determines when the job becomes active and sets the base time at which recurrence starts. Supported values: START_TYPE_NOW - The job starts immediately, and the trigger fires right away. This is the default value when this property is omitted. START_TYPE_SCHEDULE - The job starts at the startDate, at which time it fires.
startDate (conditional)	The date and time at which the job will start, required when startType=START_TYPE_SCHEDULE. The simple trigger fires at this time, and it begins its recurrence at this time. The format is "yyyy-MM-dd HH:mm" and the timezone property is applied.
endDate (optional)	The date and time at which the job will stop. The trigger will not fire after this time, even if any occurrences remain, unless a misfire occurs and the misfire policy allows it. The format is "yyyy-MM-dd HH:mm" and the timezone property is applied.
occurrenceCount (required)	An integer that defines how many times the trigger fires, provided the recurrence intervals happen before the endDate.
recurrence Interval (required)	The time interval between firings of the trigger, with the interval unit provided in the next property.
recurrence IntervalUnit (required)	The unit of time for the recurrence interval. Supported values: MINUTE, HOUR, DAY, or WEEK. For units greater than MINUTE, the startType and startDate determine the basis for recurrence. For example, if the trigger fires immediately and recurs every 2 days, then it fires at the current time on the subsequent days.

Property (as input)	Description
calendarName (optional)	The name of a previously defined exclusion calendar. An exclusion calendar defines a set of dates or times when the job does not run, for example a list of holidays. You can update the exclusion calendar without changing the job. For information about creating and modifying exclusion calendars, see The jobs calendars Services.
misfire Instruction (optional)	An integer value that defines the behavior if the trigger did not fire when scheduled. The values of misfireInstruction are described in the next table. A misfire occurs if JasperReports Server or its Quartz scheduler component is offline when a trigger was supposed to happen and run a job. It can also occur if all threads of the scheduler are busy and the job cannot run when the trigger should fire. When the scheduler restarts or a thread becomes available again, it checks for any triggers that did not fire on time and takes action based on the misfireInstruction. The misfire instruction does not apply if the trigger fires normally but the report encounters an error. The default value is 0 when this property is omitted.

Choose a misfire policy based on how frequently your job runs and how critical it is. For example, an outage may last one to two hours, and if a daily report is critical, you may want it to run as soon as the scheduler is able. However, if a report runs every hour, you may want to ignore missed reports and wait for the next report at the scheduled time. Note that different policies may have the same effect depending on how the trigger is defined, but also the same policy may have different effects on different trigger types.

misfireInstruction	Description for Simple Triggers
0 (default)	No instruction (same behavior as option -1 below). Does not trigger the job that misfired, and takes no action. Because the trigger did not fire, the number of occurrences is not decremented. This is the default behavior if no misfireInstruction is defined. The trigger will fire the next time according to the recurrence value and unit, as if the misfire had fired at the proper time.
-1	Ignore misfire policy. Instructs the scheduler that the trigger will never be evaluated for a misfire situation, and that the scheduler will

misfireInstruction	Description for Simple Triggers
	simply try to fire it as soon as it can, and then update the trigger as if it had fired at the proper time. This value has the same effect as 0: take no action and do not change the number of occurrences.
-999	Called the smart policy. Instructs the scheduler that on a misfire situation, the custom updateAfterMisfire method is called on the trigger to determine the misfire action. In this case, you must define and enable a custom trigger class on the server(this is beyond the scope of this documentation).
1	Run now: Instructs the scheduler to trigger now, which is the time the misfire is detected. If the outage covers several trigger times, they will each have a misfire, and with this value, they will each run now.
2	Instructs the scheduler that on a misfire situation, the SimpleTrigger is rescheduled to 'now' (even if the associated calendar excludes 'now') with the repeat count left as-is. This does obey the trigger end-time, so if 'now' is after the end-time, the trigger will not fire.
3	Instructs the scheduler that on a misfire situation, the SimpleTrigger is rescheduled to 'now' (even if the associated Calendar excludes 'now') with the repeat count set to what it would be, if it had not missed any firings. This does obey the trigger end-time, so if 'now' is after the end-time the trigger will not fire.
4	Ignores any missed firing of the trigger (no action is taken for the missed firing), but the repeat count is set to what it would be if the trigger had fired normally. The trigger will fire at the next scheduled time after the current time, taking into account any associated exclusion calendar or end time. The effect is that missed trigger occurrences will be skipped.
5	Ignores any missed firing of the trigger (no action is taken for the missed firing), but the repeat count is left unchanged. The trigger will fire at the next scheduled time after the current time, taking into account any associated exclusion calendar or end time. The effect is that missed trigger occurrences happen later, past the last expected occurrence.

Calendar Trigger

A calendar trigger lets you schedule a job to run multiple times based on any combination of time and date. Its properties let you define single values, ranges, or wild-cards for minutes, hours, days, weeks, or months. For example, you can run a report every 15 minutes from 10 AM to noon every Monday.

The following properties define the recurrence pattern of a calendar trigger:

Property (as input)	**Description**
calendarTrigger (conditional)	A container for the following properties, itself contained in a required trigger. For input, you must specify either the simpleTrigger or the calendarTrigger.
id (ignored)	An internal ID of the calendar trigger that has no outside use.
version (ignored)	An internal version number that has no outside use.
timezone (optional)	The timezone for all dates and times used by the trigger, for example "America/Los_Angeles". By default, the trigger uses the same time zone as the server. The time zone names are those supported by java.time.ZoneID, which are defined in the tz database.
startType (optional)	Determines when the job becomes active. Supported values: START_TYPE_NOW - The job starts immediately, and the trigger may fire right away. This is the default value when this property is omitted. START_TYPE_SCHEDULE - The trigger will not fire until the specified startDate.
startDate (conditional)	The date and time at which the job will be active, required when startType=START_TYPE_SCHEDULE. The trigger will not fire before this time; it will only fire for calendar occurrences that happen after this time. The format is "yyyy-MM-dd HH:mm" and the timezone property is applied.
endDate (optional)	The date and time at which the job will stop. The trigger will not fire after this time, unless a misfire occurs and the misfire policy allows it.

Property (as input)	Description
	The format is "yyyy-MM-dd HH:mm" and the timezone property is applied.
calendarName (optional)	The name of a previously defined exclusion calendar. An exclusion calendar defines a set of dates or times when the job will not run, for example a list of holidays. You can update the exclusion calendar without changing the job. For information about creating and modifying exclusion calendars, see The jobs calendars Services.
minutes (required)	Specifies the minute or minutes at which the trigger fires on a given hour. The value can consist of the following tokens: • A single minute value between 0 and 59. • A range of minutes, for example 0-10 means the trigger fires every minute starting from HH:00 to HH:10. Minute values and ranges can be concatenated using commas as separators. • A minute value with an increment, for example 5/10 means that the trigger fires every 10 minutes starting from HH:05. • * means the trigger fires every minute of the hour.
hours (required)	Specifies the hour or hours during which the trigger fires on a given day. The minutes property determines when during the hour that the trigger fires, possibly multiple times. All hours are specified in the 24-hour format. The value can consist of the following tokens: • A single hour value between 0 and 23. • A range of hours, for example 8-16 means the trigger fires every hour from 8 AM to 4 PM. Hour values and ranges can be concatenated using commas as separators. • An hour value with an increment, for example, 10/2 means the trigger fires during the hour every 2 hours starting from 10 AM. • * means the trigger fires during every hour.
months	A list of month values during which the trigger fires. Can be used in

Property (as input)	Description

(required)	addition to weekDays or monthDays below to suppress the trigger during certain months. The month values are 1 for January and 12 for December. In JSON, it has the following syntax:
	`"months": {`
	`    "month": ["3", "6", "9", "12"]`
	`}`

daysType (required)	Determines how the trigger days are defined: • WEEK means the trigger fires on a weekly pattern defined in the weekDays below. • MONTH means the trigger fires on a monthly pattern defined in the monthDays below. • ALL means the trigger fires every single day at the time or times defined by the hours and minutes properties.

weekDays (conditional)	Specifies a list of days of the week on which the trigger fires, to be repeated every week. This is required if daysType=WEEK, and ignored otherwise. The day values are 1 for Sunday and 7 for Saturday. On the designated days, the trigger fires at the time or times defined by the hours and minutes properties. In JSON, it has the following syntax:
	`"weekDays": {`
	`    "day": ["1", "4", "6"]`
	`}`

monthDays (conditional)	Specifies the days of the month on which the trigger fires, to be repeated every month. This is required if daysType=MONTH, and ignored otherwise. On the designated days, the trigger fires at the time or times defined by the hours and minutes properties. The value can consist of the following tokens: • A single day value between 1 and 31. • A range of days, for example 2-5 means the trigger fires during

Property (as input)	Description
	each day starting from the 2nd to the 5th of the month. Day values and ranges can be concatenated using commas as separators, for example "1,3,5-22". • A day value with an increment, for example 1/5 means the trigger fires every 5 days starting on the 1st of the month. • * means the trigger fires during every day.
misfire Instruction (optional)	An integer value that defines the behavior if the trigger did not fire when scheduled. The values of misfireInstruction are described in the next table. A misfire occurs if JasperReports Server or its Quartz scheduler component is offline when a trigger was supposed to happen and run a job. It can also occur if all threads of the scheduler are busy and the job cannot run when the trigger should fire. When the scheduler restarts or a thread becomes available again, it checks for any triggers that did not fire on time and takes action based on the misfireInstruction. The misfire instruction does not apply if the trigger fires normally but the report encounters an error. The default value is 0 when this property is omitted.

For example, the following calendar trigger should run at 3:30 AM every Monday in every month.

```
"trigger": {
    "calendarTrigger": {
        "timezone": "America/Denver",
        "startType": 2,
        "startDate": "2022-02-22 00:00",
        "misfireInstruction": 0,
        "minutes": "30",
        "hours": "3",
        "daysType": "WEEK",
        "weekDays": {
            "day": ["2"]
        },
        "months": {
            "month":
["1","2","3","4","5","6","7","8","9","10","11","12"]
        }
    }
}
```

Choose a misfire policy based on how frequently your job runs and how critical it is. For example, an outage may last one to two hours, and if a daily report is critical, you may want it to run as soon as the scheduler is able. However, if a report runs every hour, you may want to ignore missed reports and wait for the next scheduled time. Note that different policies may have the same effect depending on how the trigger is defined, but also the same policy may have different effects on different trigger types.

misfireInstruction	Description for Calendar Triggers
0 (default)	No instruction (same behavior as option -1 below). Does not trigger the job that misfired, and takes no action. This is the default behavior if no misfireInstruction is defined. The trigger will fire the next time according to the times and days that were defined.
-1	Ignore misfire policy. Instructs the scheduler that the trigger will never be evaluated for a misfire situation, and that the scheduler will simply try to fire it as soon as it can, and then update the trigger as if it had fired at the proper time. This value has the same effect as 0: take no action and wait for the next calendar trigger.
-999	Called the smart policy. Instructs the scheduler that on a misfire situation, the custom updateAfterMisfire method is called on the trigger to determine the misfire action. In this case, you must define and enable a custom trigger class on the server (this is beyond the scope of this documentation).
1	Run now: instructs the scheduler to trigger now, which is the time the misfire is detected. If the outage covers several trigger times, they will each have a misfire, and with this value, they will each run now.
2	Ignores any missed firings of the trigger (no action is taken for the missed firing), but updates the trigger to fire at the next scheduled time after now, taking into account any associated exclusion calendar or end time. The effect is that missed trigger occurrences will be skipped.

Job Output Properties

The output properties define the folders and filenames for the output files that are generated each time a scheduled job runs successfully.

Property (as input)	Description
repository Destination (optional)	A container for the following properties.
id (ignored)	An internal ID of the repository destination that has no outside purpose.
version (ignored)	An internal version number that has no outside purpose.
saveToRepository (optional)	A boolean that determines whether the output is written as file resources in the repository. The default value is true. Use the next two parameters to specify the location of the files. When false, no output is written to the repository, local file system, or remote FTP, and the output files are sent only by email if configured.
UsingDefaultReport OutputFolderURI (optional)	A boolean that determines if the output is written to the default folder: /public/Samples/Reports or /public/Samples/Dashboards (these folders can be configured as described in the JasperReports Server Administrator Guide). By default, this is false and you must specify the folderURI.
folderURI (conditional)	Defines a folder in the repository where the output file resources will be saved. This repository URI is relative to the scope of the user who authenticates the REST call. This property is required if you do not use the default output folder above.
outputDescription (optional)	A string that becomes the description of the output file resources in the repository.
sequential Filenames (optional)	A boolean that indicates whether output files have the time stamp pattern in the next parameter appended to the base filename. The default value is false.

Property (as input)	Description
timestampPattern (conditional)	A string containing "yyyyMMddHHmm" tokens to append a timestamp to the base filename. This takes effect if sequentialFilenames is true. For example, an hourly trigger might use "yyyyMMdd-HHmm" but a daily trigger only "yyyy-MM-dd". Characters in the pattern must be valid in file names where the output is being written (repository, local file system, or remote FTP). There is an implicit "-" (dash) character added between the base file name and the pattern, for example, the pattern "MM-dd" becomes basename-02-14.pdf.
overwriteFiles (optional)	A boolean to determine the behavior when a newly generated output file has the same name as a previous one. When true, the new file overwrites the old one. When false, the old file is not overwritten and the new file is not stored, though it is sent by email if configured. The default value is true. Use the sequentialFilenames and timestampPattern to create unique file names if you wish to store every generated output.
outputLocalFolder (optional)	Specifies a path in the local file system (of the JasperReports Server host) for writing output files. The user running the server process must have write permission in that location. When specified, file system output is always in addition to repository output, and the output is written to both the repository and the local folder. Before setting this property, the server must be configured to allow it. In the file .../WEB-INF/applicationContext.xml, you must set the enableSaveToHostFS property to true. However, this setting also enables file system output from the scheduler user interface for all users, which could be a security risk. The file names to be written are the same as for the repository, therefore the baseOutputFilename and sequential pattern must be valid on the host file system. The file overwrite and sequential filename behavior also apply to file system output.
outputFTPInfo (optional)	Contains parameters for writing output files to a remote FTP location. For more information, see FTP Output.

FTP Output

You have the option to specify output to a remote server through FTP (File Transfer Protocol). When specified, FTP output is always in addition to repository output, and the output is written to both the repository and the FTP location. The file names to be written are the same as for the repository, therefore the baseOutputFilename and sequential pattern must be valid on the FTP file system. The file overwrite and sequential filename behavior also apply to FTP output.

Specify the FTP server, authentication, remote path, and other FTP settings in the following properties:

Property (as input)	Description
outputFTPInfo (optional)	A container for the following properties, itself contained in repositoryDestination.
type (optional)	Type of FTP connection requested: ftp (default), ftps, or sftp.
serverName (optional)	The domain and host name of the FTP server, for example "ftp.example.com".
port (optional)	Integer value that specifies the port number of the FTP server. The default value depends on the connection type: ftp = 21, ftps = 990, and sftp = 22.
userName (conditional)	The login user name for the FTP server.
password (conditional)	The login password for the given userName on the FTP server.
folderPath (conditional)	The path of the folder where the job output resources should be created.
implicit (optional)	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.
protocol	Specifies the secure socket protocol to be used, for example SSL or TLS.

Property (as input)	Description
(optional)	
prot (optional)	Specifies the PROT command for FTP. Supported values: • C - Clear • S - Safe (SSL protocol only) • E - Confidential (SSL protocol only) • P - Private
pbsz (optional)	Specifies the protection buffer size: a decimal integer from 0 to (2^32)-1. The default is 0.
sshKey (conditional)	When using SFTP authentication, store the SSH key file in the repository and specify its repository URI in this property. For more information about secure files in the repository, see the JasperReports Server Administrator Guide.
sshPassphrase (conditional)	When using SFTP authentication, specify the passphrase for the SSH private key file stored in the repository.

The following example shows a simple use of outputFTPInfo in XML:

```
<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</scheduledOutput>
        </outputFTPInfo>
    </repositoryDestination>
    <outputFormats>
```

```
        <outputFormat>XLS</outputFormat>
        <outputFormat>PDF</outputFormat>
    </outputFormats>
  ...
</job>
```

Job Output Email

When a job runs successfully, the properties in mailNotification specify email recipients for the report output files. You can specify subject and body content, and you can choose to send output as attachments or links. When a job fails, no report output is sent but you have the option to configure an error message in Job Status Email.

Property (as input)	Description
mailNotification (optional)	This container and the following properties are optional. When omitted, the report output is not sent by email.
id (ignored)	An internal ID of the mailNotification container that has no outside purpose.
version (ignored)	An internal version number that has no outside purpose.
toAddresses (required)	A list of addresses to which job output emails are sent. This property is required if you are sending an output email.
ccAddresses (optional)	A list of addresses to which job output emails are CC'd.
bccAddresses (optional)	A list of addresses to which job output emails are BCC'd.
subject (optional)	A string that is used as a subject line for all output emails from this job.
messageText (optional)	A string that is used in the body of output emails for this job, unless HTML output is selected for the email body below.

Property (as input)	Description
resultSendType (required)	Determines whether output files are attached to the notification email. Each value has a different behavior and you must set this property based on your other properties: SEND - The links to the report output in the repository are appended to the message body, and no files are attached. This option is valid only if saveToRepository is set to true in the job descriptor. SEND_ATTACHMENT - The output files are sent as individual attachments to the email. SEND_EMBED - The HTML output is embedded in the email body, and any other output files are sent as individual attachments. This option is valid only if HTML output is specified in outputFormats for a report. SEND_ATTACHMENT_ZIP_ALL - The output files are sent as a single zipped attachment to the email. SEND_EMBED_ZIP_ALL_OTHERS - The HTML output is embedded in the email body, and any other output files are sent as a single zipped attachment. This option is valid only if HTML output is specified in outputFormats for a report.
skipEmptyReports (optional)	A boolean that suppresses the output email when the report is empty (successful completion but no content).
skipNotification WhenJobFails (ignored)	Two legacy properties that do not affect email notification. Use the properties in the alert container below to specify failure notifications.
includingStack TraceWhenJobFails (ignored)	

Job Status Email

The second type of notifications is job success and failure emails specified in the alert container. These messages indicate whether the job ran successfully or not, and contain an error message in case of failure. They do not include the report output in case of success.

Regardless of any settings here, the scheduler sends completion status to the job owner and the administrators of the same organization on each job trigger, assuming those users have email addresses defined in their profiles. You can change this behavior, as described in the configuration chapter of the JasperReports Server Administrator Guide. The scheduler also sends a server message in case of job failure (visible to administrators in View > Messages).

Property (as input)	**Description**
alert (optional)	This container and the following properties are optional. When omitted, the server sends only the default notifications listed above.
id (ignored)	An internal ID of the alert container that has no outside purpose.
version (ignored)	An internal version number that has no outside purpose.
jobState (required)	Selects which job status events trigger an email notification: ALL, FAIL_ONLY, SUCCESS_ONLY, or NONE. This property is required if you are sending a status email, however NONE is the same as not specifying an alert.
toAddresses (required)	A list of addresses to which job status emails are sent. This property is required if you are sending a status email.
subject (optional)	A string that is used as a subject line for all status email from this job.
messageText (optional)	A string that is used in the body of an email for a successful job completion.
messageTextWhen JobFails (optional)	A string that is used in the body of an email when the job fails. Additional information can be added to failure notifications with the following two properties.
includingReport JobInfo (optional)	Appends the job info (label and id) to the email body.
includingStack Trace (optional)	In the case of a job failure, appends the stack trace to the email body. This includes the exception that caused the report to fail.

Creating a Job

Contrary to REST conventions, the jobs service uses the PUT method to create a job and the POST method to modify a job. To schedule a report, report option, or dashboard, specify its properties in a job descriptor and use the PUT method of the jobs service. Specify the repository path to the resource being scheduled inside the job descriptor.

The user who is authenticated when making this request become the owner of the job that is created. This affects who can view the job and who receives notifications.

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 in The job Descriptor. You need to only specify the relevant properties in the job descriptor. Do not include any null properties.

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 and other default property values.	404 Not Found - When the report specified in the job descriptor is not found in the server.

The following example shows a basic job descriptor in JSON:

```
{
    "label": "Sample Job Name",
    "description": "Sample description",
    "trigger": {
        "simpleTrigger": {
            "timezone": "America/Los_Angeles",
            "startType": 2,
            "startDate": "2019-04-21 10:00",
            "occurrenceCount": 1,
            "recurrenceInterval": 1,
            "recurrenceIntervalUnit": "DAY"
        }
    },
    "source": {
        "reportUnitURI": "/adhoc/topics/Cascading_multi_select_topic",
        "parameters": {
            "parameterValues": {
                "Country_multi_select": ["Mexico"],
                "Cascading_name_single_select": ["Chin-Lovell
Associates"],
                "Cascading_state_multi_select":
["DF","Jalisco","Mexico"]
            }
        }
    },
    "baseOutputFilename": "Cascading_multi_select_report",
    "outputTimeZone": "America/Los_Angeles",
    "repositoryDestination": {
        "saveToRepository": true,
        "folderURI": "/temp",
        "overwriteFiles": true,
        "sequentialFilenames": false
    },
    "outputFormats": {
        "outputFormat": ["PDF", "XLS"]
    }
}
```

The following example shows a basic job descriptor in XML. As of release 7.5, input values must use the special type syntax below to pass strings, integers, and dates. The collection element is required even for a single value, and the item element is always of type string, as shown below:

```
<job>
    <label>Sample Job Name</label>
    <description>Sample description</description>
    <simpleTrigger>
        <timezone>America/Los_Angeles</timezone>
        <startType>2</startType>
        <startDate>2019-04-21 10:00</startDate>
        <occurrenceCount>1</occurrenceCount>
        <recurrenceInterval>1</recurrenceInterval>
        <recurrenceIntervalUnit>DAY</recurrenceIntervalUnit>
    </simpleTrigger>
    <source>
        <reportUnitURI>/adhoc/topics/Cascading_multi_select_
topic</reportUnitURI>
        <parameters>
            <parameterValues>
                <entry>
                    <key>StringParameter</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>IntegerParameter</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">123456</item></value>
                </entry>
                <entry>
                    <key>DateParameter</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">2021-12-
31</item></value>
                </entry>
            </parameterValues>
        </parameters>
    </source>
    <baseOutputFilename>Cascading_multi_select_
report</baseOutputFilename>
    <outputTimeZone>America/Los_Angeles</outputTimeZone>
    <repositoryDestination>
        <saveToRepository>true</saveToRepository>
        <folderURI>/temp</folderURI>
        <overwriteFiles>true</overwriteFiles>
        <sequentialFilenames>false</sequentialFilenames>
    </repositoryDestination>
    <outputFormats>
        <outputFormat>PDF</outputFormat>
        <outputFormat>XLS</outputFormat>
    </outputFormats>
</job>
```

If needed, you can configure the server to accept the other parameters and keep them with the newly created job, but the default is to store only the required properties. For more information, see Storing Additional Job Properties.

The response of the PUT request is the descriptor of the newly created job, similar to the result of the GET request shown in Viewing a Job Definition. It includes all the properties of the job descriptor, including the server-assigned ID and all the null properties.

Viewing Job Status

The following method returns the current 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 state descriptor.	404 Not Found - When the specified <jobID> does not exist or the job is no longer active.

The following example shows a typical response in XML:

```
<state>
    <nextFireTime>2022-03-29T18:01:00-07:00</nextFireTime>
    <previousFireTime>2022-03-28T18:01:00-07:00</previousFireTime>
    <value>NORMAL</value>
</state>
```

Either of the fire times may be missing, for example there is no previousFireTime if the job has not fired yet, or there is no nextFireTime if the job is currently running its last trigger time. The value property can be: NORMAL, EXECUTING, PAUSED, or some error state.

Modifying a Job

Contrary to REST conventions, the jobs service uses the PUT method to create a job and the POST method to modify a job. There are two POST methods for editing the properties of a job. The first method is simpler but the second method is more versatile.

The first method replaces the entire definition of a single job with a new descriptor. To modify an existing job definition, use the GET method to read its job descriptor, modify the desired properties, then use the following POST method:

Method	**URL**
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/<jobID>/

Content-Type	**Content**
application/job+xml application/job+json	A complete, well-formed XML or JSON job descriptor, as described in The job Descriptor. It may include null properties and other default values that are ignored when using the descriptor as input, as in the result of Viewing a Job Definition.

Options
accept: application/job+xml accept: application/job+json

Return Value on Success	Typical Return Values on Failure
200 OK - The response includes the complete job descriptor updated with the submitted changes.	404 Not Found - When the specified <jobID> does not exist or the job is no longer active.

The second POST method lets you modify individual properties in one or more existing jobs. See the examples below:

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.
Content-Type		**Content**
application/json application/xml		A well-formed jobModel descriptor, which is a fragment of a job descriptor containing only the properties to be updated. See the examples 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 modified.	404 Not Found - When the specified <jobID> does not exist or the job is no longer active.

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 properties. This update applies to one or more jobs whose ID is specified by the id argument. For example, the following simple request updates the job description in several jobs:

```
POST http://localhost:8080/jasperserver-pro/rest_
v2/jobs?id=3798&id=3802&id=3806

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

The jobModel provides two mechanisms to perform more 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 isPropertyNameModified 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 is removed from the target jobs, the trigger's time zone is modified, and the file name is changed. Note that XML descriptors do not use the trigger container and thus do not have a triggerModel container:

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":[8322,8326]}
```

XML:

```
<jobIdList>
          <jobId>8322</jobId>
          <jobId>8326</jobId>
      </jobIdList>
```

Pausing Jobs

The following method pauses currently scheduled job execution, also called disabled in the user interface. 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 the example below. If the body of the request is empty, or the list is empty, all jobs in the scheduler are 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. Jobs specified with a <jobID> that does not exist are ignored without error.	

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

```
JSON: {"jobId":[1236,1240,1244,1248]}

XML:  <jobIdList>
          <jobId>1236</jobId>
          <jobId>1240</jobId>
          <jobId>1244</jobId>
          <jobId>1248</jobId>
      </jobIdList>
```

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 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 the example below. If the body of the request is empty, or the list is empty, all paused jobs in the scheduler is resumed.

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. Jobs specified with a <jobID> that does not exist are ignored without error.	

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

JSON:

```
{"jobId":[1236,1240]}
```

XML:

```
 <jobIdList>
          <jobId>1236</jobId>
          <jobId>1240</jobId>
      </jobIdList>
```

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 the 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 the 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":[8320,8324]}
```

XML:

```
<jobIdList>
           <jobId>8320</jobId>
           <jobId>8324</jobId>
      </jobIdList>
```

Deleting Jobs

Use the DELETE method to remove jobs from the scheduler. The first form deletes a single job:

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 or the job is no longer active.

The second form deletes multiple jobs:

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=5640&id=5762
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":[5594,5640,5762]}
```

XML:

```
<jobIdList>
         <jobId>5594</jobId>
         <jobId>5640</jobId>
         <jobId>5762</jobId>
     </jobIdList>
```

Storing Additional Job Properties

When sending a job descriptor as described in The jobs Service, the server does not store all properties in the descriptor, only the ones needed to define the job. If you wish to keep any additional parameters in the newly created job, you can configure the server so that all valid job properties submitted to the jobs service are stored.

Locate the following file and modify the configuration bean. After saving the new configuration, you must restart the server for the change to take effect.

Storing Additional Job Parameters	
Configuration File	
.../WEB-INF/applicationContext-cascade.xml	
Bean	**Description**
allowExtraReportParameters	The default value is false, and only essential job properties are stored when creating a job. When set to true, all valid job descriptor properties sent when creating the job are stored in the newly created job.

The alerts Service

The rest_v2/alerts service provides the interface to create, read, update, delete, and manage single or multiple alerts in the report. When a user schedules an alert in the report to run at a given time, with a given recurrence, the server stores this information in an alert. There can be any number of alerts for any number of reports, all created by different users.

With the server's user interface, the alerts service only manages alerts for the reports that are currently active. Alerts are never listed or returned for the chart reports and the reports without numeric values.

Within the alert, a trigger is said to fire when the specified condition is met. When the condition is met, the server generates the output of the scheduled alert.

For example, consider that you are working in a warehouse. You have to restock the inventory based on the storage space. A report is generated which shows the inventory for the products. You need to check the stock twice a day to maintain inventory value. You can define an alert to notify whenever a stock is low in number. Based on which, a report in the output formats like PDF, Excel is emailed to the recipient as a repository links or attachments.

This chapter includes the following sections:

- Searching for an Alert
- Viewing an Alert Definition
- The alert Descriptor

Searching for an Alert

The GET method of the alerts service has multiple arguments to search for alerts in several ways. The credentials used for authentication determine the scope of the search. Typical users can search only the alerts that they have created. An organization admin (jasperadmin) can search all alerts in their organization, and the system admin (superuser) can search all alerts on the server. This method only returns active alerts, that still have a pending trigger or are still running after their last trigger.

When used without any arguments, this method returns all scheduled alerts defined by the user in the report. The various arguments in this method help you to find an alert created by the user using administrator credentials.

You can perform a string search in the search criteria to find an alert by name in the user interface. The search criteria show results only for those alert whose name matches with the alert name column. You can also control the number of alert results to be displayed in a single alert page using the pagination and sorting order option.

Method	**URL**	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts?<arguments>	
Argument	**Type/Value**	**Description**
id?	Integer	The unique identifier assigned to the alert.
resourceURI?	/path/to/report	The URI (repository path) of a report to schedule an alert. Performs a case-insensitive string search for this value anywhere within the resourceURI property of every alert.
owner?	String	The name of the user who scheduled the alert.
label?	String	The label to search an alert by Alert name. Performs a case-insensitive string search for this value, which contains the label property of every alert.
description?	String	The description of the alert.
resourceLabel?	String	The name of the report to find alerts in the report.
state?	String	The alert states - NORMAL, EXECUTING, COMPLETE, PAUSED, ERROR, UNKNOWN.
previousFireTime?	String	The alerts previous fire time.
nextFireTime?	String	The alerts next fire time.

example?	JSON alertModel	Searches for an alert that matches with the JSON `alertModel`. It should be the fragment of the alert descriptor with one or more properties to be matched. You can search for any parameter in the alert descriptor such as - id, version, trigger, output format, or recipient email.
offset?	Integer	Specifies the index of the first `alertsummary` to be returned. When used with the limit parameter, it can be used to implement pagination.
limit?	Integer	Specifies the number of `alertsummary` descriptors in the results. The default is -1 for no limit and thus all results are returned. When used with the offset parameter, it can be used to implement pagination of results.
sortType?		Possible values are: `NONE`, `SORTBY_ALERTID`, `SORTBY_ALERTNAME`, `SORTBY_RESOURCEURI`, `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. This argument is ignored when sortType is not specified.
Options		
accept: application/json		

Return Value on Success	**Typical Return Values on Failure**
200 OK - The body contains an array or list of alertsummary descriptors that match the search criteria, as shown in the examples below.	204 No Content - When no matching alert is found in the server. After an alert

		finishes running its last triggered instance, it no longer shows up in search results.

For example, if you want to request all the alerts for a report, then use the following GET method URL (%2F is the / character) in the request:

```
GET http://example.com:8090/jasperserver-pro/rest_
v2/alerts?reportUnitURI=%2Freports%2FAllAccounts
```

In the response from the server, the alerts are described in an `alertsummary` element, such as the following example. The `alertsummary` contains a small subset of the properties of the alert descriptor as well as the complete state descriptor:

```
{
    "alertsummary": [
        {
            "id": 4164,
            "version": 1,
            "label": "Test",
            "reportUnitURI":
"/public/Samples/Reports/RevenueDetailReport",
            "reportLabel": "07. Revenue Detail Report",
            "state": {
                "previousFireTime": "2023-04-24T15:26:00-07:00",
                "nextFireTime": "2023-04-26T15:26:00-07:00",
                "value": "NORMAL"
            },
            "hasDataPointAlert": true,
            "owner": "superuser",
            "lastAlerted" : null
        }
    ]
}
```

The example parameter lets you specify a search of any property in the alert descriptor, such as output formats. You can specify any property in the alert descriptor or in any of its nested structures. Some properties 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 alerts that specify an output format of PDF and XLS. The JSON alertModel to specify this property is:

```
"outputFormats": {
"outputFormat": ["PDF", "XLS"]
}
```

Below is the corresponding URI with proper encoding is:

```
http://<host>:<port>/jasperserver[-pro]/rest_
v2/alerts?example=%7b%22outputFormat%22%3a%22PDF%22%7d
```

Viewing an Alert Definition

Use the GET method to retrieve details of an alert using its unique id. You can use this method to view an alert definition after searching and finding the ID of an alert that is still active.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts/{alertId}
Options	
accept: application/alert+json	
Return Value on Success	**Typical Return Values on Failure**
200 OK - The body contains a descriptor with all details about the alert.	404 Not Found - When the specified alert is not found in the server. After an alert finishes running its last triggered instance, it is no longer active, then its ID will return this error.

The GET method returns a descriptor that includes the various parameters of a scheduled alert. All properties are included, many of which may be null, if not set for the chosen alert. For more information, see The alert Descriptor.

```
{
    "id": 4198,
    "version": 1,
    "username": "superuser",
    "label": "TestPostMan",
    "creationDate": "2023-04-28T00:46:17.945-07:00",
    "trigger": {
        "simpleTrigger": {
            "id": 4195,
            "version": 0,
            "timezone": "America/Los_Angeles",
            "startType": 1,
            "misfireInstruction": 0,
            "occurrenceCount": -1,
            "recurrenceInterval": 2,
            "recurrenceIntervalUnit": "DAY"
        }
    },
    "lastError": null,
    "baseOutputFilename": "RevenueDetailReport",
    "exportType": "DEFAULT",
    "outputLocale": "en_US",
    "outputTimeZone": "Asia/Calcutta",
    "source": {
        "reportUnitURI": "/public/Samples/Reports/RevenueDetailReport",
        "parameters": {
            "parameterValues": {
                "ProductFamily": [
                    "Drink",
                    "Food",
                    "Non-Consumable"
                ]
            }
        }
    },
    "outputFormats": {
        "outputFormat": [
            "PDF"
        ]
    },
    "repositoryDestination": {
        "folderURI": "/public/Test",
        "id": 4196,
        "overwriteFiles": true,
        "sequentialFilenames": false,
        "version": 1,
        "saveToRepository": true,
        "usingDefaultReportOutputFolderURI": false,
        "outputFTPInfo": {
            "userName": "anonymous",
            "type": "ftp",
            "port": 21,
            "implicit": true,
            "pbsz": 0,
            "propertiesMap": {}
        }
    },
    "mailNotification": {
        "bccAddresses": {
            "address": []
        },
        "ccAddresses": {
            "address": []
        },
        "toAddresses": {
            "address": [
                "xyz@tibco.com"
            ]
        },
        "version": 1,
        "id": 4197,
        "includingStackTraceWhenJobFails": false,
        "messageText": "test",
        "resultSendType": "SEND",
        "skipEmptyReports": false,
        "skipNotificationWhenJobFails": false,
        "subject": "test"
    },
    "dataPointAlert": {
        "id": 4199,
        "version": 0,
        "name": "Test API PostMan",
        "dataPoint": "{\"elementUUID\":\"72a3ea24-62a2-4f04-a5a6-
d501046ddbcc\"}",
        "operator": "EQUALS",
        "thresholdValue": "0",
        "dataPointType": "NUMERIC",
        "resourceURI": "/public/Samples/Reports/RevenueDetailReport"
    }
}
```

The alert Descriptor

The alert descriptor is a complex data object with nested containers for the various properties that defines a scheduled alert. The `alert` descriptor is the output of the GET method with slightly different content, and the input of the PUT and POST methods.

The properties of the `alert` descriptor are defined in the following sections:

- General Properties of an Alert, such as label and description, but also the output format and base filename.
- Source and Input Controls includes the repository URL of the report or report option and the parameter value of any input control if any of them are present and applied to the same report.
- Simple Trigger defines interval-based repetition of the alert for a given number of occurrences.
- Calendar Trigger runs at specific times, specific days of the week or days of the month.
- Alert Output Properties define the file name and locations where output files are written.
- FTP Output defines whether the output files are written to a remote server.
- Alert Output Email defines the recipients for successful output files.
- Alert DataPoint defines the numeric values for which an alert is created.

When submitting an alert descriptor to create or modify an alert schedule, not all properties are needed. In the following tables, each property is one of the following:

- **Required** - This property must have a value for input to define a valid alert.
- **Optional** - This property may be omitted on input, either because it is nullable, or because the server assigns a default value. The behavior is explained in the property description.
- **Conditional** - The property may be required or optional depending on other property values. The behavior is explained in the property description.
- **Ignored** - This property is for internal usage or output only, and the server ignores any value on input. Good practice is to omit these properties from your input.

General Properties of an Alert

A valid alert descriptor contains the following properties:

Property (as input)	Description
`id` (required)	The ID of the alert assigned by the server. Use this ID to modify, delete, or view the alert.
`version` (optional)	An internal version number, which is used for optimistic locking of an alert object.
`username` (required)	The owner and creator of this alert, including the organization name. Note that only administrators can view other users' alerts.
`label` (required)	A name for the alert, like the label of a resource in the repository.
`description` (optional)	A description string for the alert, which often describes the trigger and chosen output.
`creationDate` (required)	The time and date when the alert was first created and managed internally by the server.
`trigger` (optional)	A container for one of the triggers specified in Simple Trigger or Calendar Trigger. The trigger determines how often the alert runs and the date and time at which it runs.
`source` (required)	A container for the properties that define the repository URI of the report to run parameters. For more information, see Source and Input Controls.
`exportType` (optional)	The default value is `DEFAULT`, and for scheduling an alert in the report no other value is accepted. Not all output formats are available depending on the value of `exportType` (see next property).
`outputFormats` (required)	It contains a list of output formats, each to be generated in a separate output file when the alert is triggered. The possible values depend on the resource and the exportTypeabove.

Property (as input)	Description
	Report (`exportType="DEFAULT"` is required): `PDF, HTML, RTF, CSV, ODT, TXT, DOCX, ODS, XLSX, XLSX_NOPAG, DATA_SNAPSHOT, DATA_CSV` The XLS and XLS_NOPAG (Excel 2003) output formats are deprecated. You can use XLSX and XLSX_NOPAG (Excel 2016) instead.
`outputTimeZone` (optional)	The time zone to use when running the report, for example "America/Los_Angeles". By default, the alert is set on the report using the server's default time zone. The time zone names are those supported by `java.time.ZoneID`, which are defined in the tz database. The trigger time zone and output time zone have different purposes even when they are often set to the same value. The trigger time zone determines when the alert is scheduled to run. The output time zone affects anytime or date calculations in the report. The time zone specified in a data source also affects the time and dates in the output.
`outputLocale` (optional)	The locale to use when running the report. By default, the alert is scheduled using the server's default locale. If the report is not based on a Domain with locales, this value has no effect. Use a Java locale string that is also defined in your Domain, for example `en` (England) or `en_CA` (England or Canada).
`baseOutput Filename` (required)	The basename of the file for the generated report alert output. Each output format appends its corresponding file extension to this name. This name may also have a time stamp appended as specified in Alert Output Properties. This final output name is then used in all locations where the file is saved: repository, file system, FTP, and email attachment.
`repository Destination` (required)	A container for properties that define the folders and filenames for the output files that are generated each time a scheduled alert runs successfully. Its properties are defined in Alert Output Properties and the optional FTP Output.

Property (as input)	Description
`mailNotification` (required)	A container for properties that define email recipients when an alert runs successfully. You can customize the contents of the email and whether the report output files are sent as attachments or links. For more information, see Alert Output Email.
`systemNotification` (optional)	A container for properties that define system status when an alert runs successfully. You can customize the contents of the email and whether the report output files are sent as attachments or links. For more information, see Alert Output Email.
`lastError` (optional)	If an alert fails, the `lastError` field gets populated with the failure message.
`lastAlerted` (optional)	If an alert is successful, then only the `lastAlerted` field gets populated with the time when it was alerted.

Source and Input Controls

The source is the report being scheduled by the alert along with any required input controls.

Property (as input)	Description
`reportUnitURI` (required)	A mandatory property that defines the repository URI or the report options to schedule an alert in the report. The repository URI is relative to the scope of the user credentials used to authenticate the REST call.
`parameters` `parameterValues` (optional)	A container for the input controls (filters) for an alert. Every required input control in the report must have a JSON list item within `parameterValues`. If there are no input controls, this property can be omitted. See the example in Creating a Alert for the syntax of these properties in JSON.

Simple Trigger

The following properties define the recurrence pattern for a simple trigger.

Property (as input)	Description
`simpleTrigger` (conditional)	A container for the following properties, itself contained in a required `trigger`. For input, you must specify either the `simpleTrigger` or the `calendarTrigger`.
`timezone` (optional)	The timezone for all dates and times used by the trigger, for example `"America/Los_Angeles"`. By default, the trigger uses the same time zone as the server. The time zone names are those supported by `java.time.ZoneID`, which are defined in the tz database.
`calendarName` (optional)	The name of a previously defined exclusion calendar. An exclusion calendar defines a set of dates or times when the alert cannot be scheduled. For example, a list of holidays. You can update the exclusion calendar without changing the alert.
`startType` (optional)	Determines when the alert becomes active and sets the base time at which recurrence starts. Supported values: `START_TYPE_0` - The scheduled alert starts immediately, and the trigger fires right away. This is the default value when this property is omitted. `START_TYPE_1` - The scheduled alert starts at the specified `startDate` on which the trigger is set to fire.
`startDate` (conditional)	The date and time at which the alert is scheduled to start, required when `startType=START_TYPE_1`. The simple trigger fires and begins its recurrence at this time. The format is "yyyy-MM-dd HH:mm" and the timezone property is applied.
`endDate` (optional)	The date and time at which the alert stops. The trigger does not fire after this time, even if any occurrences remain unless a misfire occurs, and the misfire policy allows it. The format is "yyyy-MM-dd HH:mm" and the timezone property is applied.
`misfire Instruction`	An integer value that defines the behavior if the trigger did not fire

Property (as input)	Description
(optional)	when scheduled. The values of `misfireInstruction` are described in the next table. A misfire occurs if the alert is inactive or when there are no available threads in the Quartz thread pool for executing the alert. The different trigger types have different misfire instructions available to them. By default they use a 'smart policy' instruction - which has dynamic behavior based on trigger type and configuration. When the alert starts, it searches for any persistent triggers that have misfired, and it then updates each of them based on their individually configured misfire instructions.
`occurrenceCount` (required)	An integer that defines how many times the trigger fires, provided the recurrence intervals happen before the `endDate`.
`recurrence Interval` (required)	The time interval between firings of the trigger, with the interval unit provided in the next property.
`recurrence IntervalUnit` (required)	The unit of time for the recurrence interval. Supported values: `MINUTE`, `HOUR`, `DAY`, or `WEEK`. For units greater than `MINUTE`, the `startType` and `startDate` determine the basis for recurrence. For example, if the trigger fires immediately and recurs every 2 days, it fires at the current time on the subsequent days.

Choose a misfire policy based on how critical it is and how frequently you want to get notified about the alert. For example, an outage may last 1-2 hours, and if a daily report is critical, then you might want to get notified about the alert as early as possible. But, if a report runs every hour, then you might want to ignore missed alerts on the reports and wait for the next alert at the scheduled time.

Different policies might have the same effect depending on how the trigger is defined. Also, the same policy may have different effects on different trigger types.

misfireInstruction	Description for Simple Triggers
0 (default)	No instruction (same behavior as option -1 below). Does not trigger the alert that misfired, and takes no action. As the trigger did not fire, the number of occurrences is not decremented. This is the default behavior when no `misfireInstruction` is defined. The trigger fire the next time according to the recurrence value and unit, as if the misfire had fired at the proper time.
-1	Ignore misfire policy. Instructs the alert that the trigger is never evaluated for a misfire situation. In this case, the alert simply tries to fire it as soon as it can, and then update the trigger as if it had fired at the correct time. This value has the same effect as 0: take no action and do not change the number of occurrences.
-999	Called the smart policy. Instructs the alert that on a misfire situation, the custom `updateAfterMisfire` method is called on the trigger to determine the misfire action. In this case, you must define and enable a custom trigger class on the server (This is out of the scope of this document).
1	Run now: instructs the alert that on a misfire situation, the trigger wants to be fired now by the alert.
2	Instructs the alert that during a misfire situation, the SimpleTrigger has to be rescheduled to 'now' (even if the associated Calendar excludes 'now') with the repeat count left 'as-is'. This follows the trigger end-time rule, so if 'now' is after the end-time, then the trigger does not fire.
3	Instructs the alert that on a misfire situation, the SimpleTrigger need to be rescheduled to 'now' (even if the associated Calendar excludes 'now') with the repeat count set to any value, if it had not missed any firings. This follows the trigger end-time, so if 'now' is after the end-time, then the trigger does not fire after the end time.
4	Instructs the alert that on a misfire situation, the SimpleTrigger need to be rescheduled to the next scheduled time after 'now' - considering

misfireInstruction	Description for Simple Triggers
	any associated Calendar, and with the repeat count set to any value, if it had not missed any firings.
5	Instructs the alert that on a misfire situation, the SimpleTrigger wants to be rescheduled to the next scheduled time after 'now' - considering any associated Calendar, and with the repeat count left unchanged.

Calendar Trigger

A calendar trigger lets you schedule an alert to run multiple times based on any combination of time and date. Its properties let you define single values, ranges, or wild-cards for minutes, hours, days, weeks, or months. For example, you can run a report every 15 minutes from 10 AM to noon every Monday.

The following properties define the recurrence pattern of a calendar trigger:

Property (as input)	Description
`minutes` (required)	Specifies the minute or minutes at which the trigger fires on a given hour. The value can consist of the following tokens: • A single minute value 0-59. • A range of minutes, for example 0-10 means the trigger fires every minute starting from HH:00 to HH:10. Minute values and ranges can be concatenated using commas as separators. • A minute value with an increment, for example 5/10 means that the trigger fires every 10 minutes starting from HH:05. • `*` means the trigger fires every minute of the hour.
`hours` (required)	Specifies the hour or hours during which the trigger fires on a given day. All hours are specified in a 24-hour format. The value can consist of the following tokens: • A single hour value between 0-23.

Property (as input)	Description
	• A range of hours, for example 8-16 means the trigger fires every hour from 8 AM to 4 PM. Hour values and ranges can be concatenated using commas as separators. • An hour value with an increment, for example 10/2 means the trigger fires during the hour every 2 hours starting from 10 AM. • `*` means the trigger fires during every hour.

`months` (required)	A list of `month` values during which the trigger fires. It can be used in addition to `weekDays` or `monthDays` below to suppress the trigger during certain months. The month values are 1 for January and 12 for December. In JSON, it has the following syntax:
	`"months": {`
	`    "month": ["1", "11", "6"]`
	`}`

`daysType` (required)	Determines how the trigger days are defined: • `WEEK` means the trigger fires on a weekly pattern defined in the `weekDays` below. • `MONTH` means the trigger fires on a monthly pattern defined in the `monthDays` below. • `ALL` means that the trigger fires every day at the time or times defined by the `hours` and `minutes` properties.

`weekDays` (conditional)	Specifies a list of days of the week on which the trigger fires, which has to be repeated every week. This is required if `daysType=WEEK`, else ignored otherwise. The day values are 1 for Sunday and 7 for Saturday. On the designated days, the trigger fires at the time or times defined by the `hours` and `minutes` properties. In JSON, it has the following syntax:
	`"weekDays": {`
	`    "day": ["1", "4", "6"]`
	`}`

Property (as input)	Description
`monthDays` (conditional)	Specifies the days of the month on which the trigger fires, to be repeated every month. This is required if `daysType=MONTH`, and ignored otherwise. On the designated days, the trigger fires at the time or times defined by the `hours` and `minutes` properties. The value can consist of the following tokens: • A single day value 1-31. • A range of days. For example, 2-5 means the trigger fires during each day starting from the 2nd to 5th of the month. Day values and ranges can be concatenated using commas as separators, for example, `"1,3,5-22"`. • A day value with an increment, for example, 1/5 means that the trigger fires every 5 days starting on the first of the month. • `*` means the trigger fires during every day.

Choose a misfire policy based on how critical it is and how frequently you want to get notified about the alert. For example, an outage may last 1-2 hours, and if a daily report is critical, then you might want to get notified about the alert as soon as possible. But, if a report runs every hour, you might want to ignore missed alerts on the reports and wait for the next alert at the scheduled time.

Different policies might have the same effect depending on how the trigger is defined. Accordingly, the same policy might have different effects on different trigger types.

misfireInstruction	Description for Calendar Triggers
0 (default)	No instruction (same behavior as option -1 below). Does not trigger the alert that misfired, and takes no action. As the trigger did not fire, the number of occurrences is not decremented. This is the default behavior when no `misfireInstruction` is defined. The trigger fires the next time according to the recurrence value and unit, as if the misfire had fired at the proper time.
-1	Ignore misfire policy. Instructs the alert that the trigger is never

misfireInstruction	Description for Calendar Triggers
	evaluated for a misfire situation. In such cases, the alert simply tries to fire the trigger as early as possible. Subsequently, it then updates the trigger as if it had fired at the correct time. This value has the same effect as 0: take no action and do not change the number of occurrences.
-999	Called the smart policy. Instructs the alert that on a misfire situation, the custom `updateAfterMisfire` method is called on the trigger to determine the misfire action. In this case, you must define and enable a custom trigger class on the server (This is out of the scope of this document).
1	Run now: instructs the alert that on a misfire situation, the trigger wants to be fired now by the alert.
2	Instructs the alert that during a misfire situation, the SimpleTrigger has to be rescheduled to 'now' (even if the associated calendar excludes 'now') with the repeat count left 'as-is'. This follows the trigger end-time, so if 'now' is after the end-time, the trigger does not fire.
3	Instructs the alert that on a misfire situation, the SimpleTrigger has to be rescheduled to 'now' (even if the associated Calendar excludes 'now') with the repeat count set to any value, if it had not missed any firings. This does follow the trigger end-time, so if 'now' is after the end-time, the trigger does not fire.
4	Instructs the alert that on a misfire situation, the SimpleTrigger has to be rescheduled to the next scheduled time after 'now' - considering any associated Calendar with the repeat count set to any value, if it had not missed any firings.
5	Instructs the alert that on a misfire situation, the SimpleTrigger has to be rescheduled to the next scheduled time after 'now' - considering any associated Calendar with the repeat count left unchanged.

Alert Output Properties

The output properties define the folders and filenames for the output files that are generated each time a scheduled alert runs successfully.

Property (as input)	Description
`repository Destination` (optional)	A container for the following properties.
`saveToRepository` (optional)	A boolean that determines whether the alert output is written as file resources in the repository. The default value is true. Use the next two parameters to specify the location of the files. When false, no alert output is written to the repository, local file system, or remote FTP. The output files are then only sent by email.
`defaultReport OutputFolderURI` (optional)	Defines a default alert report folder in the repository of the alert owner. The output file resources are saved in this path: "defaultReportOutputFolderURI":"/some/folder/uri".
`UsingDefaultReport OutputFolderURI` (optional)	A boolean that determines if the output is written to the default folder: "usingDefaultReportOutputFolderURI": {true|false}(these folders can be configured as described in the *JasperReports Server Administrator Guide*). By default, this is false and you must specify the `folderURI`.
`folderURI` (conditional)	Defines a folder in the repository where the output file resources are saved in this path: "folderURI":"/some/folder/uri". This repository URI is relative to the scope of the user who authenticates the REST call. This property is required if you do not use the default output folder above.
`outputDescription` (optional)	A string that becomes the description of the alert output file resources in the repository. The description is used as it is by all output resources.
`sequentialFilenames` (optional)	A boolean that indicates if a timestamp is to be added to the names of the alert output resources. The timestamp added to the output resource names are created from the alert execution time

Property (as input)	Description
	using the specified pattern. The default value is false.
`timestampPattern` (conditional)	A string containing "yyyyMMddHHmm" tokens to append a timestamp to the base filename. This takes effect if `sequentialFilenames` is true. For example, an hourly trigger might use "yyyyMMdd-HHmm" but a daily trigger only "yyyy-MM-dd". Characters in the pattern must be valid in file names where the output is being written (repository, local file system, or remote FTP). There is an implicit "-" (dash) character added between the base file name and the pattern, for example, the pattern "MM-dd" becomes basename-02-14.pdf.
`overwriteFiles` (optional)	A boolean to determine the behavior when a newly generated output file has the same name as a previous one. When true, the alert owner does not have the permission to overwrite an existing resource, and the alert execution fails. When false, the alert fails if the repository already contains a resource with the same name as one of the alert output resources. The default value is false.
`outputLocalFolder` (optional)	Specifies a path in the local file system (of the JasperReports Server host) under which alert output resources would be created. The local path is the path in the JRS host's local file system. This functionality is by default disabled and this field is ignored. You can enable it in the file .../WEB-INF/applicationContext.xml, by setting the `enableSaveToHostFS` property to `true`.
`outputFTPInfo` (optional)	It contains parameters for writing alert output files to a remote FTP location. For more information, see FTP Output.

FTP Output

You have the option to specify output to a remote server through FTP (File Transfer Protocol). When specified, FTP output is always in addition to repository output, and the output is written to both the repository and the FTP location. The file names to be written are the same as for the repository, therefore the baseOutputFilename, and sequential pattern must be valid on the FTP file system. The file overwrite and sequential filename behavior also apply to FTP output.

Specify the FTP server, authentication, remote path, and other FTP settings in the following properties:

Property (as input)	Description
`outputFTPInfo` (optional)	A container for the following properties, itself contained in repositoryDestination.
`type` (optional)	The type of FTP connection requested: ftp (default), ftps, or sftp.
`serverName` (optional)	The domain and host name of the FTP server, for example "ftp.example.com".
`port` (optional)	Integer value that specifies the port number of the FTP server. The default value depends on the connection type: ftp = 21, ftps = 990, and sftp = 22.
`userName` (conditional)	The login username for the FTP server.
`password` (conditional)	The login password for the given userName on the FTP server.
`folderPath` (conditional)	The path of the folder under which the alert output resources should be created.
`implicit` (optional)	Specifies the security mode for FTPS. If true, then it is implicit (default), and when false, it is explicit. If the implicit is true, the default port is set to 990.
`protocol` (optional)	Specifies the secure socket protocol to be used, for example SSL or TLS.
`prot` (optional)	Specifies the PROT command for FTP. Supported values are: • C - Clear

Property (as input)	Description
	• S - Safe (SSL protocol only) • E - Confidential (SSL protocol only) • P - Private
`pbsz` (optional)	Specifies the protection buffer size: a decimal integer from 0 to (2^32)-1. The default is 0.
`sshKey` (conditional)	When using SFTP authentication, store the SSH key file in the repository and specify its repository URI in this property. For more information about securing files in the repository, see the *JasperReports Server Administrator Guide*.
`sshPassphrase` (conditional)	When using SFTP authentication, specify the passphrase for the SSH private key file stored in the repository.

Alert Output Email

When an alert runs successfully, the properties in mailNotification specify email recipients for the report output files. You can specify subject and body content, and you can choose to send output as attachments or links. However, if an alert fails, no report output is sent.

Property (as input)	Description
`toAddresses` (required)	A list of addresses to which alert output emails are sent. This property is required if you are sending an output email.
`ccAddresses` (optional)	A list of addresses to which alert output emails are CC'd.
`bccAddresses` (optional)	A list of addresses to which alert output emails are BCC'd.
`subject`	A string that is used as a subject line for all output emails

Property (as input)	Description
(required)	from this alert.
`messageText` (optional)	A string that is used in the body of output emails for the alert. At alert execution time, links to the output and errors might get appended to the notification message text.
`resultSendType` (required)	Determines whether the notification would include the alert output as attachments, or include links to the output in the repository. Each value has a different behavior and you must set this property based on your other properties: `SEND` - The email notification should contain links to the alert output generated in the repository. `SEND_ATTACHMENT` - The email notification should contain the alert output as attachments. `SEND_ATTACHMENT_NOZIP` - The email notification should contain the alert output in non-ZIP format attachments. `SEND_EMBED` - The email notification should embed the HTML alert output in the email body. `SEND_ATTACHMENT_ZIP_ALL` - The email notification should contain the alert output as attachments in one ZIP file. `SEND_EMBED_ZIP_ALL_OTHERS` - The email notification should embed the HTML alert output in the email body and put the other output type attachments in one ZIP file.
`skipEmptyReports` (optional)	A boolean that specifies whether the email notification should be skipped for alert executions to the produce empty reports (successful completion but no content). The default value is false.
`messageTextWhenAlertFails` (optional)	The text of the email notification when the alert fails. At alert execution time, links to the output and errors might get appended to the notification message text.
`skipNotification`	A boolean that specifies whether the mail notification

Property (as input)	Description
`WhenAlertFails` (optional)	should be send if the alert fails. The default value is false.
`includingStack TraceWhenAlertFails` (optional)	A boolean that specifies whether the mail notification would include a detailed stack trace of an exception. The default value is false.

Alert DataPoint

When an alert runs successfully, the properties in datapointalert specify the numeric values of the report using which the alert is created. You can specify the following values to create an alert in the report.

Property (as input)	Description
`dataPointAlert` (required)	A container for the following properties.
`id` (required)	The ID of the alert data point assigned by the server.
`version` (optional)	An internal version number, which is used for optimistic locking of an alert data point object.
`name` (required)	Specifies the name of the data point alert.
`dataPoint` (required)	Determines the element UUID from the report on which the alert is created.
`operator`	Supported values are: `equals`, `notEqual`, `less`, `lessOrEqual`, `greater`, `greaterOrEqual`

Property (as input)	Description
(required)	
`thresholdValue` (required)	The value for which the alert should be triggered.
`dataPointType` (required)	The supported values are numeric.
`resourceURI` (required)	The report URI on which the data point is present.

Managing a Single Alert

You can use the following REST APIs to add, modify, view, and delete single alerts:

- Creating a Single Alert
- Managing a Single Alert
- Modifying a Single Alert
- Deleting a Single Alert

Creating a Single Alert

Contrary to REST conventions, the alerts service uses the PUT method to create an alert and the POST method to modify an alert. To create an alert in the report, specify its properties in an alert descriptor and use the PUT method of the alerts service. Specify the repository path to the resource being scheduled inside the alert descriptor.

Only the authorized user who sends the alert details in the request becomes the owner of the alert, who can view the alert and receive notification.

Method	**URL**
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts

Content-Type	**Content**
application/alert+json	A well-formed JSON alert descriptor as described in The alert Descriptor. Specify only relevant properties in the alert descriptor and do not include any null properties.

Options
accept: application/alert+json

Return Value on Success	**Typical Return Values on Failure**

201 Created - The body contains the alert descriptor of the newly created alert. It is similar to the one that was sent but now contains the alertID for the new alert and other default property values. The body of the response is shown below.	404 Not Found - When no alert is found for the report in the server.

The following is an example of the request body for creating an alert in JSON:

```
{
    "label": "Alert PM Excel TestNew4",
    "trigger": {
        "simpleTrigger": {
            "timezone": "America/Los_Angeles",
            "startType": 1,
            "startDate": null,
            "endDate": null,
            "occurrenceCount": -1,
            "recurrenceInterval": 1,
            "recurrenceIntervalUnit": "HOUR"
        }
    },
    "baseOutputFilename": "RevenueDetailReport",
    "source": {
        "reportUnitURI": "/public/Samples/Reports/RevenueDetailReport",
        "parameters": {
            "parameterValues": {
                "ProductFamily": [
                    "Drink",
                    "Food",
                    "Non-Consumable"
                ]
            }
        }
    },
    "outputFormats": {
        "outputFormat": [
            "PDF"
        ]
    },
    "repositoryDestination": {
        "overwriteFiles": true,
        "sequentialFilenames": false,
        "folderURI": "/public/Samples/Reports",
        "saveToRepository": false,
        "timestampPattern": null,
        "outputFTPInfo": {
            "userName": "anonymous",
            "password": "",
            "folderPath": null,
            "serverName": null,
            "type": "ftp",
            "port": 21,
            "propertiesMap": {}
        }
    },
    "mailNotification": {
        "toAddresses": {
            "address": [
                "xyz@tibco.com"
            ]
        },
        "messageText": "test",
        "resultSendType": "SEND_ATTACHMENT",
        "skipEmptyReports": false,
        "skipNotificationWhenJobFails": false,
        "subject": "Test Now1"
    },
    "dataPointAlert": {
        "name": "AlertUI Data",
        "dataPoint": {
            "elementUUID": "72a3ea24-62a2-4f04-a5a6-d501046ddbcc"
        },
        "operator": "less",
        "thresholdValue": 6800,
        "dataPointType": "NUMERIC",
        "resourceURI": "/public/Samples/Reports/RevenueDetailReport"
    },
    "usingDefaultReportOutputFolderURI": false,
    "outputTimeZone": "America/Los_Angeles"
}
```

Creating an alert with Minute value is accepted and supported only in the REST API, but this causes performance issues in the application. The available options in UI for creating an alert are in Hours, Days, or Weeks.

The following is an example of the response body for creating an alert in JSON:

```
{
    "id": 2582,
    "version": 0,
    "username": "superuser",
    "label": "Alert PM Excel TestNew4",
    "creationDate": "2023-10-17T22:27:39.022-07:00",
    "trigger": {
        "simpleTrigger": {
            "id": 2579,
            "version": 0,
            "timezone": "America/Los_Angeles",
            "startType": 1,
            "misfireInstruction": 0,
            "occurrenceCount": -1,
            "recurrenceInterval": 1,
            "recurrenceIntervalUnit": "HOUR"
        }
    },
    "baseOutputFilename": "RevenueDetailReport",
    "exportType": "DEFAULT",
    "outputLocale": "en_US",
    "outputTimeZone": "America/Los_Angeles",
    "source": {
        "parameters": {
            "parameterValues": {
                "ProductFamily": [
                    "Drink",
                    "Food",
                    "Non-Consumable"
                ]
            }
        },
        "reportUnitURI": "/public/Samples/Reports/RevenueDetailReport"
    },
    "outputFormats": {
        "outputFormat": [
            "PDF"
        ]
    },
    "repositoryDestination": {
        "folderURI": "/public/Samples/Reports",
        "id": 2580,
        "overwriteFiles": true,
        "sequentialFilenames": false,
        "version": 0,
        "saveToRepository": false,
        "usingDefaultReportOutputFolderURI": false,
        "outputFTPInfo": {
            "userName": "anonymous",
            "password": "",
            "type": "ftp",
            "port": 21,
            "implicit": true,
            "pbsz": 0,
            "propertiesMap": {}
        }
    },
    "mailNotification": {
        "bccAddresses": {
            "address": []
        },
        "ccAddresses": {
            "address": []
        },
        "toAddresses": {
            "address": [
                "xyz@tibco.com"
            ]
        },
        "version": 0,
        "id": 2581,
        "includingStackTraceWhenJobFails": false,
        "messageText": "test",
        "resultSendType": "SEND_ATTACHMENT",
        "skipEmptyReports": false,
        "skipNotificationWhenJobFails": false,
        "subject": "Test Now1"
    },
    "dataPointAlert": {
        "id": 2583,
        "version": 0,
        "name": "AlertUI Data",
        "dataPoint": {
            "elementUUID": "72a3ea24-62a2-4f04-a5a6-d501046ddbcc"
        },
        "operator": "less",
        "thresholdValue": "6800",
        "dataPointType": "NUMERIC",
        "resourceURI": "/public/Samples/Reports/RevenueDetailReport"
    }
}
```

The response of the PUT request is the descriptor of the newly created alert, similar to the result of the GET request shown in Viewing an Alert Definition. It includes all the properties of the alert descriptor, including the server-assigned ID and all the null properties.

Modifying a Single Alert

Contrary to REST conventions, the alert service uses the PUT method to create an alert and the POST method to modify an alert. To modify an existing alert definition, use the GET method to read its alert descriptor, modify the desired properties, then use the following POST method:

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/alert/{alertId}

Content-Type	Content
application/alert+json	A complete well-formed JSON alert descriptor, as described in The alert Descriptor. It may include null properties and other default values. These values are ignored when using the descriptor as input, as shown in the result of Viewing a Alert Definition.

Options
accept: application/alert+json

Return Value on Success	Typical Return Values on Failure
200 OK - The response includes the complete alert descriptor updated with the submitted changes.	404 Not Found - When the specified <alertId> does not exist or the alert is no longer active.

The following is an example of the request body for updating an alert in JSON:

```
{
    "id": null,
    "version": null,
    "username": "superuser",
    "label": "dafc4_AlertingUpdateAlertRestAlert for report without IC_
Updated",
    "description": "Alert Description for report without IC_Updated",
    "creationDate": null,
    "trigger": {
        "simpleTrigger": {
            "id": null,
            "version": null,
            "timezone": null,
            "calendarName": null,
            "startType": 1,
            "startDate": null,
            "endDate": null,
            "misfireInstruction": null,
            "occurrenceCount": 2,
            "recurrenceInterval": 1,
            "recurrenceIntervalUnit": "HOUR"
        }
    },
    "baseOutputFilename": "dafc4_
AlertingUpdateAlertRestOutput1697608816923",
    "exportType": null,
    "outputLocale": null,
    "outputTimeZone": "America/Los_Angeles",
    "lastError": null,
    "lastAlerted": null,
    "outputFormats": {
        "outputFormat": [
            "PDF"
        ]
    },
    "repositoryDestination": {
        "folderURI": "/organizations/organization_1/qa_
automation/Alerting/0df89_AlertingUpdateAlertRest_Folder",
        "id": null,
        "outputDescription": "Report Output for alert test",
        "overwriteFiles": false,
        "sequentialFilenames": false,
        "version": null,
        "timestampPattern": null,
        "saveToRepository": false,
        "defaultReportOutputFolderURI": null,
        "usingDefaultReportOutputFolderURI": false,
        "outputLocalFolder": null,
        "outputFTPInfo": {
            "userName": "anonymous",
            "password": "",
            "folderPath": null,
            "serverName": null,
            "type": "ftps",
            "protocol": null,
            "port": 990,
            "implicit": true,
            "pbsz": 0,
            "prot": null,
            "propertiesMap": {},
            "sshKey": null,
            "sshPassphrase": null
        }
    },
    "mailNotification": {
        "bccAddresses": null,
        "ccAddresses": null,
        "toAddresses": {
            "address": [
                "xyz@tibco.com"
            ]
        },
        "version": null,
        "id": null,
        "includingStackTraceWhenJobFails": null,
        "messageText": "infra-platforms-na2-26210-
garsingh.pfa.jaspersoft.com: Test alert notification
messageAlertingUpdateAlertResttest_updateAlertNameAndDescription_
verifyAlertUpdated_29018",
        "resultSendType": "SEND_ATTACHMENT",
        "skipEmptyReports": false,
        "skipNotificationWhenJobFails": false,
        "subject": "infra-platforms-na2-26210-
garsingh.pfa.jaspersoft.com: Alert test email subject_a6bcf",
        "messageTextWhenJobFails": null
    },
    "systemNotification": null,
    "dataPointAlert": {
        "id": null,
        "version": null,
        "name": "Day is greater than 2",
        "dataPoint": {
            "elementUUID": "ce4c285c-0772-451e-bb87-18c24f09b229"
        },
        "operator": "greater",
        "thresholdValue": "2",
        "dataPointType": "NUMERIC",
        "resourceURI": "/organizations/organization_1/qa_
automation/Reports/Jrxml_reports/DaysSimpleReport"
    }
}
```

The following is an example of the response body for updating an alert in JSON:

```
{
    “alertsId”: [
        4263
    ]
}
```

Deleting a Single Alert

Use the DELETE method to remove the alert from the report. The below form deletes a single alert.

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts/{alertsId}

Return Value on Success	Typical Return Values on Failure
200 OK - The body contains the ID of the deleted alert.	404 Not Found - When the specified alert is not found in the server or the alert is no longer active.

Managing Multiple Alerts

You can use the following REST APIs to modify, pause, resume, restart, and delete multiple alerts:

- Modifying Multiple Alerts
- Pausing Multiple Alerts
- Resuming Multiple Alerts
- Restarting Multiple Alerts
- Deleting Multiple Alerts

Modifying Multiple Alerts

Contrary to REST conventions, the alert service uses the PUT method to create an alert and the POST method to modify multiple alerts. Typical user cannot modify the alert of another user but the system admin (`superuser`) can modify the alert of all users. Organization admin (`jasperadmin`) can modify only if the users and jasperadmin belong to the same organization. To modify an existing alert definition, use the GET method to read its alert descriptor, modify the desired properties, then use the following POST method:

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts?id={ID_1}&id={ID_2}&...&id={ID_n}

Content-Type	Content
application/alert+json	A complete well-formed JSON alert descriptor, as described in The alert Descriptor. It may include null properties and other default values. These are ignored when using the descriptor as input, as shown in the result of Viewing a Alert Definition.

Options

accept: application/alert+json	
Return Value on Success	**Typical Return Values on Failure**
200 OK - The response includes the complete alerts descriptor updated with the submitted changes.	404 Not Found - When the specified <alertId> does not exist or the alerts are no longer active.

The following is an example of the request body for updating multiple alerts in JSON:

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

The following is an example of the response body for the list of updated alert IDs in JSON:

```
{"alertId":[5594,5645]}
```

Pausing Multiple Alerts

The following method pauses currently scheduled alert execution, also called as disabled in the user interface. Pausing keeps the alert schedule and all other details but prevents the alert from running. It does not delete the alert.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts/pause/

Content-Type	Content
application/json	An array or list of alert IDs to pause. See the example below. If

	the body of the request is empty, or the list is empty, all alerts in the scheduler is paused.
Options	
accept: application/json	

Return Value on Success	Typical Return Values on Failure
200 OK - The array or list of alerts that were paused. Alerts specified with the <alertID> do not exist and are ignored without any error.	

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

```
{
    "alertsId": [
        5805
    ]
}
```

Resuming Multiple Alerts

Use the following method to resume all paused alerts. On the resume of an alert any defined trigger that occurs after the time it is resumed causes the report to run again. Missed triggers that occur before the alert is resumed are never run.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts/resume/

Content-Type	Content
application/json	An array or list of alert IDs to resume. See the example of request and response below. If the body of the request is empty, or the list is empty, all paused alerts are resumed.

Options
accept: application/json

Return Value on Success	Typical Return Values on Failure
200 OK - The array or list of alerts that were resumed. Alerts specified with the <alertID> do not exist and are ignored without any error.	

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

```
{
    "alertsId": [
        5805
    ]
}
```

Restarting Multiple Alerts

Use the following method to rerun failed alerts. For each alert to be restarted, this method creates an immediate single-run copy of the alert, to replace the one that failed. Therefore, all alerts listed in the request body 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/alerts/restart/

Content-Type	Content
application/json	An array or list of alert IDs to restart. See the example of request and response below.

Options	
accept: application/json	
Return Value on Success	**Typical Return Values on Failure**
200 OK - The array or list of alerts that were restarted.	

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

```
{
    "alertsId": [
        5805
    ]
}
```

Deleting Multiple Alerts

Use the DELETE method to remove multiple alerts from the report. The below form deletes multiple alerts.

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/alerts

Return Value on Success	Typical Return Values on Failure
200 OK - The body contains the ID of the deleted alert.	404 Not Found - When the specified alert is not found in the server or the alert is no longer active.

The following is an example of the request body for deleting an alert in JSON:

```
{
    "alertsId": [
        5330,
        5350
    ]
}
```

The following is an example of the response body for deleting an alert in JSON:

```
{
    "alertsId": [
        5330,
        5350
    ]
}
```

The jobs calendars Services

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 may not want to run it on holidays. The list of excluded days and times is called a calendar, and a calendar may be defined as a list of annual dates, a weekly or monthly pattern, or a cron expression.

The rest_v2/jobs/calendars service defines any number of exclusion calendars that are stored in the repository. 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 modify an exclusion calendar and update all of the report jobs that used it. Therefore, you can update the calendar of holidays every year and not need to modify any report jobs.

This chapter includes the following sections:

Creating an Exclusion Calendar

The PUT method creates a named exclusion calendar that you can use when scheduling reports. Specify a unique name for the calendar in the URL. The body of the request determines the type of the calendar, as shown in the examples below the table.

Method	URL

PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/calendars/<calendarName>

Content-Type	Content
application/xml application/json	A well-formed XML or JSON calendar descriptor (see examples below).

Return Value on Success	Typical Return Values on Failure
200 OK – The calendar is created, and the body of the response contains the calendar definition, similar to the one that was sent.	400 Bad Request – When the calendar name already exists or the descriptor is missing a parameter (the error message describes the missing parameter).

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.

  JSON:

```
{
    "calendarType":"annual",
    "description":"Annual calendar description",
    "excludeDays": [ "2021-03-20", "2021-03-21", "2021-03-22"],
    "timeZone":"GMT+03:00"
}
```

  XML:

```
<?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>2021-03-20</excludeDay>
    <excludeDay>2021-03-21</excludeDay>
    <excludeDay>2021-03-22</excludeDay>
  </excludeDays>
</reportJobCalendar>
```

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

  JSON:

  ```
  {
      "calendarType":"cron",
      "description":"Cron calendar description",
      "cronExpression":"0 30 10-13 ? * WED,FRI",
      "timeZone":"GMT+03:00"
  }
  ```

  XML:

  ```
  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <reportJobCalendar>
    <calendarType>cron</calendarType>
    <description>Cron calendar 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.

  JSON:

  ```
  {
      "calendarType":"daily",
      "description":"Daily calendar description",
      "invertTimeRange":false,
      "rangeEndingCalendar":"2020-20T14:44:37.353+03:00",
      "rangeStartingCalendar":"2020-03-20T14:43:37.353+03:00",
      "timeZone":"GMT+03:00"
  }
  ```

XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<reportJobCalendar>
  <calendarType>daily</calendarType>
  <description>Daily calendar description</description>
  <invertTimeRange>false</invertTimeRange>
  <rangeEndingCalendar>2020-03-
20T14:44:37.353+03:00</rangeEndingCalendar>
  <rangeStartingCalendar>2020-03-
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.

  JSON:

```
{
    "calendarType":"holiday",
    "description":"Holiday calendar (observed)",
    "excludeDays": [
        "2020-01-01",
        "2020-01-20",
        "2020-02-17",
        "2020-05-25",
        "2020-07-03",
        "2020-09-07",
        "2020-10-12",
        "2020-11-11",
        "2020-11-26",
        "2020-12-25"
    ],
    "timeZone":"GMT+03:00"
}
```

XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<reportJobCalendar>
  <calendarType>holiday</calendarType>
  <description>Holiday calendar (observed)</description>
  <excludeDays>
    <excludeDay>2021-03-20</excludeDay>
    <excludeDay>2020-01-01</excludeDay>
    <excludeDay>2020-01-20</excludeDay>
    <excludeDay>2020-02-17</excludeDay>
    <excludeDay>2020-05-25</excludeDay>
    <excludeDay>2020-07-03</excludeDay>
    <excludeDay>2020-09-07</excludeDay>
    <excludeDay>2020-10-12</excludeDay>
    <excludeDay>2020-11-11</excludeDay>
    <excludeDay>2020-11-26</excludeDay>
    <excludeDay>2020-12-25</excludeDay>
  </excludeDays>
  <timeZone>GMT+03:00</timeZone>
</reportJobCalendar>
```

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

  JSON:

```
{
    "calendarType": "weekly",
    "description": "Weekly calendar description",
    "excludeDaysFlags": [
        true,  /*Sunday*/
        false, /*Monday*/
        false, /*Tuesday*/
        false, /*Wednesday*/
        false, /*Thursday*/
        false, /*Friday*/
        false  /*Saturday*/
    ],
    "timeZone": "GMT+03:00"
}
```

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

  JSON:

```
{
    "calendarType":"monthly",
    "description":"Monthly calendar description",
    "excludeDaysFlags": [
        true,  /* 1*/
        false, /* 2*/
        false, /* 3*/
        false, /* 4*/
        false, /* 5*/
        false, /* 6*/
        false, /* 7*/
        false, /* 8*/
        false, /* 9*/
        false, /*10*/
        false, /*11*/
        false, /*12*/
        false, /*13*/
        false, /*14*/
        false, /*15*/
        false, /*16*/
        false, /*17*/
        false, /*18*/
        false, /*19*/
        false, /*20*/
        false, /*21*/
        false, /*22*/
        false, /*23*/
        false, /*24*/
        false, /*25*/
        false, /*26*/
        false, /*27*/
        false, /*28*/
        false, /*29*/
        false, /*30*/
        false  /*31*/
    ],
    "timeZone":"GMT+03:00"
}
```

Listing All 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/?<argument>	
Argument	**Type/Value**	**Description**
calendar Type	optional string	A type of calendar to return: annual, cron, daily, holiday, monthly, or weekly. You may specify only one calendarType parameter. When calendarType isn't specified, all calendars names are returned. If calendarType has an invalid value, an empty collection is returned.

Return Value on Success	Typical Return Values on Failure
200 OK – Body contains a list of calendar names.	401 Unauthorized

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

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

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 – The body contains the definition of the requested calendar.	404 Not Found – When the specified calendar name does not exist.

The calendar descriptor in a successful response has the following JSON format:

- Annual calendar:

```
{
    "calendarType": "annual",
    "description": "Annual calendar description",
    "timeZone": "GMT+03:00",
    "excludeDays": [
        "2012-03-20",
        "2012-03-21",
        "2012-03-22"
    ]
}
```

- Cron calendar:

```
{
    "calendarType": "cron",
    "description": "Cron calendar description",
    "timeZone": "GMT+03:00",
    "excludeDays": null,
    "cronExpression": "0 30 10-13 ? * WED,FRI"
}
```

- Daily calendar:

```
{
    "calendarType": "daily",
    "description": "Daily calendar description",
    "timeZone": "GMT+03:00",
    "excludeDays": null,
    "rangeStartingCalendar": 1332243817353,
    "rangeEndingCalendar": 1332243877353,
    "invertTimeRange": false
}
```

- Holiday calendar:

```
{
    "calendarType": "holiday",
    "description": "Holiday calendar (observed)",
    "timeZone": "GMT+03:00",
    "excludeDays": [
        "2020-01-01",
        "2020-01-20",
        "2020-02-17",
        "2020-05-25",
        "2020-07-03",
        "2020-09-07",
        "2020-10-12",
        "2020-11-11",
        "2020-11-26",
        "2020-12-25"
    ]
}
```

- Weekly calendar (day flags are Sunday to Saturday):

```
{
    "calendarType": "weekly",
    "description": "Weekly calendar description",
    "excludeDays": null,
    "excludeDaysFlags": [
        true,
        false,
        false,
        false,
        false,
        false,
        false
    ],
    "timeZone":"GMT+03:00"
}
```

- Monthly calendar (day flags are dates from 1 to 31):

```
{
    "calendarType":"monthly",
    "description":"Monthly calendar description",
    "excludeDaysFlags": [
        true,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false
    ],
    "timeZone":"GMT+03:00"
}
```

Updating an Exclusion Calendar

Use the PUT method to update a calendar that already exists, with the option to update all the jobs that use it.

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/calendars/<calendarName>?<args>	
Argument	**Type/Value**	**Description**
replace?	true	Set to true to modify an existing calendar with the given name. When this argument is omitted or false, an error is returned (see below).
update Triggers?	true / false	Whether or not to update existing triggers that reference this calendar. When triggers are updated, the new calendar is in effect on existing scheduled reports.
Content-Type		**Content**
application/xml application/json		A well-formed XML or JSON calendar descriptor. See Creating an Exclusion Calendar for examples of each type of calendar. You can specify any type of exclusion calendar such as weekly, monthly, or cron, regardless of the current type.

Return Value on Success	Typical Return Values on Failure
200 OK – The calendar is updated, and the body of the response contains the new calendar definition, similar to the one that was sent.	400 Bad Request – When the replace parameter is false or omitted, or the calendar definition is not valid. 404 Not Found – When the specified calendar name does not exist.

For example, you can make the following request to replace the calendar named weeklyCalendar. Note that the calendar name does not change, and it will contain a daily calendar, which is not good naming practice.

Request	`PUT http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/calendars/weeklyCalendar?replace=true&updateTriggers=true`
	`Content-Type=application/json`

Body	`{ "calendarType":"daily", "description":"test description", "invertTimeRange":false, "rangeEndingCalendar":"2012-03-20T14:44:37.353+03:00", "rangeStartingCalendar":"2012-03-20T14:43:37.353+03:00", "timeZone":"GMT+03:00" }`

If the replace parameter is false or omitted, the error is as follows:

Response	400 Bad Request
Body	`{ "message": "Resource 'weeklyCalendar' already exists", "errorCode": "resource.already.exists", "parameters": [ "weeklyCalendar" ] }`

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.

Error Messages

When creating or updating a calendar, the error messages can be expected in the following cases.

- Creating an annual calendar that is missing a mandatory parameter:

Request	`PUT http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/calendars/annualCalendar` `Content-Type=application/json`
Body	`{ "calendarType":"annual", "description":"Annual calendar description", "timeZone":"GMT+03:00" }`

```
{
    "calendarType":"annual",
    "description":"Annual calendar description",
    "timeZone":"GMT+03:00"
}
```

Expected Reply:

Response	400 Bad Request
Body	(see below)

```
{
    "message": "mandatory parameter
'reportJobCalendar.excludeDays' not found",
    "errorCode": "mandatory.parameter.error",
    "parameters":
    [
        "reportJobCalendar.excludeDays"
    ]
}
```

- Creating a cron calendar that is missing a mandatory parameter:

Request

```
PUT http://<host>:<port>/jasperserver[-pro]/rest_
v2/jobs/calendars/cronCalendar
Content-Type=application/json
```

Body

```
{
    "calendarType":"cron",
    "description":"Cron calendar description",
    "timeZone":"GMT+03:00"
}
```

Expected Reply:

Response 400 Bad Request

Body

```
{
    "message": "mandatory parameter
'reportJobCalendar.cronExpression' not found",
    "errorCode": "mandatory.parameter.error",
    "parameters":
    [
        "reportJobCalendar.cronExpression"
    ]
}
```

- Creating a daily calendar that is missing the mandatory start-range parameter:

Request	`PUT http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/calendars/dailyCalendar`
	`Content-Type=application/json`

Body	see below

```
{
    "calendarType":"daily",
    "description":"Daily calendar description",
    "invertTimeRange":false,
    "rangeEndingCalendar":"2021-03-20T14:44:37.353+03:00",
    "timeZone":"GMT+03:00"
}
```

Expected Reply:

Response	400 Bad Request
Body	see below

```
{
    "message": "mandatory parameter 
'reportJobCalendar.rangeStartingCalendar' not found",
    "errorCode": "mandatory.parameter.error",
    "parameters":
    [
        "reportJobCalendar.rangeStartingCalendar"
    ]
}
```

- Creating a daily calendar that is missing the mandatory end-range parameter:

Request	`PUT http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/calendars/dailyCalendar`
	`Content-Type=application/json`

Body	`{ "calendarType":"daily", "description":"Daily calendar description", "invertTimeRange":false, "rangeStartingCalendar":"2012-03-20T14:43:37.353+03:00", "timeZone":"GMT+03:00" }`

```
{
    "calendarType":"daily",
    "description":"Daily calendar description",
    "invertTimeRange":false,
    "rangeStartingCalendar":"2012-03-
20T14:43:37.353+03:00",
    "timeZone":"GMT+03:00"
}
```

Expected Reply:

Response	400 Bad Request
Body	(see below)

```
{
    "message": "mandatory parameter
'reportJobCalendar.rangeEndingCalendar' not found",
    "errorCode": "mandatory.parameter.error",
    "parameters":
    [
        "reportJobCalendar.rangeEndingCalendar"
    ]
}
```

- Creating a holiday calendar that is missing a mandatory parameter:

Request

```
PUT http://<host>:<port>/jasperserver[-pro]/rest_
v2/jobs/calendars/holidayCalendar
Content-Type=application/json
```

Body

```
{
    "calendarType":"holiday",
    "description":"Holiday calendar description",
    "timeZone":"GMT+03:00"
}
```

- Expected Reply:

Response	400 Bad Request

Body	
	```
	{
	    "message": "mandatory parameter
	'reportJobCalendar.excludeDays' not found",
	    "errorCode": "mandatory.parameter.error",
	    "parameters":
	    [
	        "reportJobCalendar.excludeDays"
	    ]
	}
	```

- Creating a weekly calendar that is missing a mandatory parameter:

Request	
	```
	PUT http://<host>:<port>/jasperserver[-pro]/rest_
	v2/jobs/calendars/weeklyCalendar
	Content-Type=application/json
	```

Body	
	```
	{
	    "calendarType":"weekly",
	    "description":"Weekly calendar description",
	    "timeZone":"GMT+03:00"
	}
	```

Expected Reply:

Response	400 Bad Request

Body	
	```
	{
	    "message": "mandatory parameter
	'reportJobCalendar.excludeDaysFlags' not found",
	    "errorCode": "mandatory.parameter.error",
	```

```
    "parameters":
    [
         "reportJobCalendar.excludeDaysFlags"
    ]
}
```

- Creating a monthly calendar that is missing a mandatory parameter:

Request	`PUT http://<host>:<port>/jasperserver[-pro]/rest_v2/jobs/calendars/monthlyCalendar`
	`Content-Type=application/json`

Body	`{ "calendarType":"monthly", "description":"Monthly calendar description", "timeZone":"GMT+03:00" }`

```
{
    "calendarType":"monthly",
    "description":"Monthly calendar description",
    "timeZone":"GMT+03:00"
}
```

Expected Reply:

Response	400 Bad Request
Body	see below

```
{
    "message": "mandatory parameter
'reportJobCalendar.excludeDaysFlags' not found",
    "errorCode": "mandatory.parameter.error",
    "parameters":
    [
        "reportJobCalendar.excludeDaysFlags"
    ]
}
```

The alerts calendars Service

The rest_v2/alerts/calendars service is the new service that defines any number of exclusion calendars that are stored in the repository.

Using the rest_v2/alerts/calendars service, you can perform various actions on the list of excluded days or times when you do not want the alert to be scheduled.

The rest_v2/alerts/calendars service reimplements the functionality of the rest_v2/jobs/calendars service implementation. The implementation methods for creating, listing, viewing, updating, and deleting alerts calendars service are similar to the jobs calendar service. For more information on the implementation of alerts calendars service functionality, see The jobs calendars Services.

The queryExecutor Service

In addition to running reports, JasperReports Server exposes queries that you can run through the rest_v2/queryExecutor service. The only resource that supports these 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 shows 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>
```

The following sample shows the result of above query. 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.

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. 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.

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 suborganizations.

This chapter includes the following sections:

Searching for Organizations

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

Method	URL	
GET	http://<host>:<port>/jasperserver-pro/rest_v2/organizations?<arguments>	
Argument	**Type**	**Description**
q	Optional String	Specify a string or substring to match the organization ID, alias, or name of any organization. The search is not case-sensitive. Only the matching organizations are returned in the results, regardless of their hierarchy.
include Parents	Optional Boolean	When used with a search, the result includes 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 tenantFolderURI 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 the 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</tenantFo
lderUri>
    <tenantName>Accounts</tenantName>
    <tenantUri>/Finance/Accounts</tenantUri>
    <theme>default</theme>
  </organization>
</organizations>
```

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 an 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"
}
```

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 (joeuser, jasperadmin) in the new organization. When not specified, the default behavior is true and organizations are created with the standard default users.

Content-Type	Content
application/xml application/json	A partial or complete organization descriptor that includes the desired properties for the organization.

Return Value on Success	Typical Return Values on Failure
201 Created - The organization was successfully created using the values in the descriptor or default values if missing.	404 Not Found - When the ID of the parent organization cannot be resolved. 400 Bad Request - When the ID or alias of the new organization is not unique on the server, or when the ID in the description contains illegal symbols. The following symbols are not allowed: id and alias: ~!+-#$%^| tenantName: |&*?<>/\

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

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

However, all properties have defaults or can be determined based on the alias value. The minimal descriptor necessary to create an organization is simply the alias property. In this case, the organization is created as a child of the logged-in user’s home organization. For example, if the 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"
}
```

Modifying Organization Properties

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

Method	URL
PUT	http://<host>:<port>/jasperserver-pro/rest_v2/organizations/organizationID/

Content-Type	Content
application/xml application/json	A partial organization descriptor that includes the properties to change. Do not specify the following properties: id - The organization ID is permanent and can never be modified.

	parentId - Organizations cannot change parents. tenantUri - Organizations cannot change the organization hierarchy. tenantFolderUri - The organization folder is automatically based on its parent, which cannot be changed.

Return Value on Success	**Typical Return Values on Failure**
200 OK – The organization was successfully updated.	400 Bad Request – When some dependent resources cannot be resolved.

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

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

Setting the Theme of an Organization

A theme determines how the JasperReports Server interface appears to users. The 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 the themes, see the JasperReports Server Administrator Guide.

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 suborganizations, 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.

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

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. As shown in the following example, the same username may exist in different organizations:

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

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

Viewing a User

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

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

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID
	http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID

Options	
accept: application/xml (default) accept: application/json	
Return Value on Success	**Typical Return Values on Failure**
200 OK – The content is the descriptor for the given user.	404 Not Found – When the user ID or organization ID does not match any user or organization. The content includes an error message.

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

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

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

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

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

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

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 Modifying User Properties.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID

Content-Type	Content
application/xml application/json	A user descriptor that includes at least the fullName and password for the user. Even if the username and tenantID are specified in the descriptor, their values in the URL take precedence. See the table below for other properties in the user descriptor.

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 user descriptor includes the following properties when being sent for creating or updating a user:

Property	Description
username	This can be omitted because the userID in the URL takes precedence. If a value is specified, it has no effect.
tenantID	This can be omitted because the orgID in the URL takes precedence. If a value is specified, it has no effect.
fullName	A string giving the name of the user that appears in the user interface, for example Joe User. This property is required when creating a user.
password	A string giving the password of the user. This property is required when creating a user.
emailAddress	The email address is optional, but when specified, the user can receive notifications such as when a scheduled report is available.
roles	The roles are optional because ROLE_USER is automatically assigned to all users and does not need to be specified. To assign further roles to the user, specify an array of roles with the name and tenantId properties. See the example below.
enabled	When false, the user cannot log in. If this property is omitted during user creation, its value is true (enabled) by default. Set this value to false if you want to create the user but not allow access to the server yet.

Property	Description
previousPasswordChangeTime	This property is set automatically by the server. Any value submitted in the descriptor is ignored.
externallyDefined	The externallyDefined property is true when the user is synchronized from a 3rd party such as an LDAP directory or single sign-on. When creating a user through the REST API, this property should be set to false. For more information, see the JasperReports Server External Authentication Cookbook.

The following example shows the user descriptor in JSON format:

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

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. See the properties of the descriptor in the table above.

- 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 Creating a User.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/users/userID http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/users/userID

Content-Type	Content
application/xml application/json	A user descriptor that includes the properties you want to change. The username and tenantID properties have no effect in the descriptor, their values in the URL always take precedence. For other properties, see the table in Creating a User.

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"}]
}
```

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.

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 suborganizations.

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

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 belongs to the role. If no search is specified, it returns all roles. The method has two forms:

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

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/roles?<arguments> http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/roles?<arguments>

Argument	Type	Description
search	Optional String	Specify a string or substring to match the role ID (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.	404 Not Found - When the organization ID does not match any organization. The content includes an error message.

204 No Content - The search did not return any roles.

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.

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 the 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"
}
```

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	http://<host>:<port>/jasperserver[-pro]/rest_v2/roles/roleID http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/roles/roleID

Content-Type	Content
application/xml application/json	An empty role descriptor, either <role></role> or {}. The role descriptor has the following properties, but they are not needed: name - The roleID value in the URL takes precedence, and any value in the descriptor is ignored. tenantID - The orgID value in the URL takes precedence, and any value in the descriptor is ignored. externallyDefined - The externallyDefined property is true when the role is synchronized from a 3rd party such as an LDAP directory. If you omit this property, its default value is false. If you include it, it must be set to false.

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.

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, which is also its roleID. 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.

Only repository permissions based on the role are updated to the new role name. If you have domain security files based on the role name, they must be updated manually by uploading a modified security file. For more information, see the JasperReports Server Data Management Using Domains.

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	A role descriptor that needs only the name property:

application/json	name - The new name for the role, which also becomes its new roleID. tenantID - The orgID value in the URL takes precedence, and any value in the descriptor is ignored. externallyDefined - The externallyDefined property is true when the role is synchronized from a 3rd party such as an LDAP directory. If you omit this property, its default value is false. If you include it, it must be set to false.

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.

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 Modifying User Properties.

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 the roles of the root organization.

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

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/roles/roleID http://<host>:<port>/jasperserver[-pro]/rest_v2/organizations/orgID/roles/roleID

Return Value on Success	Typical Return Values on Failure
204 No Content - The role was successfully deleted.	404 Not Found - When the ID of the organization cannot be resolved.

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 to any value at any time. When running dashboards, views, or reports, certain advanced features of the server 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 to 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 suborganizations. 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:

- Deleting Attributes

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 they belong to have 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 Referencing Attributes can take advantage of this to implement default values.

Secure Attributes

JasperReports Server 6.0 also introduced the notion of the 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 and 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.

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.

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 cannot 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 suborganizations, and the users in any suborganizations.
- Organization admins cannot 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 suborganization 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).

Referencing Attributes

As mentioned, several internal mechanisms of the server read attributes on users and organizations to use 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 lever (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 then determines the entities where you need to create attributes and the values of those attributes.

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	The attribute's name is longer than 255 characters.
too_long_value	The attribute's value is longer than 255 characters.
empty_name	The attribute's name is empty or contains only whitespaces.
empty_value	The 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.

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 Entities with Attributes.

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/organzation_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"
}
```

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 Entities with Attributes.

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
201 Created - When the attributes were successfully created on the given entity. 200 OK - When the attributes were successfully updated.	404 Not Found - When the user ID or organization ID does not match any user or organization. The content includes an error message. 400 Bad Request - When an attribute name or value is null, blank, or too long (see Attribute Limitations). If one attribute causes an error, the operation stops and returns an error, but the attributes that were already set remain.

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

Content-Type	Content
application/xml application/json	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.

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"
}
```

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 Entities with Attributes.

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

Method	**URL**
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/<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 Attribute Limitations). If one attribute causes an error, the operation stops and returns an error, but the 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.	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. 400 Bad Request - When an attribute name is null, blank, or too long (see Attribute Limitations).

Jaspersoft Documentation and Support Services

For information about this product, you can read the documentation, contact Support, and join Jaspersoft Community.

How to Access Jaspersoft Documentation

Documentation for Jaspersoft products is available on the Product Documentation website, mainly in HTML and PDF formats.

The Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The documentation for this product is available on the JasperReports® Server Product Documentation page.

How to Access Related Third-Party Documentation

When working with JasperReports® Server, you may find it useful to read the documentation of the following third-party products:

How to Contact Support for Jaspersoft Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

How to Join Jaspersoft Community

Jaspersoft Community is the official channel for Jaspersoft customers, partners, and employee subject matter experts to share and access their collective experience. Jaspersoft Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from Jaspersoft products. In addition, users can submit and vote on feature requests from within the Jaspersoft Ideas Portal. For a free registration, go to Jaspersoft Community.

Legal and Third-Party Notices