TIBCO JasperReports® Server REST API Reference

Software Release 7.9

TIBCO®

Important Information

TABLE OF CONTENTS

CHAPTER 1 REST API OVERVIEW

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

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

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

Historically, the REST API is considered a web service, and JasperReports Server provided several other web services. The current REST API is the second version and all services use the rest_v2/ prefix. The first REST API with the rest/ prefix and the earlier SOAP API (Simple Object Access Protocol) are deprecated and no longer maintained. Although the server might still respond to deprecated services, they are not updated for new features of the server and are never 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:

1.1 List of Services

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

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

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

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

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

In all URLs in this API Reference:

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

The REST services are available at the following URLs:

Table 1-1 REST API Services and URLs

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

Web Service	URLs
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 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

1.2 Sending REST Requests from a Browser

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

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

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

```
X-REMOTE-DOMAIN: 1
```

1.3 HTTP Response Codes

JasperReports Server REST services return standard HTTP status codes. In case of an error, a detailed message may be present in the body as plain text. Client error codes are of type 4xx, while server errors are of type 5xx. The following table lists all the standard HTTP codes. Each service returns typical success and error messages that are given in the reference chapter for that service.

Table 1-2 HTTP Response Codes

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

1.4 Deprecated Web Services

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

Table 1-3 Deprecated REST v1 Services

Web Service	URLs
Login	http://<host>:<port>/jasperserver[-pro]/rest/login http://<host>:<port>/jasperserver[-pro]/j_spring_security_check
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.

Table 1-4 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

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

CHAPTER 2 THE serverInfo SERVICE

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

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

Method	**URL**
GET	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.

CHAPTER 3 AUTHENTICATION METHODS

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

This chapter includes the following sections:

3.1 Overview of REST Authentication

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

- 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 logout operation when done. Use of the login and logout 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 **3.2, "HTTP Basic Authentication," on page 18**.
- If a username or password contains UTF-8 characters, it may be corrupted by basic authentication and the services will always return an error. In this case, you can send the username and password in URL arguments with each request. See **3.3, "Argument-based Authentication," on page 19**.

- 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 **3.4, "The login Service," on page 20**.
- However, sessions are kept for 20 minutes by default, so if your client makes a request every 15 minutes with the same credentials, the corresponding session will be kept in memory indefinitely. This can be a problem if you have many different clients running large reports, because some report output is stored in the user session, and they can fill up the available memory. In this case, you should use the logout call to make sure the memory is freed. For more information, see **3.6, "Logout," on page 23**.

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 **3.3, "Argument-based Authentication," on page 19**.
- 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 **3.4, "The login Service," on page 20**.
- 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 may cause a 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 **3.4, "The login Service," on page 20**.
- If your server is configured with Pre-Authentication, specify the `pp` argument in every API request, as shown in **3.3, "Argument-based Authentication," on page 19**.

None of these authentication methods provide privacy, meaning that passwords are sent in plain text or easily reversed encodings. Tibco 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 **3.5, "Login Encryption (Deprecated)," on page 23**.

3.2 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 HTTP basic authentication as well.

If login encryption in enabled in your server, you must encrypt the password before base64-encoding it with the username. For more information about encryption, see **3.5, "Login Encryption (Deprecated)," on page 23**.

3.3 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 will fail without the password. If the server has login encryption enabled, the password must be encrypted as explained in **3.5, "Login Encryption (Deprecated)," on page 23**.
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.
userLocale	Java locale string	An optional argument to set the locale 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 locale is used.

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.

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 will return 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 arugument, 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 **3.5, "Login Encryption (Deprecated)," on page 23**.

3.4 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 **3.1, "Overview of REST Authentication," on page 17**.

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 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 Tibco recommends that you configure your server and clients to use HTTPS, or that you use the login encryption feature. For more information, see **3.5, "Login Encryption (Deprecated)," on page 23**.

Method	**URL**	
POST GET (config.)	http[s]://<host>:<port>/jasperserver[-pro]/**rest_v2/login**[?<arguments>] 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 will fail without the password. If the server has login encryption enabled, the password must be encrypted as explained in **3.5, "Login Encryption (Deprecated)," on page 23**.
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 configuration file WEB-INF/applicationContext-externalAuth-<sso>.xml.
userLocale	Java locale string	An optional argument to set the locale for this user session. 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.
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.
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 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 will fail 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 **3.6, "Logout," on page 23**. Closing the session frees up any associated resources in memory.

3.5 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}

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.

3.6 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	

CHAPTER 4 WORKING WITH RESOURCES

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

For further information, see:

- **Chapter 5, "Resource Descriptors," on page 33** for a reference to every type of resource and its attributes.
- **Chapter 6, "The resources Service," on page 47** for methods to operate on resources in the repository.

This chapter includes the following sections:

4.1 Resource URI

Resources (such as reports, images, queries, and data sources) are stored in the server's repository. The repository is organized like a file system, with a root and a hierarchical set of folders. Each object in the repository is considered a resource: a folder is a resource of type folder, a JRXML report is a resource of type reportUnit, and images are of type file.

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

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

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

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

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

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

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

4.2 Custom Media Types

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

application/repository.<resourceType>+<format>

where:

- <resourceType> is the name for each type of repository resource, such as reportUnit, dataType, or jdbcDataSource. The names of all supported types are given in **Chapter 5, "Resource Descriptors," on page 33**.
- <format> is the representation format of the descriptor, either json or xml.

For example:

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

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

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

4.3 Accept HTTP Headers

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

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

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

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

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

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

4.4 Content-Type HTTP Headers

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

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

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

Content-Type: application/repository.dataType+xml

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

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

4.5 JSON Format

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

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

4.6 Nested Resources

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

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

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

4.7 Referenced Resources

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

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

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

To create referenced resources, send requests to the server that contain the appropriate reference objects for the target resource. See **4.7, "Referenced Resources," on page 28** for the specific reference objects available in each resource descriptor.

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

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

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

4.8 Local Resources

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

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

Use nested descriptors such as the ones above to create resources that contain local resources. Descriptors can be nested to any level, as long as the syntax of each descriptor is valid. See **4.6, "Nested Resources," on page 28** for the correct syntax of both the parent and the nested resource.

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

<parentURI>_files/

and local resources can be accessed at the following URI:

<parentURI>_files/<resourceID>

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

/adhoc/topics/Cascading_multi_select_topic_files/Country_multi_select_files/country_query

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

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

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

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

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

4.9 Optimistic Locking

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

4.10 Update-only Passwords

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

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

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

CHAPTER 5 RESOURCE DESCRIPTORS

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

- **Chapter 4, "Working With Resources," on page 25** for general guidelines about using descriptors.
- **Chapter 6, "The resources Service," on page 47** 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:

5.1 Common Attributes

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

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

application/repository.{resourceType}+json	application/repository.{resourceType}+xml
{ "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" ... }	<?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}>

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

5.2 Folder

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

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

Only the label and description fields are writable.

5.3 JNDI Data Source

application/repository.jndiJdbcDataSource+json	application/repository.jndiJdbcDataSource+xml

{
{commonAttributes},
"jndiName":"{jndiName}",
"timezone":"{timezone}"
}
	<jndiDataSource>
	{commonAttributes}
	<jndiName>{jndiName}</jndiName>
	<timezone>{timezone}</timezone>
	</jndiDataSource>

5.4 JDBC Data Source

application/repository.jdbcDataSource+json	application/repository.jdbcDataSource+xml

{
{commonAttributes},
"driverClass":"{driverClass}",
"password":"{password}",
"username":"{username}",
"connectionUrl":"{connectionUrl}",
"timezone":"{timezone}"
}
	<jdbcDataSource>
	{commonAttributes}
	<driverClass>{driverClass}</driverClass>
	<password>{password}</password>
	<username>{username}</username>
	<connectionUrl>
	{connectionUrl}
	</connectionUrl>
	<timezone>{timezone}</timezone>
	</jdbcDataSource>

5.5 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
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

5.6 Virtual Data Source

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

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

5.7 Custom Data Source

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

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

5.8 Bean Data Source

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

5.9 Datatypes

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

5.10 List of Values

application/repository.listOfValues+json	application/repository.listOfValues+xml

{
{commonAttributes},
"items":[
{
"label":"{label}",
"value":"{value}"
},
...
]
}
	<listOfValues>
	{commonAttributes}
	<items>
	<item>
	<label>{label}</label>
	<value>{value}</value>
	</item>
	...
	</items>
	</listOfValues>

5.11 Query

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

application/repository.query+json	application/repository.query+xml

{
{commonAttributes},
"value":"{query}",
"language":"{language}",
"dataSource":{
"dataSourceReference": {
"uri":"{dataSourceUri}"
}
}
}
	<query>
	{commonAttributes}
	<value>{query}</value>
	<language>{language}</language>
	<dataSourceReference>
	<uri>{dataSourceUri}</uri>
	</dataSourceReference>
	</query>

5.12 Input Control

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

application/repository.inputControl+json	application/repository.inputControl+xml

<pre>{
    {commonAttributes},
    "mandatory":"{true|false}",
    "readOnly":"{true|false}",
    "visible":"{true|false}",
    "type":"{inputControlTypeByteValue}",
    "usedFields":"{field1;field2;...}",
    "dataType": {
        "dataTypeReference": {
            "uri": "{dataTypeResourceUri}"
        }
    },
    "listOfValues": {
        "listOfValuesReference": {
            "uri": "listOfValuesResourceUri"
        }
    }
    "visibleColumns":["column1", "colum2", ...],
    "valueColumn":"{valueColumn}",
    "query": {
        "queryReference": {
            "uri": "{queryResourceUri}"
        }
    }
}</pre>
	<pre><inputControl>
	    {commonAttributes}
	    <mandatory>true|false</mandatory>
	    <readOnly>true|false</readOnly>
	    <visible>true|false</visible>
	    <type>{inputControlTypeByteValue}</type>
	    <usedFields>
	{field1;field2;...}</usedFields>
	    <dataTypeReference>
	        <uri>{dataTypeResourceUri}</uri>
	    </dataTypeReference>
	    <listOfValuesReference>
	        <uri>{listOfValuesResourceUri}</uri>
	    </listOfValuesReference>
	    <queryReference>
	        <uri>{queryResourceUri}</uri>
	    </queryReference>
	    <visibleColumns>
	        <column>{column1}</column>
	        <column>{column2}</column>
	        <column>...</column>
	    </visibleColumns>
	    <valueColumn>{valueColumn}</valueColumn>
	</inputControl></pre>

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

Type	Type of Input Control	Other Fields Required (usedFields)
1	Boolean	None
2	Single value	dataType
3	Single-select list of values	listOfValues
4	Single-select query	query; queryValueColumn
5	*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

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

5.13 File

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

application/repository.file+json

```
{
    {commonAttributes},
    "type":"pdf|html|xls|rtf|csv|odt|txt
            |docx|ods|xlsx|img|font|jrxml
            |jar|prop|jrtx|xml|css
            |olapMondrianSchema
            |accessGrantSchema
            |unspecified}",
    // content is write-only;
    // it is not included in a response
    "content":"{base64EncodedContent}"
}
```

application/repository.file+xml

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

5.14 Report Unit (JRXML Report)

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

<reportUnitURI>_files/<localResourceID>

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

/reports/samples/Cascading_multi_select_report_files/Cascading_multi_select_report

For more information, see **4.6, "Nested Resources," on page 28**.

application/repository.reportUnit+json

```
{
    {commonAttributes},
    "alwaysPromptControls":
        "{true|false}",
    // default is "popupScreen"
    "controlsLayout":
        "{popupScreen|separatePage
          |topOfPage|inPage}",
    "inputControlRenderingView":
        "{inputControlRenderingView}",
    "reportRenderingView":
        "{reportRenderingView}",
    "dataSource":{
        "dataSourceReference": {
            "uri":"{dataSourceUri}"
        }
    },
    // "query" is nullable
    "query:" {
        "queryReference": {
            uri: "{queryResourceUri}"
        }
    },
    "jrxml": {
        "jrxmlFileReference": {
            "uri": "{jrxmlFileResourceUri}"
        } or
        "jrxmlFile": {
            "label": "Main jrxml",
            "type": "jrxml"
        }
    }
    "inputControls": [
        {
            "inputControlReference": {
                "uri": "{inputControlUri}"
            }
        },
        ...
    ],
    "resources": [
        "resource": {
            "name": "{resourceName}",
            "file": {contents} <*>
            }
        },
        ...
    ]
}

<*> or "fileReference": {
          "uri": "{fileResourceUri}"
       }
```

application/repository.reportUnit+xml

```
<reportUnit>
    {commonAttributes}
    <alwaysPromptControls>true|false
        </alwaysPromptControls>
    <!-- default is "popupScreen" -->
    <controlsLayout>
        popupScreen|separatePage
        |topOfPage|inPage
    </controlsLayout>
    <inputControlRenderingView>
        {inputControlRenderingView}
    </inputControlRenderingView>
    <reportRenderingView>
        {reportRenderingView}
    </reportRenderingView>
    <dataSource>
        <dataSourceReference>
            <uri>{dataSourceUri}</uri>
        </dataSourceReference>
    </dataSource>
    <query>
        <queryReference>
            <uri>{queryResourceUri}</uri>
        </queryReference>
    </query>
    <jrxml>
        <jrxmlFileReference>
            <uri>{jrxmlFileResourceUri}</uri>
        </jrxmlFileReference>
    </jrxml>
    <inputControls>
        <inputControlReference>
            <uri>{inputControlUri}</uri>
        </inputControlReference>
        ...
    </inputControls>
    <resources>
        <resource>
            <name>{resourceName}</name>
            <file>contents</file> {*}
        </resource>
        ...
    </resources>
</reportUnit>


{*} or <fileReference>
          <uri>{fileResourceUri}</uri>
       </fileReference>
```

5.15 Report Options

application/repository.reportOptions+json	application/repository.reportOptions+xml

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

5.16 Domain (semanticLayerDataSource)

For more information about accessing the schema of a Domain, see **Chapter 8, "Working With Domains," on page 65**.

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

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

5.17 Domain Topic

A Domain Topic is a Topic created by selecting database fields from a Domain. It is structurally equivalent to a JRXML report, and thus it has the same type attributes (see **5.14, "Report Unit (JRXML Report)," on page 40**). The only difference is that the data source field will reference a Domain (semanticLayerDataSource).

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

5.18 XML/A Connection

application/repository.xmlaConnection+json	application/repository.xmlaConnection+xml

{
{commonAttributes},
"url":"{xmlaServiceUrl}",
"xmlaDataSource":"{xmlaDataSource}",
"catalog":"{catalog}",
"username":"{username}",
"password":"{password}"
}
	<xmlaConnection>
	{commonAttributes}
	<url>{xmlaServiceUrl}</url>
	<xmlaDataSource>
	{xmlaDataSource}
	</xmlaDataSource>
	<catalog>{catalog}</catalog>
	<username>{username}</username>
	<password>{password}</password>
	</xmlaConnection>

5.19 Mondrian Connection

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

application/repository.mondrianConnection+json	application/repository.mondrianConnection+xml

{
{commonAttributes},
"dataSource":{
"dataSourceReference": {
"uri":"{dataSourceUri}"
}
},
"schema": {
"schemaReference": {
"uri": "{schemaFileResourceUri}"
}
}
}
	<mondrianConnection>
	{commonAttributes}
	<dataSourceReference>
	<uri>{dataSourceUri}</uri>
	</dataSourceReference>
	<schemaReference>
	<uri>{schemaFileResourceUri}</uri>
	</schemaReference>
	</mondrianConnection>

5.20 Secure Mondrian Connection

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

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

5.21 OLAP Unit

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

5.22 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>

5.23 Other Types

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

The types in the following table contain only the common attributes described in **5.1, "Common Attributes," on page 34**.

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

CHAPTER 6 THE resources SERVICE

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

For further information, see:

- **Chapter 4, "Working With Resources," on page 25** for general guidelines about using descriptors.
- **Chapter 5, "Resource Descriptors," on page 33** for a reference to every type of resource and its attributes.
- **Chapter 7, "Working With File Resources," on page 59** to download and upload file resources.
- **Chapter 8, "Working With Domains," on page 65** to view Domains and their nested resources.

This chapter includes the following sections:

6.1 Searching the Repository

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

Method	URL
GET	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 search should include all sub-folders recursively. When omitted, the default behavior is recursive (true).
type	String	Match only resources of the given type. Valid types are listed in **Chapter 5, "Resource Descriptors," on page 33**, for example: dataType, jdbcDataSource, reportUnit, or file. Multiple type parameters are allowed. Wrong values are ignored.
accessType	viewed|modified	Filters the results by access events: viewed (by current user) or modified (by current user). By default, no access event filter is applied.
dependsOn	/path/to/resource	Searches for all resources depending on specified resource. Only data source and reportUnit resources may be specified. If this parameter is specified, then all the other parameters except pagination are ignored.
showHiddenItems	true|false	When set to true, results include nested local resources (in _files) as if they were in the repository. For more information, see **4.8, "Local Resources," on page 30**. By default, hidden items are not shown (false).
sortBy	optional String	One of the following strings representing a field in the results to sort by: uri, label, description, type, creationDate, updateDate, accessTime, or popularity (based on access events). By default, results are sorted alphabetically by label.
limit offset forceFullPage forceTotalCount		Pagination is enabled by default, and the default limit is 100 results. If you work with large repositories, you may not receive all results in a single request, and you must handle pagination issues. These parameters are described in **6.2, "Paginating Search Results," on page 49**. Note that the pagination limit is applied to raw search results before permissions are computed. After permissions, the result set may be less than 100. In extreme cases, the search may return only a few items, but the search is still incomplete. In that case use forceFullPage as described in **6.2.2, "Full Page Pagination," on page 51**, or set the limit to 0 as described in **6.2.3, "No Pagination," on page 52**.
Options		
accept: application/json (default) accept: application/xml		

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

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

application/json

```
[
    {
        "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>
```

6.2 Paginating Search Results

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

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

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

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

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

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

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

6.2.1 Default Pagination

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

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

- The system admin (`superuser`) has access to every resource, and therefore the results are effectively unfiltered and each page is full. But the same can be true when you perform a search as jasperadmin within his organization, or even as a plain user within a folder where the user has full read permission. In these cases the default pagination is very efficient and has no partially-full pages.
- If you are performing a sparse search, for example finding all reports that a given user has permission to access within an entire and large organization, then the results may have many partially-full page, all of differing lengths. In this case, you may prefer to use **6.2.2, "Full Page Pagination," on page 51**.

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

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

Headers in Responses for Default Pagination	
Header	**Description**
Result-Count	This is the number of results that are contained in the current response. It can be less than or equal to the limit.
Start-Index	The Start-Index in the response is equal to the offset specified in the request. With a limit=100, it will be 0 on the first page, 100 on the second page, etc.
Next-Offset	This is the offset to request the next page. With forceFullPage=false, the Next-Offset is equivalent to Start-Index+limit, except on the last page. On the last page, the Next-Offset is omitted to indicate there are no further pages.
Total-Count	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.

6.2.2 Full Page Pagination

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

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

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 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.

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

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

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

6.2.3 No Pagination

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

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

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.

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

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

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

6.3 Viewing Resource Details

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

Method	**URL**	
GET	http://<host>:<port>/jasperserver[-pro]/**rest_v2/resources**/path/to/resource?<argument>	
Argument	**Type/Value**	**Description**
expanded	true|false	When true, all nested resources will be given as full descriptors. The default behavior, false, has all nested resources given as references. For more information, see **4.8, "Local Resources," on page 30**.
Options		
accept: application/json (default) accept: application/xml accept: application/repository.folder+<format> (specifically to view the folder resource)		

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

6.4 Creating a Resource

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

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

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/**rest_v2/resources**/path/to/folder?<argument>	
Argument	**Type/Value**	**Description**
create Folders	true|false	By default, this is true, and the service will create all parent folders if they don't already exist. When set to false, the folders specified in the URL must all exist, otherwise the service returns an error.
Content-Type		**Content**
application/repository.<resourceType>+json application/repository.<resourceType>+xml		A well defined descriptor of the specified type and format. See **Chapter 5, "Resource Descriptors," on page 33**
Return Value on Success		**Typical Return Values on Failure**
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just created.		400 Bad Request – Mismatch between the content-type and the fields or syntax of the actual descriptor.

With the PUT method, specify a unique new resource ID as part of the URL. For more information, see **4.1, "Resource URI," on page 25**.

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 don't already exist. When set to false, the folders specified in the URL must all exist, otherwise the service returns an error.
overwrite	true|false	When true, the resource given in the URL is overwritten even if it is a different type than the resource descriptor in the content. The default is false.

Content-Type	Content
application/repository.<resourceType>+json application/repository.<resourceType>+xml	A well defined descriptor of the specified type and format. See **Chapter 5, "Resource Descriptors," on page 33**

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

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

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/**rest_v2/resources**/path/to/folder

Content-Type	Content

multipart/form-data	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 (e.g. images)

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

6.5 Modifying a Resource

Use the PUT method 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 **4.8, "Local Resources," on page 30**.

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.<resourceType>+xml		A well defined descriptor of the specified type and format. See **Chapter 5, "Resource Descriptors," on page 33**.
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.

6.6 Copying a Resource

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

Method	URL	
POST	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 don't already exist. When set to false, the folders specified in the URL must all exist, otherwise the service returns an error.
overwrite	true|false	When true, the target resource given in the URL is overwritten even if it is a different type than the resource descriptor in the content. The default is false.
Options		
Content-Location: {resourceSourceUri} - Specifies the resource to be copied.		
Return Value on Success		**Typical Return Values on Failure**
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just copied.		404 Not Found – When the {resourceSourceUri} is not valid.

6.7 Moving a Resource

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

Method	URL	
PUT	http://<host>:<port>/jasperserver[-pro]/**rest_v2/resources**/path/to/folder ?<arguments>	
Argument	**Type/Value**	**Description**
create Folders	true|false	True by default, and the service will create all parent folders if they don't already exist. When set to false, the folders specified in the URL must all exist, otherwise the service returns an error.
overwrite	true|false	When true, the target resource given in the URL is overwritten even if it is a different type than the resource descriptor in the content. The default is false.
Options		
Content-Location: {resourceSourceUri} - Specifies the resource to be moved.		

Return Value on Success	Typical Return Values on Failure
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just moved.	404 Not Found – When the {resourceSourceUri} is not valid.

6.8 Deleting Resources

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

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/**rest_v2/resources**/path/to/resource

Return Value on Success	Typical Return Values on Failure
204 No Content – The request was successful and there is no descriptor to return.	404 Not Found – When the resource path or ID is not valid.

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

Method	URL	
DELETE	http://<host>:<port>/jasperserver[-pro]/**rest_v2/resources**?resourceUri={uri}&...	
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 was successful and there is no descriptor to return.	404 Not Found – When the resourceUri is not valid.

CHAPTER 7 WORKING WITH FILE RESOURCES

This chapter includes the following sections:

7.1 MIME Types

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

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

Table 7-1 MIME Types for File Contents

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

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

7.2 Downloading File Resources

There are two read operations on file resources:

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

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

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

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

Method	**URL**
GET	http://<host>:<port>/jasperserver[-pro]/**rest_v2/resources**/path/to/file/resource

Return Value on Success	**Typical Return Values on Failure**
200 OK – The response content-type will indicate the MIME type of the binary contents. See **Table 7-1, "MIME Types for File Contents," on page 59** for the list of MIME types that correspond to file resource types.	404 Not Found – The specified resource is not found in the repository.

7.3 Uploading File Resources

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

Method	**URL**	
POST	http://<host>:<port>/jasperserver[-pro]/**rest_v2/resources**/path/to/folder?<argument>	
Argument	**Type/Value**	**Description**
create Folders	true|false	True by default, and the service will create all parent folders if they don't already exist. When set to false, the folders specified in the URL must all exist, otherwise the service returns an error.
Content-Type		**Content**
application/repository.file+json application/repository.file+xml		A well defined file resource descriptor, as described in **5.13, "File," on page 40**. 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 **Table 7-1, "MIME Types for File Contents," on page 59** • data – contains the file contents

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

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

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/**rest_v2/resources**/path/to/folder

Options
Content-Description: <file-description> – Becomes the description field of the created file resource Content-Disposition: attachment; filename=<filename> – Becomes the name of the file resource

Content-Type	Content
{MIME type}	The MIME type from **Table 7-1, "MIME Types for File Contents," on page 59** that corresponds to the desired file type. The body of the request then contains the binary data representation of that file format.

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

7.4 Updating File Resources

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

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

Method	**URL**
PUT	http://<host>:<port>/jasperserver[-pro]/**rest_v2/resources**/path/to/resource
Content-Type	**Content**
application/repository.file+json application/repository.file+xml	A well defined file resource descriptor, as described in **5.13, "File," on page 40**. 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 **Table 7-1, "MIME Types for File Contents," on page 59** that corresponds to the desired file type. The body of the request then contains the binary data representation of that file format.
Return Value on Success	**Typical Return Values on Failure**
201 Created – The request was successful and, for confirmation, the response contains the full descriptor of the resource that was just created.	400 Bad Request – Mismatch between the content-type and the fields or syntax of the actual descriptor.

CHAPTER 8 WORKING WITH DOMAINS

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

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

This chapter includes the following sections:

8.1 The metadata Service

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

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

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

Method	URL
GET	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.

8.2 Fetching a Domain Schema

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

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

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

This descriptor contains the Domain schema as an internal resource:

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

```
        </bundle>
        <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 **7.3, "Uploading File Resources," on page 61**.

8.3 Fetching Domain Bundles and Security Files

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

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

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

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

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

GET http://<host>:<port>/jasperserver-pro/rest_v2/resources/Domains/supermartDomain_files/supermart_domain_security

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

You can then upload an updated language bundle or security file for the Domain with one of the methods described in **7.3, "Uploading File Resources," on page 61**.

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

CHAPTER 9 THE permissions SERVICE

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

This chapter includes the following sections:

9.1 Permission Constants

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

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

- No access: 0
- Administer: 1
- Read-only: 2
- Read-write: 6
- Read-delete: 18
- Read-write-delete: 30
- Execute-only: 32

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

role:/ROLE_ADMINISTRATOR (this is a root role and thus has no organization specified)

user:/organization_1/joeuser

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

There are two qualities of a permission:

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

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

9.2 Viewing Multiple Permissions

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

Method	**URL**	
GET	http://<host>:<port>/jasperserver[-pro]/**rest_v2/permissions**/path/to/resource/?<arguments>	
Argument	**Type/Value**	**Description**
effective Permissions	Boolean optional	When set to true, the effective permissions are returned. By default, this argument is false and only assigned permissions are returned.
recipientType	String optional	Either `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>
```

9.3 Viewing a Single Permission

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

Method	**URL**	
GET	http://<host>:<port>/jasperserver[-pro]/**rest_v2/permissions**/path/to/resource;recipient=<recipient>	
Argument	**Type/Value**	**Description**
recipient	string required	The recipient format specifies `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).

9.4 Setting Multiple Permissions

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

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

Content-Type	**Content**

application/collection+json	A JSON object that describes a set of permissions, for example:
	`{`
	`  "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.

9.5 Setting a Single Permission

The POST method accepts a single permission descriptor.

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

Content-Type	Content

application/json	A JSON object that describes a single permission on a single resource, for example:
	`{`
	`"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 `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.

9.6 Deleting Multiple Permissions

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

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

9.7 Deleting a Single Permission

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

Method	**URL**	
DELETE	http://<host>:<port>/jasperserver[-pro]/**rest_v2/permissions**/path/to/resource;recipient=<recipient>	
Argument	**Type/Value**	**Description**
recipient	string required	The recipient format specifies `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.

CHAPTER 10 THE export SERVICE

The rest_v2/export service works asynchronously: first you request the export with the desired options, then you monitor the state of the export, and finally you request the output file. Each step requires a different service call.

You must be authenticated as the system admin (`superuser`) for the export services.

This chapter includes the following sections:

10.1 Requesting an Export

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

Method	**URL**	
POST	http://<host>:<port>/jasperserver[-pro]/**rest_v2/export/**	
Content-Type	**Content**	
application/json	A JSON object that describes the export options.	
Return Value on Success		**Typical Return Values on Failure**
200 OK – Returns a JSON object that gives the ID of the running export operation.		401 Unauthorized – Export is available only to the system admin user (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 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 **Chapter 12, "The keys Service," on page 95**. 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*.

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.
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.
include-attributes	Includes all attributes that are associated with a item being exported, such as a user, an organization, or the root.
skip-attribute-values	When specified with include-attributes, only attribute names are exported with null values. Use this to prevent applying attributes that are specific to one server or one organization.
include-server-settings	When specified, the configuration and security settings on the server are exported. When imported into another server, these settings will take effect immediately.
include-access-events	When this option is present, access events (date, time, and user name of last modification) are exported along with each exported folder and resource. This option should only be used if URIs are specified.
include-audit-events	Include audit data for all resources and users in the export. The audit feature must be enabled in the server configuration.
include-monitoring-events	Include monitoring events. The monitoring feature must be enabled in the server configuration.

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

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

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

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

10.2 Polling the Export Status

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

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

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

```
{
  "phase": "inprogress",
  "message": "Progress..."
}
```

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

```
{
  "phase": "failure",
  "message": "Not enough space on
              disk"
}
```

10.3 Fetching the Export Output

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

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

10.4 Canceling an Export Operation

To cancel 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.

CHAPTER 11 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. The operation is also asychronous, 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:

11.1 Launching an Import Operation

Typically, an application will use 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. You must be authenticated as the system admin (`superuser`) to import into root, 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, and 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 **11.6, "Importing from a Web Form," on page 91**.

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 won't be imported, but the import operation will continue. • include – Attempts to import the resource by resolving dependencies with local resources. If unsuccessful, this resource is skipped. • fail – The import operation will stop and show an error.
organization?	orgID	Destination organization for importing. The file being imported must have been exported from an organization, not the root of the server. If this argument is not specified, the organization of the user performing the operation is used.
merge Organization?	true	When importing from one organization into a different organization, specify this argument. The resulting organization takes its ID from the import file. If organization IDs of import and destination do not match, and this argument is not specified, the operation stops with an error.
skipThemes?	true	When this argument is specified, any themes in the import other than the default theme is ignored. Use this argument when importing catalogs from other JasperReports Server versions that used themes incompatible with your version.
includeAccess Events?	true	Restores the date, time, and user name of last modification if they are included in the catalog to import.
includeAudit Events?	true	Imports audit events if they are included in the catalog.
includeMonitoring Events?	true	Imports 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 which 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 **Chapter 12, "The keys Service," on page 95**. For more information about import and export keys, see the *JasperReports Server Security Guide*.
secret-key	hex 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.
secretUri	repo URI	Specify the encryption key in a secure file resource, identified by its URI in the repository. This must be the same key used when exporting the catalog from the source server.
Content-Type		**Content**
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.

11.2 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
inprogress	Import has begun and is still running.
finished	The import has 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 will happen if the import operation stopped with the following error codes: • import.organizations.not.match • import.broken.dependencies

11.3 Import Errors

In case of warnings or errors, the GET method will return 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 type than the one being imported. The resource will not be 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 will be 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.

11.4 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"]
}
```

11.5 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.

11.6 Importing from a Web Form

Alternatively, you can call the `import` service directly from a web page with `form` and `input` tags. Use checkbox inputs 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 asychronous 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 and unable to restart it 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. Submitting an option name as form data sets it to true for this operation, otherwise all options are false by default. The following table describes the options you can submit in the request:

Import Web Form Options	Description
update	When the catalog contains resources with the same path and type as existing resources in the repository, those in the repository will be overwritten. Roles and users in an organization will also be 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 specified, and the catalog is sourced from a different organization than the destination, the import will fail.
skip-themes	In commercial releases, specify this option to ignore any themes in the catalog. Otherwise, any themes in the catalog will be imported into the repository.
include-access-events	When specified, the timestamps for resource creation and modification of each resource are imported into the repository.

Import Web Form Options	Description
include-audit-events	When this option is specified, any audit event logs in the catalog will be imported into the server's audit event logs.
include-monitoring-events	When this option is specified, any monitoring event logs in the catalog will be imported into the server's monitoring event logs.
include-server-settings	When this option is specified, any global server settings in the catalog will be 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>
```

CHAPTER 12 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"}`
	`]`

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 **Chapter 10, "The export Service," on page 79** and **Chapter 11, "The import Service," on page 85**.

CHAPTER 13 THE reports SERVICE

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

This chapter includes the following sections:

13.1 Running a Report

The reports service allows clients to receive report output in a single request-response. The reports service is a synchronous request, meaning the caller will be blocked until the report is generated and returned in the response. For large datasets or long reports, the delay can be significant. If you want to use a non-blocking (asynchronous) request, see **Chapter 14, "The reportExecutions Service," on page 101**

The output format is specified in the URL as a file extension to the report URI.

Method	**URL**	
GET	http://<host>:<port>/jasperserver[-pro]/**rest_v2/reports**/path/to/report.<format>?<arguments>	
Argument	**Type/Value**	**Description**
<format>	output type	One of the following formats: pdf, html, xls, xlsx, rtf, csv, xml, docx, odt, ods, jrprint. As of JasperReports Server 6.0, it is also possible to specify json if your reports are designed for data export. For more information, see the JasperReports Library samples documentation.
page?	Integer > 0	An integer value used to export a specific page.

ignore pagination?	Boolean	When set to true, the report will be generated as a single page. This can be useful for some formats such as csv. When omitted, this argument's default value is false and the report is paginated normally.
<inputControl>	String	Any input control that is defined for the report. Input controls that are multi-select may appear more than once. See examples below.
interactive?	Boolean	In a commercial editions of the server where HighCharts are used in the report, this property determines whether the JavaScript necessary for interaction is generated when exporting to HTML. By default it is true. If set to false, the chart is generated as a non-interactive image file.
onePage PerSheet?	Boolean	Valid only for the XLS format. When true, each page of the report is on a separate spreadsheet. When false or omitted, the entire report is on a single spreadsheet. If your reports are very long, set this argument to true, otherwise the report will not fit on a single spreadsheet and cause an error.
baseUrl	String	Specifies the base URL that the report will use to load static resources such as JavaScript files. You can also set the deploy.base.url property in the .../WEB-INF/js.config.properties file to set this value permanently. If both are set, the baseUrl parameter in this request takes precedence.
attachmentsPrefix	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 follow examples show various combinations of formats, arguments, and input controls:

http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/AllAccounts.html (all pages)

http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/AllAccounts.html?page=43

http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/AllAccounts.pdf (all pages)

http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/AllAccounts.pdf?page=1

http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/EmployeeAccounts.html?EmployeeID=sarah_id

http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/Cascading_multi_select_report.html?Country_multi_select=USA&Cascading_state_multi_select=WA&Cascading_state_multi_select=CA

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

13.2 Finding Running Reports

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

This syntax of the reports service is deprecated. See **Chapter 14, "The reportExecutions Service," on page 101**.

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

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

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

13.3 Stopping a Running Report

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

This syntax of the reports service is deprecated. See **Chapter 14, "The reportExecutions Service," on page 101**.

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

Content-Type	**Content**
application/xml	Either an empty instance of the ReportExecutionCancellation class or <status>cancelled</status>.

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

CHAPTER 14 THE reportExecutions SERVICE

As described in **Chapter 13, "The reports Service ," on page 97**, synchronous report execution blocks the client waiting for the response. When managing large reports that may take minutes to complete, or when running a large number of reports simultaneously, synchronous report execution slows down the client or uses many threads, each waiting for a report.

The rest_v2/reportExecutions service provides asynchronous report execution, so that the client does not need to wait for report output. Instead, the client obtains a request ID and periodically checks the status of the report to know when it is ready (also called polling). When the report is finished, the client can download the output. 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 completed.

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

This chapter includes the following sections:

14.1 Running a Report Asynchronously

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

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

Content-Type	Content
application/xml application/json	A complete `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>
    <freshData>false</freshData>
    <saveDataSnapshot>false</saveDataSnapshot>
    <outputFormat>html</outputFormat>
    <interactive>true</interactive>
    <ignorePagination>false</ignorePagination>
    <pages>1-5</pages>
    <parameters>
        <reportParameter name="someParameterName">
            <value>value 1</value>
            <value>value 2</value>
        </reportParameter>
        <reportParameter name="someAnotherParameterName">
            <value>another value</value>
        </reportParameter>
    </parameters>
</reportExecutionRequest>
```

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

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

Property	Required or Default	Description
outputFormat	Required	Specifies the desired output format: pdf, html, xls, xlsx, rtf, csv, xml, docx, odt, ods, jrprint. As of JasperReports Server 6.0, it is also possible to specify json if your reports are designed for data export. For more information, see the JasperReports Library samples documentation.
freshData	false	When data snapshots are enabled, specifies whether the report should get fresh data by querying the data source or if false, use a previously saved data snapshot (if any). By default, if a saved data snapshot exists for the report it will be used when running the report.
saveDataSnapshot	false	When data snapshots are enabled, specifies whether the data snapshot for the report should be written or overwritten with the new data from this execution of the report.
interactive	true	In a commercial editions of the server where HighCharts are used in the report, this property determines whether the JavaScript necessary for interaction is generated and returned as an attachment when exporting to HTML. If false, the chart is generated as a non-interactive image file (also as an attachment).
allowInlineScripts	true	Affects HTML export only. If true, then inline scripts are allowed, otherwise no inline script is included in the HTML output.
ignorePagination	Optional	When set to true, the report is generated as a single long page. This can be used with HTML output to avoid pagination. When omitted, the ignorePagination property on the JRXML, if any, is used.
pages	Optional	Specify a page range to generate a partial report. The format is: <startPageNumber>-<endPageNumber>
async	false	Determines whether reportExecution is synchronous or asynchronous. When set to true, the response is sent immediately and the client must poll the report status and later download the result when ready. By default, this property is false and the operation will wait until the report execution is complete, forcing the client to wait as well, but allowing the client to download the report immediately after the response.
transformerKey	Optional	Advanced property used when requesting a report as a JasperPrint object. This property can specify a JasperReports Library generic print element transformers of class net.sf.jasperreports.engine.export.GenericElementTransformer. These transformers are pluggable as JasperReports Library extensions.

Property	Required or Default	Description
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 will use to load static resources such as JavaScript files. You can also set the deploy.base.url property in the .../WEB-INF/js.config.properties file to set this value permanently. If both are set, the baseUrl parameter in this request takes precedence.
parameters	See example	A list of input control parameters and their values.

When successful, the reply from the server contains the `reportExecution` descriptor. This descriptor contains the request ID and status needed in order for the client to request the output. There are two statuses, one for the report execution itself, and one for the chosen output format.

The following descriptor shows that the report is still executing (<status>execution</status>).

```
<reportExecution>
    <currentPage>1</currentPage>
    <exports>
        <export>
            <id>html</id>
            <status>queued</status>
        </export>
    </exports>
    <reportURI>/supermart/details/CustomerDetailReport</reportURI>
    <requestId>f3a9805a-4089-4b53-b9e9-b54752f91586</requestId>
    <status>execution</status>
</reportExecution>
```

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

14.2 Polling Report Execution

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

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

Options	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.

14.3 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.

14.4 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": "b487a05a-4989-8b53-b2b9-b54752f998c4",
    "reportURI": "/reports/samples/AllAccounts",
    "exports": [{
        "id": "195a65cb-1762-450a-be2b-1196a02bb625",
        "options": {
            "outputFormat": "html",
            "attachmentsPrefix": "./images/",
            "allowInlineScripts": false
        },
```

```
        "status": "ready",
        "outputResource": {
            "contentType": "text/html"
        },
        "attachments": [{
            "contentType": "image/png",
            "fileName": "img_0_46_0"
        },
        {
            "contentType": "image/png",
            "fileName": "img_0_0_0"
        },
        {
            "contentType": "image/jpeg",
            "fileName": "img_0_46_1"
        }]
    },
    {
        "id": "4bac4889-0e63-4f09-bbe8-9593674f0700",
        "options": {
            "outputFormat": "html",
            "attachmentsPrefix": "{contextPath}/rest_v2/reportExecutions/{reportExecutionId}/exports/
{exportExecutionId}/attachments/",
            "baseUrl": "http://localhost:8080/jasperserver-pro",
            "allowInlineScripts": true
        },
        "status": "ready",
        "outputResource": {
            "contentType": "text/html"
        },
        "attachments": [{
            "contentType": "image/png",
            "fileName": "img_0_0_0"
        }]
    }]
}
```


14.5 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, in 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. You must download all attachments to display the HTML content properly. The given URL is the default path, but it can be modified with the `attachmentsPrefix` property in the `reportExecutionRequest`, as described in **14.1, "Running a Report Asynchronously," on page 101**.

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

14.6 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 with links to the bookmarks and parts that are defined by the report. 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"
      }
    ]
  }
}
```

14.7 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:

```
<exportExecution>
    <id>html;attachmentsPrefix=./images/</id>
```

```
    <status>ready</status>
    <outputResource>
        <contentType>text/html</contentType>
    </outputResource>
</exportExecution>
```

14.8 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 **Chapter 15, "The inputControls Service," on page 117**.

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 valeu of false, as explained in the table of properties in **14.1, "Running a Report Asynchronously," on page 101**.
Media-Type	**Content**	

application/json	<pre>[
	    {
	        "name":"someParameterName",
	        "value":["value 1", "value 2"]
	    },
	    {
	        "name":"someAnotherParameterName",
	        "value":["another value"]
	    }
	]</pre>	

application/xml	<pre><reportParameters>
	    <reportParameter name="Country_multi_select">
	        <value>Mexico</value>
	    </reportParameter>
	     <reportParameter name="Cascading_state_multi_select">
	        <value>Guerrero</value>
	        <value>Sinaloa</value>
	    </reportParameter>
	</reportParameters></pre>	

Return Value on Success		**Typical Return Values on Failure**
204 No Content – There is no content to return.		404 Not Found – When the request ID specified in the request does not exist.

14.9 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" your client can download the new export output and any attachments as described in **14.5, "Requesting Report Output," on page 107**. 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

14.10 Finding Running Reports and Jobs

The reportExecutions service provides a method to search for reports that are running on the server, which includes asychronous reports that are still running and those that are finished but still in the 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 the currently logged-in user's organization.
jobID	Optional String	For scheduler jobs, this argument matches the ID of the job that triggered the running report.
jobLabel	Optional String	For scheduler jobs, this argument matches the name of the job that triggered the running report.
userName	Optional String	For scheduler jobs, this argument matches the user ID that created the job.
fireTimeFrom	Optional Date/Time	For scheduler jobs, the fire time arguments define a range of time that matches if the job that is currently running was triggered during this time. You can specify either or both of the arguments. Specify the date and time in the following pattern: yyyy-MM-dd'T'HH:mmZ.
fireTimeTo		
Options		
accept: application/xml (default) accept: application/json		
Return Value on Success	**Typical Return Values on Failure**	
200 OK – The content is a descriptor for each of the matching results.	204 No Content – When the search results are empty.	

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

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

Given the request ID, you can obtain more information about each result by downloading the full `reportExecution` descriptor, as described in **14.4, "Requesting Report Execution Details," on page 106**.

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 of the same organization or one of its child organizations.
- A regular user can see every running report, but can cancel only the reports that he initiated.

14.11 Stopping Running Reports and Jobs

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

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

Content-Type	**Content**
application/xml application/json	Send a `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.

14.12 Removing a Report Execution

Deleting a report that has been executed 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.

CHAPTER 15 THE inputControls SERVICE

The reportExecutions service includes only a simple mechanism for setting input controls (parameters) in reports. The inputControls service provides a complete set of operations for reading and setting input controls. 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`

This chapter includes the following sections:

15.1 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 drop-down 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 at a later time.

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 **15.3, "Listing Input Control Values," on page 122**.
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 shows 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. Indpendently 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
            }
        }
    ]
}
```


15.2 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 drop-down list. The type is one of the following values:

bool (checkbox)	singleValue
singleSelect (drop-down)	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 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 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 upon. 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 upon this input control. If this input control is modifided (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.

15.3 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 controls, 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 contols 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 upon 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 shows 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 stucture 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.

15.4 Changing the Order of Input Controls

The inputControls service does not allow you 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 will invalidate your current order in memory. You will 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 strucutres, 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.

15.5 Setting Input Control Values

After your app has fetched all structures and values and created a UI, your users 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 several 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 will respond 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.

<table>
<tr><th>Method</th><th colspan="2">URL</th></tr>
<tr><td>POST</td><td colspan="2">http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/<resourceURI>
/inputControls?<argument></td></tr>
<tr><th>Argument</th><th>Type/Value</th><th>Description</th></tr>
<tr><td>freshData</td><td>true|false</td><td>When <code>freshData=true</code> is specified, the list of values for any selection input contols 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.</td></tr>
<tr><th colspan="2">Content-Type</th><th>Content</th></tr>
<tr><td colspan="2">application/xml</td><td>An XML object that lists the new value for just those input controls that are modified, for example:
<pre><reportParameters>
    <reportParameter name="Country_multi_select">
        <value>Mexico</value>
    </reportParameter>
     <reportParameter name="Cascading_state_multi_select">
        <value>Guerrero</value>
        <value>Sinaloa</value>
    </reportParameter>
</reportParameters></pre></td></tr>
</table>

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 you request 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 contols 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 – 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 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"
                }
            ]
        },
        ...
    ]
}
```

CHAPTER 16 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 **Chapter 15, "The inputControls Service," on page 117**.

This chapter includes the following sections:

16.1 Listing Report Options

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

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

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

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

16.2 Creating Report Options

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

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

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

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

With the following request body:

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

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

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

16.3 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 **Chapter 15, "The inputControls Service," on page 117**.

Method	**URL**
PUT	http://<host>:<port>/jasperserver[-pro]/**rest_v2/reports**/path/to/report**/options**/<optionID>/

Content-Type	**Content**
application/json	A JSON object that lists the input control selections. See example below.

Return Value on Success	**Typical Return Values on Failure**
200 OK	404 Not Found – When the specified report URI is not found in the repository.

For example, we change the report option we created in **16.2, "Creating Report Options," on page 132** 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"]
}
```

16.4 Deleting Report Options

Use the following method to delete a given report option.

Method	**URL**
DELETE	http://<host>:<port>/jasperserver[-pro]/**rest_v2/reports**/path/to/report**/options**/<optionID>/

Return Value on Success	Typical Return Values on Failure
200 OK	404 Not Found – When the specified report URI is not found in the repository.

CHAPTER 17 THE jobs SERVICE

The rest_v2/jobs service provides the interface to schedule reports and manage scheduled reports (also called jobs). This service provides an API to scheduler features such as bulk updates, pausing jobs, FTP output and exclusion calendars.

This chapter includes the following sections:

17.1 Listing Report Jobs

Use the following method to list jobs, either all scheduled jobs on the server or the jobs for a specific report:

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/**rest_v2/jobs**?<argument>

Argument	Type/Value	Description
reportUnitURI?	/path/to/report	Optional URI (repository path) of a report or report option 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 report or report option.
Options		
accept: application/xml accept: application/json		
Return Value on Success		**Typical Return Values on Failure**
200 OK – The body contains and array or list of `jobsummary` descriptors.		404 Not Found – When no job is found.

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` XML element returned by the rest_v2/jobs service has a different structure than the element with the same name returned by the deprecated rest/jobsummary (v1) service.

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

XML:  <jobs>
          <jobsummary>
              <id>1898</id>
              <label>SampleJobName</label>
              <description>Accounts Sample Job</description>
              <reportUnitURI>/reports/AllAccounts</reportUnitURI>
```

```
            <reportLabel>Accounts Report</reportLabel>
            <state>
                <nextFireTime>2013-10-12T00:00:00+03:00</nextFireTime>
                <value>NORMAL</value>
            </state>
            <owner>jasperadmin|organization_1</owner>
            <version>0</version>
        </jobsummary>
        ...
    </jobs>
```

17.2 Extended Job Search

The GET method is also used for more advanced job searches. You can search by owner or for strings in the label, or you can match arbitrary fields of the job descriptor with a special syntax. You can also control the pagination and sorting order of the reply.

Method	**URL**	
GET	http://<host>:<port>/jasperserver[-pro]/**rest_v2/jobs**?<arguments>	
Argument	**Type/Value**	**Description**
label	string	Performs a case-insensitive string search for this value anywhere within the label field of every job.
owner	string	The name of the user who scheduled the report, if necessary in the format <username>%7C<organization> (%7C is the | character).
reportUnitURI?	/path/to/report	Specify the URI of a report or report option to list all of its jobs. You may need to encode the / characters in the URI with `%2F`.
example?	JSON `jobModel`	Searches for jobs that match the JSON `jobModel`, which is a fragment of a job descriptor containing one or more fields to be matched. Because the JSON fragment appears as an argument to the URL, it must be properly URL-encoded (percent-encoded). See the example below.
limit	integer	Turns on pagination of the result by specifying the number of `jobsummary` descriptors per results page. The default is -1 for no limit and thus no pagination (all results are returned together).
offset	integer	Determines the page number in paginated results by specifying the index of the first `jobsummary` to be returned.
sortType		Possible values are: NONE, SORTBY_JOBID, SORTBY_JOBNAME, SORTBY_REPORTURI, SORTBY_REPORTNAME, SORTBY_REPORTFOLDER, SORTBY_OWNER, SORTBY_STATUS, SORTBY_LASTRUN, SORTBY_NEXTRUN

isAscending	true / false	Determines the sort order: ascending if true, descending if false or omitted.
Options		
accept: application/xml accept: application/json		
Return Value on Success		**Typical Return Values on Failure**
200 OK – The body contains and array or list of `jobsummary` descriptors that match the search criteria.		404 Not Found – When no matching job is found in the server.

The body of the response contains `jobsummary` elements, in the same format as shown in **17.1, "Listing Report Jobs," on page 135**.

The `example` parameter lets you specify a search on fields in the job descriptor, such as output formats. You can specify any field in the job descriptor or in any of its nested structures. Some fields may be specified in both the `example` parameter and in a dedicated parameter, for example label. In that case, the search specified in the `example` parameter takes precedence.

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

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

And the corresponding URI, with proper encoding, is:

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

17.3 Viewing a Job Definition

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

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

The GET method returns a descriptor that fully describes all the aspects of a scheduled job, such as recurrence, parameters, output, email notifications, and alerts, if any. All fields are included, many of which may be null if

not set for the chosen job. For additional options around file output, see **17.13, "Specifying FTP Output," on page 155** and **17.14, "Specifying File System Output," on page 157**.

JSON:


```
{
    "id": 1906,
    "version": 0,
    "username": "jasperadmin|organization_1",
    "label": "Daily ",
    "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</usingDefaultReportOutputFolderURI>
        <version>-1</version>
    </repositoryDestination>
</clientJob>
```

17.4 Defining the Job Trigger

Within the descriptor of 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 report associated with the job. There are two kinds of triggers:

- `simpleTrigger` - Simple recurrence that fires a given number of times with a given interval in between.
- `calendarTrigger` - Calendar recurrence that fires at specific times on given days of the week or days of the month.

The following sections give the fields that define each trigger and when they fire.

17.4.1 Simple Trigger

The `simpleTrigger` has the following fields to define its recurrence pattern:

Field and JSON Example	Description
`timezone` `"timezone": "America/Los_Angeles"`	The timezone for all dates and times used by the trigger.
`startType` `"startType":2`	Determines when the job becomes active. For the simple trigger, this also determines the base time at which recurrence starts. Supported values: 1 - The job starts immediately, and the trigger fires right away. 2 - The job starts at the `startDate`, at which time it will fire.
`startDate` `"startDate": "2020-02-29 01:00"`	The date and time at which the job will start. The simple trigger will fire at this time, and it will begin its recurrence at this time. The format is "yyyy-MM-dd HH:mm" and the timezone field is applied.
`endDate` `"endDate": "2020-05-31 23:00"`	The date and time at which the job will stop. The trigger will not fire after this time, even if any occurrences still remain, unless a misfire occurs and the misfire policy allows it. The format is "yyyy-MM-dd HH:mm" and the timezone field is applied. Note that using T as a separator in the endDate is not supported.
`calendarName` `"calendarName":"holidayCalendar"`	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 **Chapter 18, "The calendars Service," on page 159**.

Field and JSON Example	Description
`misfireInstruction` `"misfireInstruction": 0`	An integer value that defines the behavior if the trigger did not fire when scheduled. 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 did not fire on time. In this case, the scheduler takes action based on the value in this field. The misfire instruction does not apply if the trigger fires normally but the report encounters an error. The values are described in the next table.
`occurrenceCount` `"occurrenceCount": 1`	An integer that defines how many times the trigger will fire, provided the recurrence intervals happen before the `endDate`.
`recurrenceInterval` `"recurrenceInterval": 2`	The time interval between scheduled firings of the trigger. The interval unit is provided in the next field.
`recurrenceIntervalUnit` `"recurrenceIntervalUnit": "DAY"`	The unit of time for the recurrence interval. Supported values: MINUTE, HOUR, DAY, WEEK. For units greater than MINUTE, the `startType` and `startDate` is the basis for recurrence. For example, if the trigger runs immediately and recurs every 2 days, it will run at the current time on the subsequent days.

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	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 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 upon a misfire situation, the custom `updateAfterMisfire` method will be called on the trigger to determine the misfire action. In this case, you must define and enable a custom trigger class on the server, which is beyond the scope of this documentation.

misfireInstruction	Description for Simple Triggers
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 upon a misfire situation, the SimpleTrigger will be re-scheduled 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 upon a misfire situation, the SimpleTrigger will be re-scheduled 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 firings 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 firings 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 will happen later, past the last expected occurrence.

17.4.2 Calendar Trigger

A calendar trigger lets you schedule a job to run multiple times based on any combination of time and date. The various fields 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 10am to noon every Monday.

Field and JSON Example	Description
`timezone` `"timezone": "America/Los_Angeles"`	The timezone for all dates and times used by the trigger.
`startType` `"startType":2`	Determines when the job becomes active. Supported values: 1 - The job starts immediately, and the trigger may fire right away. 2 - The job is scheduled to start at the specified start date.

Field and JSON Example	Description
startDate `"startDate": "2020-02-29 01:00"`	The date and time at which the job will be active. 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 field is applied.
endDate `"endDate": "2020-05-31 23:00"`	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. The format is "yyyy-MM-dd HH:mm" and the timezone field is applied. Note that using T as a separator in the endDate is not supported.
calendarName `"calendarName":"holidayCalendar"`	The name of a previously defined exclusion calendar. An exclusion calendar defines a set of dates or times when 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 **Chapter 18, "The calendars Service," on page 159**.
misfireInstruction `"misfireInstruction": 0`	An integer value that defines the behavior if the trigger did not fire when scheduled. 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 did not fire on time. In this case, the scheduler takes action based on the value in this field. The misfire instruction does not apply if the trigger fires normally but the report encounters an error. The values are described in the next table.
minutes `"minutes":"0-10"`	Specifies the minute or minutes at which the trigger fires. 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 the trigger fires every 10 minutes starting from HH:05. • * means the trigger fires every minute of the hour.

Field and JSON Example	Description
hours `"hours":"8-16"`	Specifies the hour or hours at which the trigger fires. The minutes field determines when during the hour that the trigger will fire, possibly multiple times. All hours are specified in 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 during the hours 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
`"months": {`
`"month": ["3", "6", "9", "12"]`
`}`
	A list of months during which the trigger fires. The month values are 1 for January and 12 for Dececmber.

daysType `"daysType":"WEEK"`	Determines whether trigger days are defined by week or by month, or both. Supported values: ALL, WEEK, MONTH

weekDays
`"weekDays": {`
`"day": ["1", "4", "6"]`
`}`
	Specifies a list of days of the week on which the trigger fires. The hours and minutes fields determine when during the day the trigger will fire, possibly multiple times. The day values are 1 for Sunday and 7 for Saturday.

monthDays `"monthDays": "1,3,5-22"`	Specifies the days of the month on which the trigger fires. The hours and minutes fields determine when during the day the trigger will fire, possibly multiple times. 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 each day starting from the 2nd to the 5th of the month. Day values and ranges can be concatenated using commas as separators. • A day value with an increment, for example 1/5 means the trigger fires every 5 days starting on 1st of the month. • * means the trigger fires during every day.

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": "2020-04-24 00:00",
        "endDate": "2020-08-03 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 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 Calendar Triggers
0	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 upon a misfire situation, the custom `updateAfterMisfire` method will be called on the trigger to determine the misfire action. In this case, you must define and enable a custom trigger class on the server, which 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.

misfireInstruction	Description for Calendar Triggers
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.

17.5 Scheduling a Report

To schedule a report, create its job descriptor and use the PUT method of the jobs service. Specify the report being scheduled inside the job descriptor.

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 below.

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

Return Value on Success	Typical Return Values on Failure
201 Created – The body contains the job descriptor of the newly created job. It is similar to the one that was sent but now contains the jobID for the new job.	404 Not Found – When the report specified in the job descriptor is not found in the server.

When you schedule a report, create a job descriptor similar to the one returned by the GET method. However, you need only specify the relevant fields in the job descriptor; do not include any null fields. Do not specify any job IDs in the descriptor, because the server will assign them. For example 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 Engineering Associates"],
                "Cascading_state_multi_select": ["DF",
                "Jalisco",
                "Mexico"]
            }
        }
    },
    "baseOutputFilename": "Cascading_multi_select_report",
    "outputTimeZone": "America/Los_Angeles",
    "repositoryDestination": {
        "folderURI": "/temp"
    },
    "outputFormats": {
        "outputFormat": ["PDF", "XLS"]
    }
}
```

If needed, you can configure the server to accept the other parameters and keep them with the newly create job, but the default is to only store the required fields. For more information, see **17.15, "Storing Additional Job Parameters," on page 157**.

The response of the PUT request is the descriptor of the newly created job, similar to the result of the GET request shown in **17.3, "Viewing a Job Definition," on page 138**. It includes all the fields of the job descriptor, including the server-assigned ID and all the null fields.

17.6 Viewing Job Status

The following method returns the current run time state of a job:

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

17.7 Editing a Job Definition

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

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/**rest_v2/jobs**/<jobID>/	
Content-Type		**Content**
application/job+xml application/job+json		A well-formed XML or JSON job descriptor. The request can include either a complete descriptor such as the result of a GET request described in **17.3, "Viewing a Job Definition," on page 138**, or it can be a minimally sufficient descriptor as shown in **17.5, "Scheduling a Report," on page 148**.
Options		
accept: application/job+xml accept: application/job+json		
Return Value on Success		**Typical Return Values on Failure**
200 OK – The response include the complete job descriptor, for an example, see **17.3, "Viewing a Job Definition," on page 138**.		404 Not Found – When the specified <jobID> does not exist.

17.8 Updating Jobs in Bulk

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

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/**rest_v2/jobs**?<arguments>	
Argument	**Type/Value**	**Description**
id?	jobID string	Can be used multiple times to create a list of job IDs to update.
replace Trigger IgnoreType	true / false	Specify true when you send a new trigger type. By default, this is false, and the trigger can be updated but not changed to a different type. See below.
Content-Type		**Content**
application/json application/xml		A well-formed `jobModel` descriptor, which is a fragment of a job descriptor containing only the fields to be updated. See example below.
Options		
accept: application/json accept: application/xml		

Return Value on Success	Typical Return Values on Failure
200 OK – The array or list of jobs that were restarted.	404 Not Found – When the specified <jobID> does not exist.

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

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

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

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

- You can describe nested structures by using the *nestedName*`Model` 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 `is`*ElementName*`Modified` element and giving it the value true. This indicates that the element's new value is null, and thus that the element should be removed altogether from the job descriptor.

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

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

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

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

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

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

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

17.9 Pausing Jobs

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

Method	**URL**
POST	http://<host>:<port>/jasperserver[-pro]/**rest_v2/jobs/pause**/
Content-Type	**Content**
application/xml application/json	An array or list of job IDs to pause. See example below. If the body of the request is empty, or the list is empty, all jobs in the scheduler will be paused.
Options	
accept: application/json accept: application/xml	
Return Value on Success	**Typical Return Values on Failure**
200 OK – The array or list of jobs that were paused. Jobs specified with a <jobID> that doesn't exist are ignored without error.	

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

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

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

17.10 Resuming Jobs

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

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

Content-Type	Content
application/xml application/json	An array or list of job IDs to resume. See example below. If the body of the request is empty, or the list is empty, all paused jobs in the scheduler will resume.

Options
accept: application/json accept: application/xml

Return Value on Success	Typical Return Values on Failure
200 OK – The array or list of jobs that were resumed. Jobs specified with a <jobID> that doesn't exist are ignored without error.	

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

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

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

17.11 Restarting Failed Jobs

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

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

Content-Type	Content
application/xml application/json	An array or list of job IDs to restart. See example below.

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

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

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

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

17.12 Deleting Jobs

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

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

Method	URL	
DELETE	http://<host>:<port>/jasperserver[-pro]/**rest_v2/jobs**?<arguments>	
Argument	**Type/Value**	**Description**
id	Multiple String	Enter as many job IDs as you want to delete, for example: ?id=5594&id=5645&id=5761
Options		
accept: application/xml accept: application/json		

Return Value on Success	Typical Return Values on Failure
200 OK – The content is a list of deleted jobs, as shown in the example below.	

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

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

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

17.13 Specifying FTP Output

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

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

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

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

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

Field	Description	JSON Example
type	Type of FTP connection requested: ftp (default), ftps, or sftp.	"type": "sftp"
serverName	The host name of the FTP server.	"serverName": "ftp.example.com"
port	Integer value that specifies the port number of the ftp server. The default value depends on the connection type: ftp = 21, sftp = 22, and ftps = 990.	"port": 22
userName	The login user name for the FTP server.	"userName": "anonymous"
password	The login password for the given userName on the FTP server.	"password": "securePassword"
folderPath	The path of the folder where the job output resources should be created.	"folderPath": "/path/to/folder"
implicit	Specifies the security mode for FTPS, Implicit if true (default) or Explicit if false. If implicit is true, the default port is set to 990.	"implicit": false
protocol	Specifies the secure socket protocol to be used, for example SSL or TLS.	"protocol": "TLS"
prot	Specifies the PROT command for FTP. Supported values: • C - Clear • S - Safe (SSL protocol only) • E - Confidential (SSL protocol only) • P - Private	"prot": "C"
pbsz	Specifies the protection buffer size: 0 to (2^32)-1 decimal integer. The default is 0.	"pbsz": 0
sshKey	Repository URI of the SSH private key resource (used for SFTP authentication).	"sshKey": "/sshKeys/myOpenSSHKey"
sshPassphrase	The passphrase for the SSH private key (used for SFTP authentication).	"sshPassphrase": "mySecurePhrase"

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

17.14 Specifying File System Output

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

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

To create a job with file system output, add the `outputLocalFolder` element to the XML job descriptor, as shown in the following example:

```
<job>
    <reportUnitURI>/reports/samples/AllAccounts</reportUnitURI>
    <label>MyJob</label>
    <description>MyJob description</description>
    <baseOutputFilename>WeeklyAccountsReport</baseOutputFilename>
    <repositoryDestination>
        <folderURI>/reports/samples/</folderURI>
        <overwriteFiles>true</overwriteFiles>
        <sequentialFilenames>false</sequentialFilenames>
        <outputLocalFolder>/temp/scheduledReports/</outputLocalFolder>
    </repositoryDestination>
    <outputFormats>
        <outputFormat>XLS</outputFormat>
        <outputFormat>PDF</outputFormat>
    </outputFormats>
  ...
</job>
```

As with FTP output, file system output is always specified in addition to repository output, and the output will be written to both the repository and the local folder. The file names to be written are the same ones that are generated by the job output, as specified by the `baseOutputFilename`, sequential pattern if any, and format extensions such as .pdf. Similarly, the file overwrite and sequential filename behavior specified for repository output also apply to file system output.

17.15 Storing Additional Job Parameters

When sending a job descriptor as described in **17.5, "Scheduling a Report," on page 148**, the server does not store all fields 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 fields 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 parameters are stored when creating a job. When set to `true`, all valid job descriptor parameters sent when creating the job are stored in the newly created job.

CHAPTER 18 THE calendars SERVICE

The scheduler allows a job to be defined with a list of excluded days or times when you do not want the job to run. For example, if you have a report scheduled to run every business day, you want to exclude holidays that change every year. The list 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. When scheduling a report, reference the name of the calendar to exclude, and the scheduler automatically calculates the correct days to trigger the report. The scheduler also allows you to update an exclusion calendar and update all of the report jobs that used it. Therefore, you can update the calendar of excluded holidays every year and not need to modify any report jobs.

This chapter includes the following sections:

18.1 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"
}
```

18.2 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>
```

18.3 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"
}
```

18.4 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 **18.1, "Creating an Exclusion Calendar," on page 159** 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"`
	`]`
	`}`

18.5 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.

18.6 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"`
	`}`

Expected Reply:

Response	400 Bad Request

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

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

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

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

Expected Reply:

Respons e	400 Bad Request

Body	<pre>{
	"message": "mandatory parameter 'reportJobCalendar.rangeStartingCalendar' not found",
	"errorCode": "mandatory.parameter.error",
	"parameters":
	[
	"reportJobCalendar.rangeStartingCalendar"
	]
	}</pre>

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

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

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

Expected Reply:

Response	400 Bad Request

Body	{
	"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"
	}

Expected Reply:

Response	400 Bad Request

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

CHAPTER 19 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 show the format of a query in XML:

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

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

```
<queryResult>
  <names>
    <name>expense_join_account.ej_account_account_description</name>
    <name>expense_join_account.ej_expense_fact_account_id</name>
    <name>expense_join_account.ej_account_account_parent</name>
    <name>expense_join_account.ej_account_account_rollup</name>
    <name>expense_join_account.ej_account_account_type</name>
    <name>expense_join_account.ej_account_Custom_Members</name>
    <name>expense_join.ej_expense_fact_amount</name>
    <name>expense_join_store.ej_store_store_type</name>
    <name>expense_join_store.ej_store_store_street_address</name>
    <name>expense_join_store.ej_store_store_city</name>
    <name>expense_join_store.ej_store_store_state</name>
    <name>expense_join_store.ej_store_store_postal_code</name>
```

```
    <name>expense_join_store.sample_time</name>
  </names>

  <values>
    <row>
      <value xsi:type="xs:string">Marketing</value>
      <value xsi:type="xs:int">4300</value>
      <value xsi:type="xs:int">4000</value>
      <value xsi:type="xs:string">+</value>
      <value xsi:type="xs:string">Expense</value>
      <value xsi:nil="true"/>
      <value xsi:type="xs:double">1884.0000</value>
      <value xsi:type="xs:dateTime">1997-01-01T04:05:06+02:00</value>
      <value xsi:type="xs:string">HeadQuarters</value>
      <value xsi:type="xs:string">1 Alameda Way</value>
      <value xsi:type="xs:string">Alameda</value>
      <value xsi:type="xs:string">CA</value>
      <value xsi:type="xs:int">94502</value>
      <value xsi:type="xs:string">USA</value>
      <value xsi:type="xs:time">04:05:06+02:00</value>
    </row>
     ...
  </values>
</queryResult>
```

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

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

CHAPTER 20 THE caches SERVICE

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

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

Method	**URL**
DELETE	http://<host>:<port>/jasperserver-pro/**rest_v2/caches/vds**/

Return Value on Success	**Typical Return Values on Failure**
204 No Content – There is nothing to return.	404 Not Found – When the specified cache does not exist.

CHAPTER 21 THE organizations SERVICE

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

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

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

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

This chapter includes the following sections:

21.1 Searching for Organizations

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

Method	URL
GET	http://<host>:<port>/jasperserver-pro/**rest_v2/organizations**?<arguments>

Argument	Type	Description
q	Optional String	Specify a string or substring to match the organization ID, alias, or name of any organization. The search is not case sensitive. Only the matching organizations are returned in the results, regardless of their hierarchy.
include Parents	Optional Boolean	When used with a search, the result will include the parent hierarchy of each matching organization. When not specified, this argument is false by default.
rootTenantId	Optional String	Specifies an organization ID as a base for searching and listing child organizations. The base is not included in the results. Regardless of this base, the `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 superuser at the root of the organization hierarchy:

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

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

21.2 Viewing an Organization

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

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

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

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

21.3 Creating an Organization

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

Method	**URL**	
POST	http://<host>:<port>/jasperserver-pro/**rest_v2/organizations**?<argument>	
Argument	**Type**	**Description**
create Default Users	Optional Boolean	Set this argument to false to suppress the creation of default users (`joeuser`, `jasperadmin`) in the new organization. When not specified, the default behavior is true and organizations are created with the standard default users.
Content-Type		**Content**
application/xml application/json		A partial or complete organization descriptor that includes the desired properties for the organization.

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

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

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

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

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

21.4 Modifying Organization Properties

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

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

Content-Type	**Content**
application/xml application/json	A partial organization descriptor that includes the properties to change. Do not specify the following properties: `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"
}
```

21.5 Setting the Theme of an Organization

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

For example:

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

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

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

21.6 Deleting an Organization

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

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

Return Value on Success	**Typical Return Values on Failure**
204 No Content – The organization was successfully deleted.	400 Bad Request – When attempting to delete the organization of the logged-in user. 404 Not Found – When the ID of the organization cannot be resolved.

CHAPTER 22 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:

22.1 Searching for Users

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

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

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

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

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

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

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

```
    <fullName>Joe User</fullName>
    <username>joeuser</username>
  </user>
</users>
```

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

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

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

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

22.2 Viewing a User

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

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

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

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

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

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

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

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

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

22.3 Creating a User

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

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

To create a user, the user ID in the URL must be unique on the server or in the organization. If the user ID already exists, that user account will be modified, as described in **22.4, "Modifying User Properties," on page 190**.

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. The role ROLE_USER is automatically assigned to all users, so it does not need to be specified. Do not specify the following properties: `username` – Specified in the URL and cannot be modified in the descriptor. `tenantID` – Specified in the URL and cannot be modified in the descriptor. `previousPasswordChangeTime` – Computed automatically by the server.

Return Value on Success	**Typical Return Values on Failure**
201 Created – The user was successfully created using the values in the descriptor. The response contains the full descriptor of the new user.	404 Not Found – When the organization ID cannot be resolved.

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

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

The `externallyDefined` property is true when the user is synchronized from a 3rd party such as an LDAP directory or single sign-on mechanism. 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*.

22.4 Modifying User Properties

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

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

To modify a user, the user ID in the URL must already exist on the server or in the organization. If the user ID doesn't exist, a user account will be created, as described in **22.3, "Creating a User," on page 189**.

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

Content-Type	**Content**
application/xml application/json	A user descriptor that includes the properties you want to change. Do not specify the following properties: `username` – Specified in the URL and cannot be modified in the descriptor. `tenantID` – Specified in the URL and cannot be modified in the descriptor. `previousPasswordChangeTime` – Computed automatically by the server.

Return Value on Success	**Typical Return Values on Failure**
200 OK – The user properties were successfully updated.	404 Not Found – When the organization ID cannot be resolved.

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

```
{
  "enabled":true,
  "password":"newPassword",
  "roles":[
    {"name":"ROLE_USER"},
    {"name":"ROLE_STOREMANAGER", "tenantId":"organization_1"}]
}
```

22.5 Deleting a User

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

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

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

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

Return Value on Success	**Typical Return Values on Failure**
204 No Content – The user was successfully deleted.	404 Not Found – When the ID of the organization cannot be resolved.

CHAPTER 23 THE roles SERVICE

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

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

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

This chapter includes the following sections:

23.1 Searching for Roles

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

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

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

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

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

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

```
<roles>
  <role>
    <externallyDefined>false</externallyDefined>
    <name>ROLE_ADMINISTRATOR</name>
  </role>
```

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

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

23.2 Viewing a Role

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

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

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/**rest_v2/roles**/roleID http://<host>:<port>/jasperserver[-pro]/**rest_v2/organizations**/orgID/**roles**/roleID
Options	
accept: application/xml (default) accept: application/json	

Return Value on Success	Typical Return Values on Failure
200 OK – The content is the descriptor for the given role.	404 Not Found – When the role ID or organization ID does not match any role or organization. The content includes an error message.

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

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

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

23.3 Creating a Role

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

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

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

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/**rest_v2/roles**/roleID http://<host>:<port>/jasperserver[-pro]/**rest_v2/organizations**/orgID/**roles**/roleID

Content-Type	Content
application/xml application/json	An empty role descriptor, either <role></role> or {}. Do *not* specify the following properties: `name` – Specified in the URL and should not be modified in the descriptor. `tenantID` – Specified in the URL and cannot be modified in the descriptor. `externallyDefined` – Computed automatically by the server.

Return Value on Success	Typical Return Values on Failure
201 Created – The role was successfully created. The response contains the full descriptor of the new role.	404 Not Found – When the organization ID cannot be resolved.

23.4 Modifying a Role

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

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

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

Method	**URL**
PUT	http://<host>:<port>/jasperserver[-pro]/**rest_v2/roles**/roleID http://<host>:<port>/jasperserver[-pro]/**rest_v2/organizations**/orgID/**roles**/roleID

Content-Type	**Content**
application/xml application/json	A role descriptor containing a single property: `name` – The new name for the role.

Return Value on Success	**Typical Return Values on Failure**
200 OK – The role was successfully updated. The response contains the full descriptor of the updated role.	404 Not Found – When the organization ID cannot be resolved.

23.5 Setting Role Membership

To assign role membership to a user, set the roles property on the user account with the PUT method of the users service. For details, see **22.4, "Modifying User Properties," on page 190**.

23.6 Deleting a Role

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

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

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

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/**rest_v2/roles**/roleID http://<host>:<port>/jasperserver[-pro]/**rest_v2/organizations**/orgID/**roles**/roleID

Return Value on Success	Typical Return Values on Failure
204 No Content – The role was successfully deleted.	404 Not Found – When the ID of the organization cannot be resolved.

CHAPTER 24 THE attributes SERVICE

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

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

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

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

This chapter includes the following sections:

24.1 Attribute Descriptors

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

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

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

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

Attributes with the same name may be defined on different entities. For example, a user has a specific value for an attribute, the organization he belongs to has a default value for the same attribute, and the server level has yet another value for it. In this example, three separate attributes are defined, but they have the same name because they occur on different entities. The mechanisms described in **24.5, "Referencing Attributes," on page 201** can take advantage of this to implement default values.

24.2 Secure Attributes

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

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

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

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

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

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

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

24.3 Entities with Attributes

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

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

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

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

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

24.4 Permissions for Accessing Attributes

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

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

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

24.5 Referencing Attributes

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

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

The server provides two different methods to reference attributes:

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

24.6 Attribute Limitations

Attributes have the following limitations in the attributes service:

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

The attributes service detects these conditions and returns errors accordingly:

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

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

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

24.7 Viewing Attributes

The GET method of the attributes service retrieves the list of attributes, if any, defined for the specified entity (a user, an organization, or the server-level). For possible values of <entity> in the URL, see **24.3, "Entities with Attributes," on page 201**.

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"
}
```

24.8 Setting Attributes

The PUT method of the attributes service adds or replaces attributes on the specified entity (a user, an organization, or the server-level). For possible values of <entity> in the URL, see **24.3, "Entities with Attributes," on page 201**.

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 **24.6, "Attribute Limitations," on page 202**). If one attribute causes an error, the operation stops and returns an error, but 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"
}
```

24.9 Deleting Attributes

The DELETE method of the attributes service removes attributes from the specified entity (a user, an organization, or the server-level). When attributes are removed, both the name and the value of the attribute are removed, not only the value. For possible values of <entity> in the URL, see **24.3, "Entities with Attributes," on page 201**.

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 **24.6, "Attribute Limitations," on page 202**). If one attribute causes an error, the operation stops and returns an error, but attributes that were already deleted remain deleted.

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

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/**rest_v2/**<entity>**attributes**/attrName

Return Value on Success	Typical Return Values on Failure
204 No Content – The attribute was successfully removed from the given entity.	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 **24.6, "Attribute Limitations," on page 202**).