# Use REST/HTTP

This guide shows you how to configure and use DS REST APIs to access directory services over HTTP. The RESTful HTTP APIs return directory data as JSON resources. DS APIs are based on the ForgeRock® Common REST API framework.

The ForgeRock® Common REST API works across the platform to provide common ways to access web resources and collections of resources.

**Create**

Create resources over HTTP.

**Read**

Read resources over HTTP.

**Update**

Update resources over HTTP.

**Delete**

Delete resources over HTTP.

**Search**

Query DS over HTTP.

**Passwords/Accounts**

Manage passwords and accounts.

ForgeRock® Identity Platform serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of

their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

# DS REST APIs

DS REST APIs offer HTTP access to directory data as JSON resources. DS software maps JSON resources onto LDAP entries.

NOTE

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see Learn About the Evaluation Setup Profile.

The examples demonstrate how to contact the server over HTTPS using the deployment CA certificate. Before trying the examples, generate the CA certificate in PEM format from the server deployment key and password:

```
$ dskeymgr \
 export-ca-cert \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --outputFile ca-cert.pem
```

## About ForgeRock Common REST

ForgeRock® Common REST is a common REST API framework. It works across the ForgeRock platform to provide common ways to access web resources and collections of resources. Adapt the examples in this section to your resources and deployment.

NOTE

This page describes the full Common REST framework. Some platform component products do not implement all Common REST behaviors exactly as described. For details, refer to the product-specific examples and reference information.

### *Common REST Resources*

Servers generally return JSON-format resources, though resource formats can depend on the implementation.

Resources in collections can be found by their unique identifiers (IDs). IDs are exposed in the resource URIs. For example, if a server has a user collection under `/users`, then

you can access a user at `/users/user-id`. The ID is also the value of the `_id` field of the resource.

Resources are versioned using revision numbers. A revision is specified in the resource's `_rev` field. Revisions make it possible to figure out whether to apply changes without resource locking and without distributed transactions.

### *Common REST Verbs*

The Common REST APIs use the following verbs, sometimes referred to collectively as `CRUDPAQ`. For details and HTTP-based examples of each, follow the links to the sections for each verb.

***Create***
: Add a new resource.

  This verb maps to HTTP PUT or HTTP POST.

  For details, see Create.

***Read***
: Retrieve a single resource.

  This verb maps to HTTP GET.

  For details, see Read.

***Update***
: Replace an existing resource.

  This verb maps to HTTP PUT.

  For details, see Update.

***Delete***
: Remove an existing resource.

  This verb maps to HTTP DELETE.

  For details, see Delete.

***Patch***
: Modify part of an existing resource.

  This verb maps to HTTP PATCH.

  For details, see Patch.

***Action***
: Perform a predefined action.

This verb maps to HTTP POST.

For details, see Action.

*Query*
: Search a collection of resources.

  This verb maps to HTTP GET.

  For details, see Query.

### *Common REST Parameters*

Common REST reserved query string parameter names start with an underscore, `_`. Reserved query string parameters include, but are not limited to, the following names:

- `_action`
- `_api`
- `_crestapi`
- `_fields`
- `_mimeType`
- `_pageSize`
- `_pagedResultsCookie`
- `_pagedResultsOffset`
- `_prettyPrint`
- `_queryExpression`
- `_queryFilter`
- `_queryId`
- `_sortKeys`
- `_totalPagedResultsPolicy`

NOTE

Some parameter values are not safe for URLs, so URL-encode parameter values as necessary.

Continue reading for details about how to use each parameter.

### *Common REST Extension Points*

The *action* verb is the main vehicle for extensions. For example, to create a new user with HTTP POST rather than HTTP PUT, you might use `/users?_action=create`. A server can define additional actions. For example, `/tasks/1?_action=cancel`.

A server can define *stored queries* to call by ID. For example, `/groups?_queryId=hasDeletedMembers`. Stored queries can call for additional parameters. The parameters are also passed in the query string. Which parameters are valid depends on the stored query.

### *Common REST API Documentation*

Common REST APIs often depend at least in part on runtime configuration. Many Common REST endpoints therefore serve *API descriptors* at runtime. An API descriptor documents the actual API as it is configured.

Use the following query string parameters to retrieve API descriptors:

***_api***
: Serves an API descriptor that complies with the OpenAPI specification.

  This API descriptor represents the API accessible over HTTP. It is suitable for use with popular tools such as Swagger UI.

***_crestapi***
: Serves a native Common REST API descriptor.

  This API descriptor provides a compact representation that is not dependent on the transport protocol. It requires a client that understands Common REST, as it omits many Common REST defaults.

NOTE

Consider limiting access to API descriptors in production environments in order to avoid unnecessary traffic.

To provide documentation in production environments, see To Publish OpenAPI Documentation instead.

### *To Publish OpenAPI Documentation*

In production systems, developers expect stable, well-documented APIs. Rather than retrieving API descriptors at runtime through Common REST, prepare final versions, and publish them alongside the software in production.

Use the OpenAPI-compliant descriptors to provide API reference documentation for your developers:

1. Configure the software to produce production-ready APIs.

   In other words, configure the software as for production so that the APIs match exactly.

2. Retrieve the OpenAPI-compliant descriptor.

The following command saves the descriptor to a file, `myapi.json`:

```
$ curl -o myapi.json endpoint?_api
```

3. If necessary, edit the descriptor.

   For example, add security definitions to describe the API protection.

4. Publish the descriptor using a tool such as Swagger UI.

### *Create*

There are two ways to create a resource, HTTP POST or HTTP PUT.

To create a resource using POST, perform an HTTP POST with the query string parameter `_action=create`, and the JSON resource as a payload. Accept a JSON response. The server creates the identifier if not specified:

```
POST /users?_action=create HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
{ JSON resource }
```

To create a resource using PUT, perform an HTTP PUT including the case-sensitive identifier for the resource in the URL path, and the JSON resource as a payload. Use the `If-None-Match: *` header. Accept a JSON response:

```
PUT /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-None-Match: *
{ JSON resource }
```

The `_id` and content of the resource depend on the server implementation. The server is not required to use the `_id` that the client provides. The server response to the request indicates the resource location as the value of the `Location` header.

If you include the `If-None-Match` header, you must use `If-None-Match: *`. In this case, the request creates the object if it does not exist, and fails if the object does exist. If you include any value other `If-None-Match: *`, the server returns an HTTP 400 Bad

Request error. For example, creating an object with `If-None-Match: revision` returns a bad request error.

If you do not include `If-None-Match: *`, the request creates the object if it does not exist, and *updates* the object if it does exist.

Parameters

***_fields=field[,field…]***

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

***_prettyPrint=true***

Format the body of the response.

### *Read*

To retrieve a single resource, perform an HTTP GET on the resource by its case-sensitive identifier ( `_id` ), and accept a JSON response:

```
GET /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
```

Parameters

***_fields=field[,field…]***

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

***_mimeType=mime-type***

Some resources have fields whose values are multi-media resources, such as a profile photo.

If the feature is enabled for the endpoint, you can read a single field that is a multi-media resource by specifying the *field* and *mime-type*.

In this case, the content type of the field value returned matches the *mime-type* that you specify, and the body of the response is the multi-media resource.

Do not use the `Accept` header in this case. For example, `Accept: image/png` does not work. Use the `_mimeType` query string parameter instead.

***_prettyPrint=true***
: Format the body of the response.

### *Update*

To update a resource, perform an HTTP PUT including the case-sensitive identifier ( `_id` ) as the final element of the path to the resource, and the JSON resource as the payload. Use the `If-Match: _rev` header to check that you are actually updating the version you modified. Use `If-Match: *` if the version does not matter. Accept a JSON response:

```
PUT /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-Match: _rev
{ JSON resource }
```

When updating a resource, include all the attributes to retain. Omitting an attribute in the resource amounts to deleting the attribute unless it is not under the control of your application. Attributes not under the control of your application include private and read-only attributes. In addition, virtual attributes and relationship references might not be under the control of your application.

Parameters

***_fields=field[,field…]***
: Return only the specified fields in the body of the response.

  The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

  If the `field` is left blank, the server returns all default values.

***_prettyPrint=true***
: Format the body of the response.

### *Delete*

To delete a single resource, perform an HTTP DELETE by its case-sensitive identifier ( `\_id` ) and accept a JSON response:

```
DELETE /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
```

Parameters

***_fields=field[,field…]***

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

***_prettyPrint=true***

Format the body of the response.

### *Patch*

To patch a resource, send an HTTP PATCH request with the following parameters:

- `operation`
- `field`
- `value`
- `from` (optional with copy and move operations)

You can include these parameters in the payload for a PATCH request, or in a JSON PATCH file. If successful, you'll see a JSON response similar to the following:

```
PATCH /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-Match: _rev
{ JSON array of patch operations }
```

PATCH operations apply to three types of targets:

- **single-valued**, such as an object, string, boolean, or number.
- **list semantics array**, where the elements are ordered, and duplicates are allowed.
- **set semantics array**, where the elements are not ordered, and duplicates are not allowed.

ForgeRock PATCH supports multiple `operations`:

#### *Patch Operation: Add*

The `add` operation ensures that the target field contains the value provided, creating parent fields as necessary.

If the target field is single-valued, then the value you include in the PATCH replaces the value of the target. A single-valued field is an `object`, `string`, `boolean`, or `number`.

An `add` operation has different results on two standard types of arrays:

- **List semantic arrays**: you can run any of these `add` operations on that type of array:
  - If you `add` an array of values, the PATCH operation appends it to the existing list of values.
  - If you `add` a single value, specify an ordinal element in the target array, or use the `{-}` special index to add that value to the end of the list.
- **Set semantic arrays**: The value included in the patch is merged with the existing set of values. Any duplicates within the array are removed.

As an example, start with the following list semantic array resource:

```
{
    "fruits" : [ "orange", "apple" ]
}
```

The following add operation includes the pineapple to the end of the list of fruits, as indicated by the `-` at the end of the `fruits` array.

```
{
    "operation" : "add",
    "field" : "/fruits/-",
    "value" : "pineapple"
}
```

The following is the resulting resource:

```
{
    "fruits" : [ "orange", "apple", "pineapple" ]
}
```

You can add only one array element one at a time, as per the corresponding JSON Patch specification. If you add an array of elements, for example:

```
{
    "operation" : "add",
    "field" : "/fruits/-",
    "value" : ["pineapple", "mango"]
}
```

The resulting resource would have the following invalid JSON structure:

```
{
    "fruits" : [ "orange", "apple", ["pineapple", "mango"]]
}
```

*Patch Operation: Copy*

The copy operation takes one or more existing values from the source field. It then adds those same values on the target field. Once the values are known, it is equivalent to performing an `add` operation on the target field.

The following `copy` operation takes the value from a field named `mail`, and then runs a `replace` operation on the target field, `another_mail`.

```
[
  {
     "operation":"copy",
     "from":"mail",
     "field":"another_mail"
  }
]
```

If the source and target field values are arrays, the result depends on whether the array has list semantics or set semantics, as described in Patch Operation: Add.

*Patch Operation: Increment*

The `increment` operation changes the value or values of the target field by the amount you specify. The value that you include must be one number, and may be positive or negative. The value of the target field must accept numbers. The following `increment` operation adds `1000` to the target value of `/user/payment`.

```
[
  {
    "operation" : "increment",
    "field" : "/user/payment",
    "value" : "1000"
  }
]
```

Since the `value` of the `increment` is a single number, arrays do not apply.

*Patch Operation: Move*

The move operation removes existing values on the source field. It then adds those same values on the target field. This is equivalent to a `remove` operation on the source, followed by an `add` operation with the same values, on the target.

The following `move` operation is equivalent to a `remove` operation on the source field, `surname`, followed by a `replace` operation on the target field value, `lastName`. If the target field does not exist, it is created:

```
[
  {
     "operation":"move",
     "from":"surname",
     "field":"lastName"
  }
]
```

To apply a `move` operation on an array, you need a compatible single-value, list semantic array, or set semantic array on both the source and the target. For details, see the criteria described in Patch Operation: Add.

*Patch Operation: Remove*

The `remove` operation ensures that the target field no longer contains the value provided. If the remove operation does not include a value, the operation removes the field. The following `remove` deletes the value of the `phoneNumber`, along with the field.

```
[
  {
    "operation" : "remove",
    "field" : "phoneNumber"
  }
]
```

If the object has more than one `phoneNumber`, those values are stored as an array.

A `remove` operation has different results on two standard types of arrays:

- **List semantic arrays**: A `remove` operation deletes the specified element in the array. For example, the following operation removes the first phone number, based on its array index (zero-based):

```
[
   {
      "operation" : "remove",
      "field" : "/phoneNumber/0"
   }
]
```

- **Set semantic arrays**: The list of values included in a patch are removed from the existing array.

*Patch Operation: Replace*

The `replace` operation removes any existing value(s) of the targeted field, and replaces them with the provided value(s). It is essentially equivalent to a `remove` followed by a `add` operation. If the arrays are used, the criteria is based on Patch Operation: Add. However, indexed updates are not allowed, even when the target is an array.

The following `replace` operation removes the existing `telephoneNumber` value for the user, and then adds the new value of `+1 408 555 9999`.

```
[
  {
    "operation" : "replace",
    "field" : "/telephoneNumber",
    "value" : "+1 408 555 9999"
  }
]
```

A PATCH replace operation on a list semantic array works as a PATCH remove operation. The following example demonstrates how the effect of both operations. Start with the following resource:

```
{
    "fruits" : [ "apple", "orange", "kiwi", "lime" ],
}
```

Apply the following operations on that resource:

```
[
  {
    "operation" : "remove",
    "field" : "/fruits/0",
    "value" : ""
  },
```

```
  {
    "operation" : "replace",
    "field" : "/fruits/1",
    "value" : "pineapple"
  }
]
```

The PATCH operations are applied sequentially. The `remove` operation removes the first member of that resource, based on its array index, ( `fruits/0` ), with the following result:

```
[
  {
    "fruits" : [ "orange", "kiwi", "lime" ],
  }
]
```

The second PATCH operation, a `replace` , is applied on the second member ( `fruits/1` ) of the intermediate resource, with the following result:

```
[
  {
    "fruits" : [ "orange", "pineapple", "lime" ],
  }
]
```

*Patch Operation: Transform*

The `transform` operation changes the value of a field based on a script, or some other data transformation command. The following `transform` operation takes the value from the field named `/objects` , and applies the `something.js` script as shown:

```
[
  {
    "operation" : "transform",
    "field" : "/objects",
    "value" : {
      "script" : {
        "type" : "text/javascript",
        "file" : "something.js"
      }
    }
```

```
  }
]
```

*Patch Operation Limitations*

Some HTTP client libraries do not support the HTTP PATCH operation. Make sure that the library you use supports HTTP PATCH before using this REST operation.

For example, the Java Development Kit HTTP client does not support PATCH as a valid HTTP method. Instead, the method `HttpURLConnection.setRequestMethod("PATCH")` throws `ProtocolException`.

Parameters

***_fields=field[,field…]***
: Return only the specified fields in the body of the response.

  The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

  If the `field` is left blank, the server returns all default values.

***_prettyPrint=true***
: Format the body of the response.

## *Action*

Actions are a means of extending Common REST APIs and are defined by the resource provider, so the actions you can use depend on the implementation.

The standard action indicated by `_action=create` is described in Create.

Parameters
In addition to these parameters, specific action implementations have their own parameters:

***_fields=field[,field…]***
: Return only the specified fields in the body of the response.

  The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

  If the `field` is left blank, the server returns all default values.

***_prettyPrint=true***
: Format the body of the response.

## *Query*

To query a resource collection (or resource container), perform an HTTP GET, and accept a JSON response, including either a `_queryExpression`, `_queryFilter`, or `_queryId` parameter. The parameters cannot be used together:

```
GET /users?_queryFilter=true HTTP/1.1
Host: example.com
Accept: application/json
```

The server returns the result as a JSON object including a `"results"` array, and other fields that depend on the parameters.

Parameters

***_fields=field[,field…]***

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

***_queryFilter=filter-expression***

Query filters request that the server return entries that match the filter expression. You must URL-escape the filter expression.

The string representation is summarized as follows. Continue reading for additional explanation:

```
Expr           = OrExpr
OrExpr         = AndExpr ( 'or' AndExpr ) *
AndExpr        = NotExpr ( 'and' NotExpr ) *
NotExpr        = '!' PrimaryExpr | PrimaryExpr
PrimaryExpr    = '(' Expr ')' | ComparisonExpr | PresenceExpr |
LiteralExpr
ComparisonExpr = Pointer OpName JsonValue
PresenceExpr   = Pointer 'pr'
LiteralExpr    = 'true' | 'false'
Pointer        = JSON pointer
OpName         = 'eq' |  # equal to
                 'co' |  # contains
                 'sw' |  # starts with
                 'lt' |  # less than
                 'le' |  # less than or equal to
                 'gt' |  # greater than
                 'ge' |  # greater than or equal to
                 STRING  # extended operator
JsonValue      = NUMBER | BOOLEAN | '"' UTF8STRING '"'
```

```
STRING          = ASCII string not containing white-space
UTF8STRING      = UTF-8 string possibly containing white-space
```

*JsonValue* components of filter expressions follow RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format. In particular, as described in section 7 of the RFC, the escape character in strings is the backslash character. For example, to match the identifier `test\`, use `_id eq 'test\\'`. In the JSON resource, the `\` is escaped the same way: `"_id":"test\\"`.

When using a query filter in a URL, the filter expression is part of a query string parameter. A query string parameter must be URL encoded, as described in RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. For example, white space, double quotes ( " ), parentheses, and exclamation characters must be URL encoded in HTTP query strings. The following rules apply to URL query components:

```
query       = *( pchar / "/" / "?" )
pchar       = unreserved / pct-encoded / sub-delims / ":" / "@"
unreserved  = ALPHA / DIGIT / "-" / "." / "_" / "~"
pct-encoded = "%" HEXDIG HEXDIG
sub-delims  = "!" / "$" / "&amp;" / "'" / "(" / ")"
                  / "*" / "+" / "," / ";" / "="
```

`ALPHA`, `DIGIT`, and `HEXDIG` are core rules of RFC 5234: Augmented BNF for Syntax Specifications:

```
ALPHA       =  %x41-5A / %x61-7A   ; A-Z / a-z
DIGIT       =  %x30-39             ; 0-9
HEXDIG      =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
```

As a result, a backslash escape character in a *JsonValue* component is percent-encoded in the URL query string parameter as `%5C`. To encode the query filter expression `_id eq 'test\\'`, use `_id+eq+'test%5C%5C'`, for example.

A simple filter expression can represent a comparison, presence, or a literal value.

For comparison expressions, use *`json-pointer comparator json-value`*, where the *comparator* is one of the following:

```
eq (equals)
co (contains)
sw (starts with)
lt (less than)
le (less than or equal to)
```

```
gt (greater than)
ge (greater than or equal to)
```

For presence, use *`json-pointer pr`* to match resources where the JSON pointer is present, and the value it points to is not `null`.

Literal values include `true` (match anything) and `false` (match nothing).

Complex expressions employ `and`, `or`, and `!` (not), with parentheses, (*`expression`*), to group expressions.

***`_queryId=identifier`***

Specify a query by its identifier.

Specific queries can take their own query string parameter arguments, which depend on the implementation.

***`_pagedResultsCookie=string`***

The string is an opaque cookie used by the server to keep track of the position in the search results. The server returns the cookie in the JSON response as the value of `pagedResultsCookie`.

In the request `_pageSize` must also be set and non-zero. You receive the cookie value from the provider on the first request, and then supply the cookie value in subsequent requests until the server returns a `null` cookie, meaning the final page of results has been returned.

The `_pagedResultsCookie` parameter is supported when used with the `_queryFilter` parameter. The `_pagedResultsCookie` parameter is not guaranteed to work with the `_queryExpression` or `_queryId` parameters.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and not to be used together.

***`_pagedResultsOffset=integer`***

When `_pageSize` is non-zero, use this as an index in the result set indicating the first page to return.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and not to be used together.

***`_pageSize=integer`***

Return query results in pages of this size. After the initial request, use `_pagedResultsCookie` or `_pageResultsOffset` to page through the results.

***`_prettyPrint=true`***

Format the body of the response.

***`_totalPagedResultsPolicy=string`***

When a `_pageSize` is specified, and non-zero, the server calculates the `"totalPagedResults"`, in accordance with the `totalPagedResultsPolicy`, and provides the value as part of the response.

The `"totalPagedResults"` is either an estimate of the total number of paged results (`_totalPagedResultsPolicy=ESTIMATE`), or the exact total result count (`_totalPagedResultsPolicy=EXACT`). If no count policy is specified in the query, or if `_totalPagedResultsPolicy=NONE`, result counting is disabled, and the server returns value of -1 for `"totalPagedResults"`.

***_sortKeys=(|-)__field__[,(|-)field…]***

Sort the resources returned based on the specified field(s), either in `+` (ascending, default) order, or in `-` (descending) order.

Because ascending order is the default, including the `` ` `` character in the query is unnecessary. If you do include the `` ` `` character, it must be URL-encoded as `%2B`, for example:

```
http://localhost:8080/api/users?
_queryFilter=true&_sortKeys=%2Bname/givenName
```

The `_sortKeys` parameter is not supported for predefined queries (`_queryId`).

## *HTTP Status Codes*

When working with a Common REST API over HTTP, client applications should expect at least these HTTP status codes. Not all servers necessarily return all status codes identified here:

***200 OK***

The request was successful and a resource returned, depending on the request.

***201 Created***

The request succeeded and the resource was created.

***204 No Content***

The action request succeeded, and there was no content to return.

***304 Not Modified***

The read request included an `If-None-Match` header, and the value of the header matched the revision value of the resource.

***400 Bad Request***

The request was malformed.

***401 Unauthorized***

The request requires user authentication.

*403 Forbidden*
: Access was forbidden during an operation on a resource.

*404 Not Found*
: The specified resource could not be found, perhaps because it does not exist.

*405 Method Not Allowed*
: The HTTP method is not allowed for the requested resource.

*406 Not Acceptable*
: The request contains parameters that are not acceptable, such as a resource or protocol version that is not available.

*409 Conflict*
: The request would have resulted in a conflict with the current state of the resource.

*410 Gone*
: The requested resource is no longer available, and will not become available again. This can happen when resources expire for example.

*412 Precondition Failed*
: The resource's current version does not match the version provided.

*415 Unsupported Media Type*
: The request is in a format not supported by the requested resource for the requested method.

*428 Precondition Required*
: The resource requires a version, but no version was supplied in the request.

*500 Internal Server Error*
: The server encountered an unexpected condition that prevented it from fulfilling the request.

*501 Not Implemented*
: The resource does not support the functionality required to fulfill the request.

*503 Service Unavailable*
: The requested resource was temporarily unavailable. The service may have been disabled, for example.

## API Versions

DS REST APIs support versioning. If DS exposes multiple versions of a REST API, then you must select the version. In your HTTP requests, set a version header to specify the resource version:

```
Accept-API-Version: resource=version
```

Here, version is the `version` in the REST to LDAP mapping file. For details, see API Configuration.

If you do not set a version header, then the latest version is returned.

The default example configuration includes only one API, whose version is `1.0`. In this case, the header can be omitted. If the version were specified in the examples that follow, the appropriate header would be `Accept-API-Version: resource=1.0`.

## Authentication

When you first try to read a resource without authenticating, the request results in an error:

```
$ curl --cacert ca-cert.pem
https://localhost:8443/api/users/bjensen

{"code":401,"reason":"Unauthorized","message":"Invalid
Credentials"}
```

HTTP status code 401 indicates that the request requires user authentication.

To disable the requirement to authenticate, set the Rest2ldap endpoint `authorization-mechanism` to map anonymous HTTP requests to LDAP requests performed by an authorized user. The following example uses Kirsten Vaughan's identity:

▼ Show example

```
$ dsconfig \
 set-http-authorization-mechanism-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --mechanism-name "HTTP Anonymous" \
 --set enabled:true \
 --set user-dn:uid=kvaughan,ou=people,dc=example,dc=com \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
```

```
set-http-endpoint-prop \
--hostname localhost \
--port 4444 \
--bindDN uid=admin \
--bindPassword password \
--endpoint-name "/api" \
--set authorization-mechanism:"HTTP Anonymous" \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--no-prompt
```

By default, both the Rest2ldap endpoint and also the REST to LDAP gateway allow HTTP Basic authentication and HTTP header-based authentication in the style of IDM software. The authentication mechanisms map HTTP credentials to LDAP credentials.

When you set up a directory server with the `ds-evaluation` profile, the relative distinguished name (RDN) attribute is the user ID ( `uid` ). For example, the DN and user ID for Babs Jensen are:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
uid: bjensen
```

Given this pattern in the user entries, the default sample REST to LDAP configuration translates the HTTP user name to the LDAP user ID. The default sample finds user entries under `ou=People,dc=example,dc=com` . (The default sample mapping requires that LDAP entries be immediate subordinates of the mapping's base DN.) Babs Jensen authenticates as `bjensen` (password: `hifalutin` ) over HTTP. The corresponding LDAP bind DN is `uid=bjensen,ou=People,dc=example,dc=com` :

- Example using HTTP Basic authentication:

```
$ curl \
 --user bjensen:hifalutin \
 --cacert ca-cert.pem \
 --silent \
 https://localhost:8443/api/users/bjensen?_fields=userName

{"_id":"bjensen","_rev":"
<revision>","userName":"bjensen@example.com"}
```

- Example using HTTP Basic authentication with credentials in the URL ( *`username:password`*`@` form):

```
$ curl \
 --cacert ca-cert.pem \
```

```
 --silent \
 https://bjensen:hifalutin@localhost:8443/api/users/bjensen?
_fields=userName

{"_id":"bjensen","_rev":"
<revision>","userName":"bjensen@example.com"}
```

- Example using HTTP header based authentication:

```
$ curl \
 --header "X-OpenIDM-Username: bjensen" \
 --header "X-OpenIDM-Password: hifalutin" \
 --cacert ca-cert.pem \
 --silent \
 https://localhost:8443/api/users/bjensen?_fields=userName

{"_id":"bjensen","_rev":"
<revision>","userName":"bjensen@example.com"}
```

If the directory data is arranged differently, or if the user names are email addresses rather than user IDs, then you must update the configuration for authentication to work.

The REST to LDAP gateway can translate HTTP username/password authentication to LDAP PLAIN SASL authentication. The gateway falls back to proxied authorization, binding as the directory superuser by default. For details, see REST to LDAP Reference.

## DS Query Parameters

REST to LDAP supports the following additional query string parameters:

- `dryRun` (boolean) to discover how a server will react to a operation, without performing the operation.

  This parameter relies on the LDAP no-op control, OID `1.3.6.1.4.1.4203.1.10.2`.

  You can use with parameter with CREST create, update, patch, and the password-related actions, `modifyPassword` and `resetPassword`.

  Default: `false`

- `passwordQualityAdvice` (boolean) to get additional information for a failed password update.

  The `passwordQualityAdvice` parameter relies on the DS LDAP password quality advice control, OID `1.3.6.1.4.1.36733.2.1.5.5`.

You can use with parameter with CREST create, update, patch, and the password-related actions, `modifyPassword` and `resetPassword`.

Default: `false`

# Create

NOTE

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see Learn About the Evaluation Setup Profile.

## Create (HTTP PUT)

To choose the identifier when creating a resource, perform an HTTP PUT request with the headers `Content-Type: application/json` and `If-None-Match: *`, and the JSON content of your resource:

```
$ curl \
 --request PUT \
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --header "If-None-Match: *" \
 --data '{
  "_id": "newuser",
  "_schema":"frapi:opendj:rest2ldap:user:1.0",
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1212",
    "emailAddress": "newuser@example.com"
  },
  "name": {
    "familyName": "New",
    "givenName": "User"
  },
  "displayName": ["New User"],
  "manager": {
    "_id": "kvaughan"
  }
 }' \
 --silent \
 https://localhost:8443/api/users/newuser?_prettyPrint=true
```

```
{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName" : "User",
    "familyName" : "New"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "manager" : {
    "_id" : "kvaughan",
    "_rev": "<revision>"
  }
}
```

## Create (HTTP POST)

To let the server choose the identifier when creating a resource, perform an HTTP POST with the header `Content-Type: application/json`, and the JSON content of the resource.

Including `_action=create` in the query string is optional:

```
$ curl \
 --request POST \
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --data '{
  "_id": "newuser",
  "_schema": "frapi:opendj:rest2ldap:user:1.0",
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1212",
    "emailAddress": "newuser@example.com"
  },
```

```
  "name": {
    "familyName": "New",
    "givenName": "User"
  },
  "displayName": ["New User"],
  "manager": {"_id": "kvaughan"}
 }' \
 --silent \
 https://localhost:8443/api/users?_prettyPrint=true

{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName" : "User",
    "familyName" : "New"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "manager" : {
    "_id" : "kvaughan",
    "_rev" : "<revision>"
  }
}
```

# Read

NOTE

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see Learn About the Evaluation Setup Profile.

## Read a Resource

Perform an HTTP GET:

```
$ curl \
 --request GET \
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 --silent \
 "https://localhost:8443/api/users/newuser?_prettyPrint=true"

{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName" : "User",
    "familyName" : "New"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "manager" : {
    "_id" : "kvaughan",
    "_rev": "<revision>"
  }
}
```

## Referenced Resource Fields

Notice in the preceding example that the operation returns only the manager's `_id` and `_rev` fields. The fields returned depend on how the reference is configured in the REST to LDAP mapping.

To return additional fields from resources referenced with a resource path in the REST to LDAP mapping, explicitly specify the field names:

```
$ curl \
 --request GET \
 --cacert ca-cert.pem \
```

```
 --user kvaughan:bribery \
 --silent \
 "https://localhost:8443/api/users/newuser?
_fields=/displayName,/manager/displayName&_prettyPrint=true"

{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "displayName" : [ "New User" ],
  "manager" : {
    "_id" : "kvaughan",
    "displayName" : [ "Kirsten Vaughan" ],
    "_rev" : "<revision>"
  }
}
```

To return all configured fields for the resource and the manager, use `_fields=/,/manager`. This returns all fields of the referenced manager resource that are configured for the REST to LDAP mapping.

## Reverse References

When you read a manager's entry, the `"reverseReference"` field in the mapping returns the list of users reporting to the manager when the reverse reference field is explicitly requested.

The search is not indexed by default, so the directory superuser makes the request:

```
$ curl \
 --request GET \
 --cacert ca-cert.pem \
 --user admin:password \
 --silent \
 "https://localhost:8443/api/users/kvaughan?
_fields=/reports/displayName&_prettyPrint=true"

{
  "_id" : "kvaughan",
  "_rev" : "<revision>",
  "reports" : [ {
    "_id" : "ashelton",
    "displayName" : [ "Alexander Shelton" ],
    "_rev" : "<revision>"
  }, {
    "_id" : "btalbot",
```

```
  "displayName" : [ "Brad Talbot" ],
  "_rev" : "<revision>"
}, {
  "_id" : "dakers",
  "displayName" : [ "David Akers" ],
  "_rev" : "<revision>"
}, {
  "_id" : "dsmith",
  "displayName" : [ "Daniel Smith" ],
  "_rev" : "<revision>"
}, {
  "_id" : "eward",
  "displayName" : [ "Eric Ward" ],
  "_rev" : "<revision>"
}, {
  "_id" : "gjensen",
  "displayName" : [ "Gern Jensen" ],
  "_rev" : "<revision>"
}, {
  "_id" : "hmiller",
  "displayName" : [ "Harry Miller" ],
  "_rev" : "<revision>"
}, {
  "_id" : "jburrell",
  "displayName" : [ "James Burrell" ],
  "_rev" : "<revision>"
}, {
  "_id" : "jcampai2",
  "displayName" : [ "Jeffrey Campaigne" ],
  "_rev" : "<revision>"
}, {
  "_id" : "jfalena",
  "displayName" : [ "John Falena" ],
  "_rev" : "<revision>"
}, {
  "_id" : "jvaughan",
  "displayName" : [ "Jeff Vaughan" ],
  "_rev" : "<revision>"
}, {
  "_id" : "kcarter",
  "displayName" : [ "Karen Carter" ],
  "_rev" : "<revision>"
}, {
  "_id" : "mreuter",
  "displayName" : [ "Matthew Reuter" ],
```

```
    "_rev" : "<revision>"
  }, {
    "_id" : "newuser",
    "displayName" : [ "New User" ],
    "_rev" : "<revision>"
  }, {
    "_id" : "pworrell",
    "displayName" : [ "Pete Worrell" ],
    "_rev" : "<revision>"
  }, {
    "_id" : "rbannist",
    "displayName" : [ "Richard Bannister" ],
    "_rev" : "<revision>"
  }, {
    "_id" : "rdaugherty",
    "displayName" : [ "Robert Daugherty" ],
    "_rev" : "<revision>"
  }, {
    "_id" : "rschneid",
    "displayName" : [ "Rachel Schneider" ],
    "_rev" : "<revision>"
  }, {
    "_id" : "striplet",
    "displayName" : [ "Stephen Triplett" ],
    "_rev" : "<revision>"
  }, {
    "_id" : "tclow",
    "displayName" : [ "Torrey Clow" ],
    "_rev" : "<revision>"
  }, {
    "_id" : "tmason",
    "displayName" : [ "Torrey Mason" ],
    "_rev" : "<revision>"
  }, {
    "_id" : "tschmith",
    "displayName" : [ "Tobias Schmith" ],
    "_rev" : "<revision>"
  }, {
    "_id" : "tward",
    "displayName" : [ "Tobias Ward" ],
    "_rev" : "<revision>"
  } ]
}
```

Notice that the example explicitly requests reports with the `_fields` query parameter.

# Update

NOTE

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see Learn About the Evaluation Setup Profile.

To update a resource, replace it with an HTTP PUT that includes the case-sensitive identifier ( `_id` ) as the final element of the path, and a JSON payload that contains _all_ writable fields of the resource that you want to retain. Use an `If-Match` header to ensure the resource already exists. For read-only fields, either include unmodified versions, or omit them from your updated version.

To update a resource regardless of the revision, use an `If-Match: *` header:

```
$ curl \
 --request PUT \
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --header "If-Match: *" \
 --data '{
   "contactInformation": {
     "telephoneNumber": "+1 408 555 4798",
     "emailAddress": "scarter@example.com"
   },
   "name": {
     "familyName": "Carter",
     "givenName": "Sam"
   },
   "userName": "scarter@example.com",
   "displayName": ["Sam Carter", "Samantha Carter"],
   "groups": [
     {
       "_id": "Accounting Managers"
     }
   ],
   "manager": {
     "_id": "trigden"
   },
  "uidNumber": 1002,
  "gidNumber": 1000,
  "homeDirectory": "/home/scarter"
```

```
 }' \
 --silent \
 https://localhost:8443/api/users/scarter?_prettyPrint=true

{
  "_id" : "scarter",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "userName" : "scarter@example.com",
  "displayName" : [ "Sam Carter", "Samantha Carter" ],
  "name" : {
    "givenName" : "Sam",
    "familyName" : "Carter"
  },
  "description" : "Description on ou=People",
  "manager" : {
    "_id" : "trigden",
    "_rev" : "<revision>"
  },
  "groups" : [ {
    "_id" : "Accounting Managers",
    "_rev" : "<revision>"
  } ],
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 4798",
    "emailAddress" : "scarter@example.com"
  },
  "uidNumber" : 1002,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/scarter"
}
```

To update a resource only if the resource matches a particular version, use an `If-Match: revision` header as shown in the following example:

```
$ export REVISION=$(cut -d \" -f 8 <(curl --silent \
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 https://localhost:8443/api/users/scarter?_fields=_rev))

$ curl \
 --request PUT \
```

```
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 --header "If-Match: $REVISION" \
 --header "Content-Type: application/json" \
 --data '{
   "_id" : "scarter",
   "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
   "contactInformation": {
     "telephoneNumber": "+1 408 555 4798",
     "emailAddress": "scarter@example.com"
   },
   "name": {
     "familyName": "Carter",
     "givenName": "Sam"
   },
   "userName": "scarter@example.com",
   "displayName": ["Sam Carter", "Samantha Carter"],
  "uidNumber": 1002,
  "gidNumber": 1000,
  "homeDirectory": "/home/scarter"
 }' \
 --silent \
 https://localhost:8443/api/users/scarter?_prettyPrint=true

{
  "_id" : "scarter",
  "_rev" : "<new-revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "userName" : "scarter@example.com",
  "displayName" : [ "Sam Carter", "Samantha Carter" ],
  "name" : {
    "givenName" : "Sam",
    "familyName" : "Carter"
  },
  "description" : "Description on ou=People",
  "groups" : [ {
    "_id" : "Accounting Managers",
    "_rev" : "<revision>"
  } ],
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 4798",
    "emailAddress" : "scarter@example.com"
```

```
  },
  "uidNumber" : 1002,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/scarter"
}
```

## Delete

NOTE

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see Learn About the Evaluation Setup Profile.

To delete a resource, perform an HTTP DELETE on the resource URL. The operation returns the resource you deleted:

```
$ curl \
 --request DELETE \
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 --silent \
 https://localhost:8443/api/users/newuser?_prettyPrint=true

{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName" : "User",
    "familyName" : "New"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "manager" : {
    "_id" : "kvaughan",
```

```
    "_rev" : "<revision>"
  }
}
```

To delete a resource only if the resource matches a particular version, use an `If-Match: revision` header:

```
$ export REVISION=$(cut -d \" -f 8 <(curl --silent \
 --user kvaughan:bribery \
 --cacert ca-cert.pem \
 https://localhost:8443/api/users/newuser?_fields=_rev))

$ curl \
 --request DELETE \
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 --header "If-Match: $REVISION" \
 --silent \
 https://localhost:8443/api/users/newuser?_prettyPrint=true

{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName" : "User",
    "familyName" : "New"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "manager" : {
    "_id" : "kvaughan",
    "_rev" : "<revision>"
  }
}
```

To delete a resource and all of its children, follow these high-level steps:

- Make sure that the REST to LDAP configuration does map the resources to delete to LDAP entries.

  For an example, see Nested Resources.

- If you are using the gateway, this requires the default setting of true for `useSubtreeDelete` in `WEB-INF/classes/rest2ldap/rest2ldap.json`.

  NOTE

  Only users who have access to request a tree delete can delete resources with children.

- Allow the REST user to use the subtree delete control:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --add global-aci:"(targetcontrol=\"SubtreeDelete\")\
 (version 3.0; acl \"Allow Subtree Delete\"; allow(read) \

userdn=\"ldap:///uid=kvaughan,ou=People,dc=example,dc=com\";)"
\
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin
\
 --no-prompt
```

- Request the delete as a user who has rights to perform a subtree delete on the resource.

  NOTE

  This can be a resource-intensive operation. The resources required to remove a branch depend on the number of LDAP entries to delete.

## Patch

NOTE

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see Learn About the Evaluation Setup Profile.

Patching is updating part of the resource rather than replacing the resource. For example, you could change Babs Jensen's email address with an HTTP PATCH request:

```
$ curl \
 --user kvaughan:bribery \
 --cacert ca-cert.pem \
 --request PATCH \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "replace",
    "field": "/contactInformation/emailAddress",
    "value": "babs@example.com"
  }
 ]' \
 --silent \
 https://localhost:8443/api/users/bjensen?_prettyPrint=true

{
  "_id" : "bjensen",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "userName" : "babs@example.com",
  "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
  "name" : {
    "givenName" : "Barbara",
    "familyName" : "Jensen"
  },
  "description" : "Original description",
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1862",
    "emailAddress" : "babs@example.com"
  },
  "uidNumber" : 1076,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/bjensen",
  "manager" : {
    "_id" : "trigden",
    "_rev" : "<revision>"
  },
  "groups" : [ {
    "_id" : "Carpoolers"
```

```
  } ]
}
```

Notice in the example that the data sent specifies the type of patch operation, the field to change, and a value that depends on the field you change and on the operation. A single-valued field takes an object, boolean, string, or number depending on its type, whereas a multi-valued field takes an array of values. Getting the type wrong results in an error. Notice that the patch data is itself an array. This makes it possible to patch more than one part of the resource by using a set of patch operations in the same request.

DS software supports these patch operations:

***`add`***
: The add operation ensures that the target field contains the value provided, creating parent fields as necessary.

  If the target field is single-valued and a value already exists, then that value is replaced with the value you provide. *Note that you do not get an error when adding a value to a single-valued field that already has a value.* A single-valued field is one whose value is not an array (an object, string, boolean, or number).

  If the target field is multi-valued, then the array of values you provide is merged with the set of values already in the resource. New values are added, and duplicate values are ignored. A multi-valued field takes an array value.

***`remove`***
: The remove operation ensures that the target field does not contain the value provided. If you do not provide a value, the entire field is removed if it already exists.

  If the target field is single-valued and a value is provided, then the provided value must match the existing value to remove, otherwise the field is left unchanged.

  If the target field is multi-valued, then values in the array you provide are removed from the existing set of values.

***`replace`***
: The replace operation removes existing values on the target field, and replaces them with the values you provide. It is equivalent to performing a remove on the field, then an add with the values you provide.

***`increment`***
: The increment operation increments or decrements the value or values in the target field by the amount you specify, which is positive to increment and negative to decrement. The target field must take a number or a set of numbers. The value you provide must be a single number.

One key nuance in how a patch works with DS software concerns multi-valued fields. Although JSON resources represent multi-valued fields as *arrays*, DS software treats those values as *sets*. In other words, values in the field are unique, and the ordering of an array of values is not meaningful in the context of patch operations. If you reference array values by index, DS software returns an error. DS software does, however, allow use of a hyphen to add an element to a set. Include the hyphen as the last element of the `field` JSON pointer path as in the `"/members/-"` field of this example patch: `[{ "operation" : "add", "field" : "/members/-", "value" : { "_id" : "bjensen" } }]`.

Perform patch operations as if arrays values were sets. The following example includes Barbara Jensen in a group by adding her to the set of members:

```
$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --cacert ca-cert.pem \
 --header "Content-Type: application/json" \
 --data '[{
     "operation": "add",
     "field": "/members",
     "value": [{"_id": "bjensen"}]
   }]' \
 --silent \
 https://localhost:8443/api/groups/Directory%20Administrators?
_prettyPrint=true

{
  "_id" : "Directory Administrators",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:group:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "displayName" : "Directory Administrators",
  "members" : [ {
    "_id" : "kvaughan"
  }, {
    "_id" : "rdaugherty"
  }, {
    "_id" : "hmiller"
  }, {
    "_id" : "bjensen"
  } ]
}
```

The following example removes Barbara Jensen from the group:

```
$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --cacert ca-cert.pem \
 --header "Content-Type: application/json" \
 --data '[{
    "operation": "remove",
    "field": "/members",
    "value": [{"_id": "bjensen"}]
  }]' \
 --silent \
 https://localhost:8443/api/groups/Directory%20Administrators?
_prettyPrint=true

{
  "_id" : "Directory Administrators",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:group:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "displayName" : "Directory Administrators",
  "members" : [ {
    "_id" : "kvaughan"
  }, {
    "_id" : "rdaugherty"
  }, {
    "_id" : "hmiller"
  } ]
}
```

To change the value of more than one attribute in a patch operation, include multiple operations in the body of the JSON patch, as shown in the following example:

```
$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --cacert ca-cert.pem \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "replace",
```

```
    "field": "/contactInformation/telephoneNumber",
    "value": "+1 408 555 9999"
  },
  {
    "operation": "add",
    "field": "/contactInformation/emailAddress",
    "value": "barbara.jensen@example.com"
  }
 ]' \
 --silent \
 https://localhost:8443/api/users/bjensen?_prettyPrint=true

{
  "_id" : "bjensen",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "userName" : "barbara.jensen@example.com",
  "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
  "name" : {
    "givenName" : "Barbara",
    "familyName" : "Jensen"
  },
  "description" : "Original description",
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 9999",
    "emailAddress" : "barbara.jensen@example.com"
  },
  "uidNumber" : 1076,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/bjensen",
  "manager" : {
    "_id" : "trigden",
    "_rev" : "<revision>"
  },
  "groups" : [ {
    "_id" : "Carpoolers"
  } ]
}
```

Notice that for a multi-valued attribute, the `value` field takes an array, whereas the `value` field takes a single value for a single-valued field. For single-valued fields, an `add` operation has the same effect as a `replace` operation.

You can use resource revision numbers in `If-Match: revision` headers to patch the resource only if the resource matches a particular version, as shown in the following example:

```
$ export REVISION=$(cut -d \" -f 8 <(curl --silent \
 --user kvaughan:bribery \
 --cacert ca-cert.pem \
 https://localhost:8443/api/users/bjensen?_fields=_rev))

$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --cacert ca-cert.pem \
 --header "If-Match: $REVISION" \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "add",
    "field": "/contactInformation/emailAddress",
    "value": "babs@example.com"
  }
 ]' \
 --silent \
 https://localhost:8443/api/users/bjensen?_prettyPrint=true

{
  "_id" : "bjensen",
  "_rev" : "<new-revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "userName" : "babs@example.com",
  "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
  "name" : {
    "givenName" : "Barbara",
    "familyName" : "Jensen"
  },
  "description" : "Original description",
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 9999",
    "emailAddress" : "babs@example.com"
  },
  "uidNumber" : 1076,
  "gidNumber" : 1000,
```

```
  "homeDirectory" : "/home/bjensen",
  "groups" : [ {
    "_id" : "Carpoolers"
  } ],
  "manager" : {
    "_id" : "trigden",
    "_rev" : "<revision>"
  }
}
```

The resource revision changes when the patch is successful.

## Actions

NOTE

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see Learn About the Evaluation Setup Profile.

### Change Your Password

NOTE

This action *requires HTTPS* to avoid sending the password over an insecure connection.

Perform an HTTPS POST with the header `Content-Type: application/json`, `_action=modifyPassword` in the query string, and the old and new passwords in JSON format as the POST data.

The JSON POST DATA must include the following fields:

***`oldPassword`***
: The value of this field is the current password as a UTF-8 string.

***`newPassword`***
: The value of this field is the new password as a UTF-8 string.

On success, the HTTP status code is 200 OK, and the response body is an empty JSON resource:

```
$ curl \
 --request POST \
```

```
 --cacert ca-cert.pem \
 --user bjensen:hifalutin \
 --header "Content-Type: application/json" \
 --data '{"oldPassword": "hifalutin", "newPassword":
"chngthspwd"}' \
 --silent \
 https://localhost:8443/api/users/bjensen?_action=modifyPassword

{}
```

## Check Password Quality

The `passwordQualityAdvice` and `dryRun` query string parameters let you get additional information for a password update that might fail. The `passwordQualityAdvice` parameter relies on the LDAP password quality advice control, OID `1.3.6.1.4.1.36733.2.1.5.5`, which users must have access to request. The `dryRun` parameter relies on the LDAP no-op control, OID `1.3.6.1.4.1.4203.1.10.2`.

NOTE

The password quality advice control and the `passwordQualityAdvice` parameter have interface stability: *Evolving*.

The following commands demonstrate how the parameters cause the server to return information. On failure, the status code is HTTP 400 Bad Request, and the response is a JSON object listing what passed validation and what failed:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: cn=Minimum length policy,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: ds-pwp-password-policy
objectClass: ds-pwp-validator
objectClass: ds-pwp-length-based-validator
cn: Minimum length policy
ds-pwp-password-attribute: userPassword
ds-pwp-default-password-storage-scheme: PBKDF2-HMAC-SHA512
```

```
ds-pwp-length-based-min-password-length: 8
subtreeSpecification: {base "ou=people", specificationFilter "
(uid=bjensen)" }

dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetcontrol="PasswordQualityAdvice")
  (version 3.0; acl "Authenticated users can check password
quality";
  allow(read) userdn="ldap:///all";)
EOF

$ curl \
 --request POST \
 --cacert ca-cert.pem \
 --user bjensen:chngthspwd \
 --header "Content-Type: application/json" \
 --data '{"oldPassword": "chngthspwd", "newPassword": "passwd"}' \
 --silent \
 "https://localhost:8443/api/users/bjensen?
_action=modifyPassword&dryRun=true&passwordQualityAdvice=true"

{
  "code" : 400,
  "reason" : "Bad Request",
  "message" : "Constraint Violation: The provided new password
failed the validation checks defined in the server: The provided
password is shorter than the minimum required length of 8
characters",
  "detail" : {
    "passwordQualityAdvice" : {
      "passingCriteria" : [ ],
      "failingCriteria" : [ {
        "type" : "length-based",
        "parameters" : {
          "max-password-length" : 0,
          "min-password-length" : 8
        }
      } ],
      "attributeType" : "userPassword"
    }
  }
}
```

You can use `passwordQualityAdvice` without the `dryRun` parameter:

```
$ curl \
 --request POST \
 --cacert ca-cert.pem \
 --user bjensen:password \
 --header "Content-Type: application/json" \
 --data '{"oldPassword": "chngthspwd", "newPassword":
"hifalutin"}' \
 --silent \
 "https://localhost:8443/api/users/bjensen?
_action=modifyPassword&passwordQualityAdvice=true"
```

On success, the HTTP status code is 200 OK, and the response body is an empty JSON resource.

## Reset a Password

Whenever one user changes another user's password, DS servers consider it a password reset. Often, password policies specify that users must change their passwords again after a password reset.

NOTE

This action *requires HTTPS* to avoid sending the password over an insecure connection.

Perform an HTTPS POST with the header `Content-Type: application/json`, `_action=resetPassword` in the query string, and an empty JSON document ( `{}` ) as the POST data.

The following example demonstrates an administrator changing a user's password. Before trying this example, make sure the password administrator has been given the `password-reset` privilege. Otherwise, the password administrator has insufficient access. On success, the HTTP status code is 200 OK, and the response body is a JSON resource with a `generatedPassword` containing the new password:

```
$ curl \
 --request POST \
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --data '{}' \
 --silent \
 https://localhost:8443/api/users/bjensen?_action=resetPassword
```

```
{"generatedPassword":"<new-password>"}
```

As password administrator, provide the new, generated password to the user.

Use this feature in combination with a password policy that forces the user to change their password after a reset. For an example, see Require Password Change on Add or Reset.

## Account Usability Action

The `accountUsability` action lets a password administrator read information about whether the user can authenticate to the directory. This mirrors the LDAP Account Usability Control:

- The `"supportedActions"` list in the REST to LDAP mapping for the user must include the `"accountUsability"` action.

  This action is not in the `"supportedActions"` list by default.
- The remote LDAP directory service must support the LDAP control, which has OID `1.3.6.1.4.1.42.2.27.9.5.8`.
- The password administrator must be able to use the LDAP control.

Try the `accountUsability` action:

1. Edit the mapping configuration to include the `"accountUsability"` action in the list for the user resource:

   ```
   "supportedActions": [ "accountUsability","modifyPassword",
   "resetPassword" ],
   ```

2. Enable the password administrator to use the LDAP account usability control.

   The following example sets a global ACI for Kirsten Vaughan:

   ```
   $ dsconfig \
    set-access-control-handler-prop \
    --hostname localhost \
    --port 4444 \
    --bindDN uid=admin \
    --bindPassword password \
    --add global-aci:"(targetcontrol=\"AccountUsability\")\
    (version 3.0; acl \"Account usability access\";
   allow(read) \
   ```

```
userdn=\"ldap:///uid=kvaughan,ou=People,dc=example,dc=com\
";)" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

3. Use a password policy that produces results for account usability, as in the following example:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: cn=Lockout with max age and grace
logins,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: ds-pwp-password-policy
cn: Lockout with max age and grace logins
ds-pwp-password-attribute: userPassword
ds-pwp-default-password-storage-scheme: PBKDF2-HMAC-SHA256
ds-pwp-lockout-failure-expiration-interval: 10 m
ds-pwp-grace-login-count: 3
ds-pwp-lockout-duration: 5 m
ds-pwp-lockout-failure-count: 3
ds-pwp-max-password-age: 30 d
subtreeSpecification: { base "ou=people",
specificationFilter "(uid=bjensen)" }
EOF
```

4. Produce some account usability information on a user account:

```
$ curl \
 --user bjensen:wrong-password \
 --cacert ca-cert.pem \
 --silent \
 https://localhost:8443/api/users/bjensen?_fields=userName
```

```
$ curl \
 --user bjensen:wrong-password \
 --cacert ca-cert.pem \
 --silent \
 https://localhost:8443/api/users/bjensen?_fields=userName

$ curl \
 --user bjensen:wrong-password \
 --cacert ca-cert.pem \
 --silent \
 https://localhost:8443/api/users/bjensen?_fields=userName
```

5. Use the action to get account usability information:

```
$ curl \
 --request POST \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --data '{}' \
 --cacert ca-cert.pem \
 --silent \
 https://localhost:8443/api/users/bjensen?
_action=accountUsability

{"status":"locked","unlockIn":<seconds>}
```

The JSON response can contain these fields. Only the `"status"` property is always present in the response. Other fields are optional:

```
{
  "status": "string",              // One of "disabled", "locked",
"passwordExpired",
                                   // "mustChangePassword", or
"valid"
  "unlockIn": integer,             // Seconds until locked account
is unlocked
  "graceLoginsRemaining": integer, // Number of remaining
authentications allowed with
                                   // an expired password
  "passwordExpiresIn": integer,    // Seconds until password
expires
}
```

# Query

NOTE

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see Learn About the Evaluation Setup Profile.

To search, perform an HTTP GET with a `_queryFilter=expression` parameter. For details about the query filter expression, see Query.

NOTE

The `_queryId`, and `_totalPagedResultsPolicy` parameters, described in Query are not used in DS software at present.

The following table shows LDAP search filters and corresponding query filter expressions:

| LDAP Filter | REST Filter |
|---|---|
| `(&)` | `_queryFilter=true` |
| `(uid=*)` | `_queryFilter=_id+pr` |
| `(uid=bjensen)` | `_queryFilter=_id+eq+'bjensen'` |
| `(uid=*jensen*)` | `_queryFilter=_id+co+'jensen'` |
| `(uid=jensen*)` | `_queryFilter=_id+sw+'jensen'` |
| `(&(uid=*jensen*)(cn=babs*))` | `_queryFilter=(_id+co+'jensen'and+displayName+sw'babs')` |
| `(\|(uid=*jensen*)(cn=sam*))` | `_queryFilter=(_id+co+'jensen'or+displayName+sw'sam')` |
| `(!(uid=*jensen*))` | `_queryFilter=!(_id+co+'jensen')` |
| `(uid<=jensen)` | `_queryFilter=_id+le+'jensen'` |
| `(uid>=jensen)` | `_queryFilter=_id+ge+'jensen'` |

For query operations, the filter expression is constructed from the following building blocks. Make sure you URL-encode the filter expressions, which are shown here without URL-encoding to make them easier to read.

In filter expressions, the simplest json-pointer is a field of the JSON resource, such as `userName` or `id`. A json-pointer can also point to nested elements, as described in the JSON Pointer Internet-Draft:

*Comparison expressions*

Build filters using the following comparison expressions:

***`json-pointer eq json-value`***

Matches when the pointer equals the value, as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users?
_queryFilter=userName+eq+'babs@example.com'&_prettyPrint=tru
e"

{
  "result" : [ {
    "_id" : "bjensen",
    "_rev" : "<revision>",
    "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
    "_meta" : {
      "lastModified" : "<datestamp>"
    },
    "userName" : "babs@example.com",
    "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
    "name" : {
      "givenName" : "Barbara",
      "familyName" : "Jensen"
    },
    "description" : "Original description",
    "manager" : {
      "_id" : "trigden",
      "_rev" : "<revision>"
    },
    "contactInformation" : {
      "telephoneNumber" : "+1 408 555 9999",
      "emailAddress" : "babs@example.com"
    },
    "uidNumber" : 1076,
    "gidNumber" : 1000,
    "homeDirectory" : "/home/bjensen",
    "groups" : [ {
      "_id" : "Carpoolers"
```

```
    } ]
  } ],
  "resultCount" : 1,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

***json-pointer co json-value***

Matches when the pointer contains the value, as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users?
_queryFilter=userName+co+'jensen'&_fields=userName&_prettyPr
int=true"

{
  "result" : [ {
    "_id" : "ajensen",
    "_rev" : "<revision>",
    "userName" : "ajensen@example.com"
  }, {
    "_id" : "gjensen",
    "_rev" : "<revision>",
    "userName" : "gjensen@example.com"
  }, {
    "_id" : "jjensen",
    "_rev" : "<revision>",
    "userName" : "jjensen@example.com"
  }, {
    "_id" : "kjensen",
    "_rev" : "<revision>",
    "userName" : "kjensen@example.com"
  }, {
    "_id" : "rjensen",
    "_rev" : "<revision>",
    "userName" : "rjensen@example.com"
  }, {
    "_id" : "tjensen",
    "_rev" : "<revision>",
    "userName" : "tjensen@example.com"
```

```
  } ],
  "resultCount" : 6,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

***json-pointer sw json-value***

Matches when the pointer starts with the value, as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users?
_queryFilter=userName+sw+'ab'&_fields=userName&_prettyPrint=
true"

{
  "result" : [ {
    "_id" : "abarnes",
    "_rev" : "<revision>",
    "userName" : "abarnes@example.com"
  }, {
    "_id" : "abergin",
    "_rev" : "<revision>",
    "userName" : "abergin@example.com"
  } ],
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

***json-pointer lt json-value***

Matches when the pointer is less than the value, as in the following example:

```
$ curl \
 --user admin:password \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users?
```

```
_queryFilter=userName+lt+'ac'&_fields=userName&_prettyPrint=
true"

{
  "result" : [ {
    "_id" : "abarnes",
    "_rev" : "<revision>",
    "userName" : "abarnes@example.com"
  }, {
    "_id" : "abergin",
    "_rev" : "<revision>",
    "userName" : "abergin@example.com"
  } ],
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

***json-pointer le json-value***

Matches when the pointer is less than or equal to the value, as in the following example:

```
$ curl \
 --user admin:password \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users?
_queryFilter=userName+le+'ad'&_fields=userName&_prettyPrint=
true"

{
  "result" : [ {
    "_id" : "abarnes",
    "_rev" : "<revision>",
    "userName" : "abarnes@example.com"
  }, {
    "_id" : "abergin",
    "_rev" : "<revision>",
    "userName" : "abergin@example.com"
  }, {
    "_id" : "achassin",
    "_rev" : "<revision>",
    "userName" : "achassin@example.com"
```

```
  } ],
  "resultCount" : 3,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

***json-pointer gt json-value***

Matches when the pointer is greater than the value, as in the following example:

```
$ curl \
 --user admin:password \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users?
_queryFilter=userName+gt+'wa'&_fields=userName&_prettyPrint=
true"

{
  "result" : [ {
    "_id" : "wlutz",
    "_rev" : "<revision>",
    "userName" : "wlutz@example.com"
  } ],
  "resultCount" : 1,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

***json-pointer ge json-value***

Matches when the pointer is greater than or equal to the value, as in the following example:

```
$ curl \
 --user admin:password \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users?
_queryFilter=userName+ge+'va'&_fields=userName&_prettyPrint=
true"
```

```
{
  "result" : [ {
    "_id" : "wlutz",
    "_rev" : "<revision>",
    "userName" : "wlutz@example.com"
  } ],
  "resultCount" : 1,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

*Presence expression*

`json-pointer pr` matches any resource on which the json-pointer is present:

```
$ curl \
 --user kvaughan:bribery \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/groups?
_queryFilter=displayName+pr&_fields=displayName&_prettyPrint=tr
ue"

{
  "result" : [ {
    "_id" : "Accounting Managers",
    "_rev" : "<revision>",
    "displayName" : "Accounting Managers"
  }, {
    "_id" : "Directory Administrators",
    "_rev" : "<revision>",
    "displayName" : "Directory Administrators"
  }, {
    "_id" : "HR Managers",
    "_rev" : "<revision>",
    "displayName" : "HR Managers"
  }, {
    "_id" : "PD Managers",
    "_rev" : "<revision>",
    "displayName" : "PD Managers"
  }, {
    "_id" : "QA Managers",
    "_rev" : "<revision>",
    "displayName" : "QA Managers"
```

```
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

*Literal expressions*

`true` matches any resource in the collection.

`false` matches no resource in the collection.

In other words, you can list all resources in a collection as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/groups?
_queryFilter=true&_fields=_id&_prettyPrint=true"

{
  "result" : [ {
    "_id" : "Accounting Managers",
    "_rev" : "<revision>"
  }, {
    "_id" : "Directory Administrators",
    "_rev" : "<revision>"
  }, {
    "_id" : "HR Managers",
    "_rev" : "<revision>"
  }, {
    "_id" : "PD Managers",
    "_rev" : "<revision>"
  }, {
    "_id" : "QA Managers",
    "_rev" : "<revision>"
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

*Complex expressions*

Combine expressions using boolean operators `and`, `or`, and `!` (not), and by using parentheses (`expression`) with group expressions. The following example queries resources with last name Jensen and manager name starting with `Sam`:

```
$ curl \
 --user kvaughan:bribery \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users?_queryFilter=\
(name/familyName+co+'jensen'+and+manager/displayName+sw+'Sam')&
_fields=name/familyName,manager/displayName&_prettyPrint=true"

{
  "result" : [ {
    "_id" : "bjense2",
    "_rev" : "<revision>",
    "name" : {
      "familyName" : "Jensen"
    },
    "manager" : {
      "displayName" : [ "Sam Carter", "Samantha Carter" ],
      "_id" : "scarter",
      "_rev" : "<revision>"
    }
  }, {
    "_id" : "jjensen",
    "_rev" : "<revision>",
    "name" : {
      "familyName" : "Jensen"
    },
    "manager" : {
      "displayName" : [ "Sam Carter", "Samantha Carter" ],
      "_id" : "scarter",
      "_rev" : "<revision>"
    }
  }, {
    "_id" : "tjensen",
    "_rev" : "<revision>",
    "name" : {
      "familyName" : "Jensen"
    },
    "manager" : {
      "displayName" : [ "Sam Carter", "Samantha Carter" ],
      "_id" : "scarter",
```

```
      "_rev" : "<revision>"
    }
  } ],
  "resultCount" : 3,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

Notice the filters use the JSON pointers `name/familyName` and `manager/displayName` to identify the fields nested inside the `name` and `manager` objects.

## Graph-Like Queries

The default REST to LDAP sample mapping defines references with `resourcePath` fields. When the REST to LDAP mapping defines references to other resources in this way, the mapping supports recursion. Client applications can use query filters that traverse a "graph" of resources.

The following example gets users whose manager belongs to the `Directory Administrators` group. The search is not indexed by default, so the directory superuser makes the request:

```
$ curl \
 --user admin:password \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users/?
_queryFilter=/manager/groups/_id+eq+'Directory%20Administrators'&_
fields=_id&_prettyPrint=true"

{
  "result" : [ {
    "_id" : "ashelton",
    "_rev" : "<revision>"
  }, {
    "_id" : "btalbot",
    "_rev" : "<revision>"
  }, {
    "_id" : "dakers",
    "_rev" : "<revision>"
  }, {
    "_id" : "dsmith",
```

```
    "_rev" : "<revision>"
  }, {
    "_id" : "eward",
    "_rev" : "<revision>"
  }, {
    "_id" : "gjensen",
    "_rev" : "<revision>"
  }, {
    "_id" : "hmiller",
    "_rev" : "<revision>"
  }, {
    "_id" : "jburrell",
    "_rev" : "<revision>"
  }, {
    "_id" : "jcampai2",
    "_rev" : "<revision>"
  }, {
    "_id" : "jfalena",
    "_rev" : "<revision>"
  }, {
    "_id" : "jvaughan",
    "_rev" : "<revision>"
  }, {
    "_id" : "kcarter",
    "_rev" : "<revision>"
  }, {
    "_id" : "mreuter",
    "_rev" : "<revision>"
  }, {
    "_id" : "newuser",
    "_rev" : "<revision>"
  }, {
    "_id" : "pworrell",
    "_rev" : "<revision>"
  }, {
    "_id" : "rbannist",
    "_rev" : "<revision>"
  }, {
    "_id" : "rdaugherty",
    "_rev" : "<revision>"
  }, {
    "_id" : "rschneid",
    "_rev" : "<revision>"
  }, {
    "_id" : "striplet",
```

```
    "_rev" : "<revision>"
  }, {
    "_id" : "tclow",
    "_rev" : "<revision>"
  }, {
    "_id" : "tmason",
    "_rev" : "<revision>"
  }, {
    "_id" : "tschmith",
    "_rev" : "<revision>"
  }, {
    "_id" : "tward",
    "_rev" : "<revision>"
  } ],
  "resultCount" : 23,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

REST to LDAP translates such graph-like HTTP queries into a series of corresponding LDAP requests. Complex graph-like queries can have a significant server-side performance impact.

## Paged Results

You can page through search results using the following query string parameters that are further described in Query:

- `_pagedResultsCookie=`*`string`*
- `_pagedResultsOffset=`*`integer`*
- `_pageSize=`*`integer`*

The following example demonstrates how paged results are used:

```
# Request five results per page, and retrieve the first page:
$ curl \
 --user admin:password \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users?
_queryFilter=true&_fields=userName&_pageSize=5&_prettyPrint=true"
```

```
{
  "result" : [ {
    "_id" : "abarnes",
    "_rev" : "<revision>",
    "userName" : "abarnes@example.com"
  }, {
    "_id" : "abergin",
    "_rev" : "<revision>",
    "userName" : "abergin@example.com"
  }, {
    "_id" : "achassin",
    "_rev" : "<revision>",
    "userName" : "achassin@example.com"
  }, {
    "_id" : "ahall",
    "_rev" : "<revision>",
    "userName" : "ahall@example.com"
  }, {
    "_id" : "ahel",
    "_rev" : "<revision>",
    "userName" : "ahel@example.com"
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : "<cookie>",
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}

$ export COOKIE=$(cut -d \" -f 4 <(grep pagedResultsCookie \
 <(curl --cacert ca-cert.pem \
 --user admin:password \
 --silent \
 "https://localhost:8443/api/users?
_queryFilter=true&_fields=userName&_pageSize=5&_prettyPrint=true")
))

# Provide the cookie to request the next five results:
$ curl \
 --user admin:password \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users?
_queryFilter=true&_fields=userName&_pageSize=5\
&_pagedResultsCookie=$COOKIE&_prettyPrint=true"
```

```
{
  "result" : [ {
    "_id" : "ahunter",
    "_rev" : "<revision>",
    "userName" : "ahunter@example.com"
  }, {
    "_id" : "ajensen",
    "_rev" : "<revision>",
    "userName" : "ajensen@example.com"
  }, {
    "_id" : "aknutson",
    "_rev" : "<revision>",
    "userName" : "aknutson@example.com"
  }, {
    "_id" : "alangdon",
    "_rev" : "<revision>",
    "userName" : "alangdon@example.com"
  }, {
    "_id" : "alutz",
    "_rev" : "<revision>",
    "userName" : "alutz@example.com"
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : "<new-cookie>",
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}

# Request the tenth page of five results:
$ curl \
 --user admin:password \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users?
_queryFilter=true&_fields=userName\
&_pageSize=5&_pagedResultsOffset=10&_prettyPrint=true"

{
  "result" : [ {
    "_id" : "ashelton",
    "_rev" : "<revision>",
    "userName" : "ashelton@example.com"
  }, {
```

```
    "_id" : "awalker",
    "_rev" : "<revision>",
    "userName" : "awalker@example.com"
  }, {
    "_id" : "awhite",
    "_rev" : "<revision>",
    "userName" : "awhite@example.com"
  }, {
    "_id" : "aworrell",
    "_rev" : "<revision>",
    "userName" : "aworrell@example.com"
  }, {
    "_id" : "bfrancis",
    "_rev" : "<revision>",
    "userName" : "bfrancis@example.com"
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : "<cookie>",
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

Notice the following features of the responses:

- `"remainingPagedResults" : -1` means that the number of remaining results is unknown.
- `"totalPagedResults" : -1` means that the total number of paged results is unknown.
- `"totalPagedResultsPolicy" : "NONE"` means that result counting is disabled.

## Server-Side Sort

You can use the `_sortKeys` parameter, described in Query, to request that the server sort the results it returns.

The following example sorts results by given name:

```
$ curl \
 --user admin:password \
 --cacert ca-cert.pem \
 --silent \
 "https://localhost:8443/api/users?
_queryFilter=name/familyName+eq+'jensen'\
```

```
&_sortKeys=name/givenName&_fields=name&_prettyPrint=true"

{
  "result" : [ {
    "_id" : "ajensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Allison",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "bjensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Barbara",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "bjense2",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Bjorn",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "gjensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Gern",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "jjensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Jody",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "kjensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Kurt",
      "familyName" : "Jensen"
    }
```

```
  }, {
    "_id" : "user.5814",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Mollie",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "user.19233",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Molly",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "user.32652",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Mommy",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "user.46071",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Mona",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "user.59490",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Monah",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "user.72909",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Monica",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "user.86328",
    "_rev" : "<revision>",
```

```
    "name" : {
      "givenName" : "Moniek",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "user.99747",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Monika",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "rjense2",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Randy",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "rjensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Richard",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "tjensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Ted",
      "familyName" : "Jensen"
    }
  } ],
  "resultCount" : 17,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```


To sort in reverse order, use `_sortKeys=-`*`field`*.

To specify multiple sort keys, use a comma-separated list of fields.

The sort key fields that you specify must exist in the result entries.

The server must store and then sort the result set for your search. If you expect a large result set for your search, use paged results, described in Paged Results, to limit the impact on the server and get your results more quickly.

# Binary Resources

NOTE

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see Learn About the Evaluation Setup Profile.

## Map a Binary Resource

1. Edit the attributes section for a resource in the configuration file `/path/to/opendj/config/rest2ldap/endpoints/api/example-v1.json`.

   The following JSON excerpt maps the user `photo` property to the `jpegPhoto` LDAP attribute:

   ```
   "photo" : { "type" : "simple", "ldapAttribute" :
   "jpegPhoto", "isBinary" : true },
   ```

2. Force the Rest2ldap endpoint to reread the updated configuration file:

   ```
   $ dsconfig \
    set-http-endpoint-prop \
    --hostname localhost \
    --port 4444 \
    --bindDN uid=admin \
    --bindPassword password \
    --endpoint-name "/api" \
    --set enabled:false \
    --usePkcs12TrustStore /path/to/opendj/config/keystore \
    --trustStorePassword:file
   /path/to/opendj/config/keystore.pin \
    --no-prompt

   $ dsconfig \
    set-http-endpoint-prop \
    --hostname localhost \
    --port 4444 \
   ```

```
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

## Update a Binary Resource

1. Ensure that the application has a resource to upload.

   For example, copy a JPEG photo `picture.jpg` to the current directory.

2. Upload the binary resource as a base64-encoded JSON string.

   The following example patches Babs Jensen's resource to add a profile photo:

```
$ base64 encode --rawDataFile picture.jpg

/9j/4AAQSkZJRgABAQEAYABgAAD/4QAWRXhpZgAASUkqAAgAAAAAAAAAA
D/2wBDAAEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEB
AQEBAQEBAQEBAQEBAQEBAQEBAQH/2wBDAQEBAQEBAQEBAQEBAQ
EBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEB
AQEBAQEBAQH/wAARCAABAAEDASIAAhEBAxEB/8QAFQABAQAAAAAAAAAAAA
AAAAAAAr/xAAUEAEAAAAAAAAAAAAAAAAAAA/8QAFAEBAAAAAAAAAAAA
AAAAAAAAP/EABQRAQAAAAAAAAAAAAAAAAAD/2gAMAwEAAhEDEQA/AL
+AAf/Z

$ curl \
 --request PATCH \
 --cacert ca-cert.pem \
 --header "Content-Type: application/json" \
 --data "[{\"operation\": \"add\", \"field\": \"/photo\",
\"value\":
\"/9j/4AAQSkZJRgABAQEAYABgAAD/4QAWRXhpZgAASUkqAAgAAAAAAAA
AAD/2wBDAAEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQ
EBAQEBAQEBAQEBAQEBAQEBAQEBAQH/2wBDAQEBAQEBAQEBAQEBAQ
AQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQ
EBAQEBAQEBAQH/wAARCAABAAEDASIAAhEBAxEB/8QAFQABAQAAAAAAAAA
AAAAAAAAAr/xAAUEAEAAAAAAAAAAAAAAAAAAA/8QAFAEBAAAAAAAAAA
AAAAAAAAAP/EABQRAQAAAAAAAAAAAAAAAAAD/2gAMAwEAAhEDEQA/
AL+AAf/Z\"}]" \
```

```
 --silent \

"https://kvaughan:bribery@localhost:8443/api/users/bjensen
?_prettyPrint=true"

{
  "_id" : "bjensen",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "userName" : "babs@example.com",
  "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
  "name" : {
    "givenName" : "Barbara",
    "familyName" : "Jensen"
  },
  "description" : "Original description",
  "manager" : {
    "_id" : "trigden",
    "_rev" : "<revision>"
  },
  "groups" : [ {
    "_id" : "Carpoolers",
    "_rev" : "<revision>"
  } ],
  "photo" : "<base64-encoded-photo>",
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 9999",
    "emailAddress" : "babs@example.com"
  },
  "uidNumber" : 1076,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/bjensen"
}
```

## Read a Binary Resource

1. Read the binary resource as a base64-encoded JSON string.

   The following example reads Babs Jensen's profile photo:

```
$ curl \
 --cacert ca-cert.pem \
 --silent \

"https://kvaughan:bribery@localhost:8443/api/users/bjensen
?_fields=photo"

{"_id":"bjensen","_rev":"<revision>","photo":"<base64-
photo>"}
```

# Configuration Examples

NOTE

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see Learn About the Evaluation Setup Profile.

## Custom Object

The example REST to LDAP mapping configuration file, `config/rest2ldap/endpoints/api/example-v1.json`, works well with the `ds-evaluation` setup profile. For most deployments, you must customize the mapping to expose additional information.

This example demonstrates how to configure an additional mapping for a custom LDAP object called `affiliateObject`, whose `affiliate` attribute references user entries. This demonstration extends a server set up with the `ds-evaluation` profile. The mappings will also work if you are using the REST to LDAP gateway.

To begin, add the schema for the custom LDAP object, and an example LDAP entry with the object class:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: cn=schema
```

```
changetype: modify
add: attributeTypes
attributeTypes: ( affiliate-oid  NAME 'affiliate' SUP
distinguishedName X-SCHEMA-FILE '99-example-mods.ldif' )
-
add: objectClasses
objectClasses: ( affiliateObject-oid NAME 'affiliateObject' SUP
top STRUCTURAL MUST cn MAY ( description $ affiliate ) X-SCHEMA-
FILE '99-example-mods.ldif' )
EOF

$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
dn: ou=affiliates,dc=example,dc=com
objectClass: top
objectClass: organizationalUnit
ou: affiliates

dn: cn=My Affiliates,ou=affiliates,dc=example,dc=com
objectClass: top
objectClass: affiliateObject
cn: My Affiliates
affiliate: uid=bjensen,ou=People,dc=example,dc=com
affiliate: uid=kvaughan,ou=People,dc=example,dc=com
EOF
```

For details, see LDAP Schema and Add.

In the REST to LDAP mapping configuration file, define an `affiliates` collection subresource, and an `examples:affiliates:1.0` resource type that describes the objects in the `affiliates` collection. This causes the REST API to add an `/api/affiliates` path next to `/api/users` and `/api/groups`. Under `/api/affiliates`, REST clients access the affiliate JSON resources.

The following definition for the `affiliates` collection subresource belongs in the set of `subResources` in the mapping configuration file:

```
// This sub-resource definition maps JSON resources under
/api/affiliates
```

```
// to LDAP entries under ou=affiliates,dc=example,dc=com.
"affiliates" : {
    "type": "collection",
    "dnTemplate": "ou=affiliates,dc=example,dc=com",
    "resource": "examples:affiliates:1.0",
    "namingStrategy": {
        "type": "clientDnNaming",
        "dnAttribute": "cn"
    }
}
```

The parser for the REST to LDAP mapping configuration file is lenient. It lets you include comments in the JSON, although the JSON standard does not allow comments.

The following definition for the `examples:affiliates:1.0` resource type belongs in the set of `example-v1` resource types in the mapping configuration file:

```
// An "affiliate" resource includes property mappings for an
"affiliateObject",
// which is identified by the "cn" attribute. The affiliate DNs
reference person entries.
// Rather than return DNs in JSON resources, a mapper returns
identifiers and display names.
"examples:affiliates:1.0": {
    "superType": "frapi:opendj:rest2ldap:object:1.0",
    "objectClasses": [ "affiliateObject" ],
    "properties": {
        "displayName": {
            "type": "simple",
            "ldapAttribute": "cn",
            "isRequired": true,
            "writability": "createOnly"
        },
        "affiliate": {
            "type": "reference",
            "resourcePath": "../../users",
            "ldapAttribute": "affiliate",
            "isMultiValued": true
        },
        "description": {
            "type": "simple"
        }
    }
}
```

Alternatively, download example-v1.json to replace the existing file.

Replace the directory configuration file, `config/rest2ldap/endpoints/api/example-v1.json`, with the updated mapping configuration file. If you have not already done so, enable HTTP access. For details, see Configure HTTP User APIs. The following example forces the Rest2ldap endpoint to reread its configuration:

```
$ dsconfig \
 set-http-endpoint-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:false \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
 set-http-endpoint-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

With the updated REST to LDAP mapping configuration loaded, REST clients can access affiliates:

```
$ curl \
 --cacert ca-cert.pem \
 --user bjensen:hifalutin \
 --silent \
 "https://localhost:8443/api/affiliates/my%20affiliates?
_fields=affiliate/id&_prettyPrint=true"

{
  "_id" : "My Affiliates",
  "_rev" : "<revision>",
  "affiliate" : [ {
```

```
    "_id" : "bjensen",
    "_rev" : "<revision>"
  }, {
    "_id" : "kvaughan",
    "_rev" : "<revision>"
  } ]
}
```

## Per-Server Password Policies

This example demonstrates how to add a per-server password policy over REST. Per-server password policies are set in the server configuration, and not replicated. You must create them on each replica.

The password policy in this example includes:

- A password history setting to retain the last five password values.
- The same password storage scheme as the default password policy.
- The default random password generator.
- The default length-based password validator.
- The default dictionary password validator, which is available, but not enabled by default.

With the default password policy, a user can change their password to `password`. A password policy with the default dictionary validator would not allow this:

```
$ curl \
 --request POST \
 --cacert ca-cert.pem \
 --user bjensen:hifalutin \
 --header "Content-Type: application/json" \
 --data '{"oldPassword": "hifalutin", "newPassword": "password"}'
\
 --silent \
 "https://localhost:8443/api/users/bjensen?
_action=modifyPassword&dryRun=true&passwordQualityAdvice=true"

{}
```

Update the server configuration to enable the password policy for all Example.com users:

1. Enable the default dictionary password validator:

```
$ curl \
 --request PATCH \
 --user admin:password \
 --data '[{"operation": "replace", "field": "/enabled",
"value": true}]' \
 --cacert ca-cert.pem \
 --header "Content-Type: application/json" \
 --silent \
 "https://localhost:8443/admin/config/password-
validators/Dictionary"
```

2. Add the password policy:

```
$ curl \
 --request POST \
 --user admin:password \
 --data '{
  "_id": "Per-Server Password Policy",
  "_schema": "password-policy",
  "password-attribute": "userPassword",
  "default-password-storage-scheme": [{"_id": "PBKDF2-HMAC-
SHA256"}],
  "password-generator": { "_id": "Random Password Generator"
},
  "password-validator": [{"_id": "Dictionary"}, {"_id":
"Length-Based Password Validator"}],
  "password-history-count": 5
 }' \
 --cacert ca-cert.pem \
 --header "Content-Type: application/json" \
 --silent \
 "https://localhost:8443/admin/config/password-policies/"
```

3. Assign the password policy to users:

   The following command adds a virtual attribute that assigns the password policy to all Example.com users:

```
$ curl \
 --request POST \
 --user admin:password \
 --data '{
  "_id": "Password Policy Virtual Attribute",
  "_schema": "user-defined-virtual-attribute",
  "enabled": true,
```

```
    "base-dn": [ "ou=people,dc=example,dc=com" ],
    "filter": [ "(objectClass=person)" ],
    "attribute-type": "ds-pwp-password-policy-dn",
    "value": [ "cn=Per-Server Password Policy,cn=Password
  Policies,cn=config" ]
   }' \
   --cacert ca-cert.pem \
   --header "Content-Type: application/json" \
   --silent \
   "https://localhost:8443/admin/config/virtual-attributes/"
```

Check that the new policy does not let a user change their password to `password`:

```
$ curl \
 --request POST \
 --cacert ca-cert.pem \
 --user bjensen:hifalutin \
 --header "Content-Type: application/json" \
 --data '{"oldPassword": "hifalutin", "newPassword": "password"}'
\
 --silent \
 "https://localhost:8443/api/users/bjensen?
_action=modifyPassword&dryRun=true&passwordQualityAdvice=true"

{
  "code" : 400,
  "reason" : "Bad Request",
  "message" : "Constraint Violation: The provided new password
failed the validation checks defined in the server: The provided
password contained a word from the server's dictionary",
  "detail" : {
    "passwordQualityAdvice" : {
      "passingCriteria" : [ {
        "type" : "length-based",
        "parameters" : {
          "min-password-length" : 6,
          "max-password-length" : 0
        }
      } ],
      "failingCriteria" : [ {
        "type" : "dictionary",
        "parameters" : {
          "case-sensitive-validation" : false,
          "min-substring-length" : 5,
          "test-reversed-password" : true,
```

```
          "check-substrings" : true
        }
      } ]
    }
  }
}
```

For details on password policy settings, see Per-Server Password Policies.

## Subentry Password Policies

This example demonstrates how to configure REST to LDAP to manage Internet-Draft subentry password policies. This demonstration extends a server set up with the `ds-evaluation` profile. The mappings will also work if you are using the REST to LDAP gateway.

To begin, edit the default REST to LDAP mapping file, `config/rest2ldap/endpoints/api/example-v1.json`.

Define an `/api/subentries` endpoint that exposes LDAP subentries over REST, including password policies, by adding the following under `subResources`:

```
// This subresource definition maps JSON resources under
/api/subentries
// to subentries under dc=example,dc=com.
"subentries" : {
  "type": "collection",
  "dnTemplate": "dc=example,dc=com",
  "resource": "examples:subentry:1.0",
  "namingStrategy": {
    "type": "clientDnNaming",
    "dnAttribute": "cn"
  }
}
```

Define the subentry and password policy under the other `example-v1` resource types:

```
// A subentry resource maps to a subentry object.
"examples:subentry:1.0": {
  "superType": "frapi:opendj:rest2ldap:object:1.0",
  "isAbstract": true,
  "objectClasses": [ "top", "subentry" ],
  "properties": {
    "_id": {
      "type": "simple",
```

```
      "ldapAttribute": "cn",
      "isRequired": true,
      "writability": "createOnly"
    },
    "subtreeSpecification": {
      "type": "simple"
    }
  }
},
// A passwordPolicy resource maps to a subentry password policy
object.
// This mapping uses the LDAP attribute names,
// except for passwordValidator which maps to the validator in the
configuration.
"examples:passwordPolicy:1.0": {
  "superType": "examples:subentry:1.0",
  "objectClasses": [ "pwdPolicy", "pwdValidatorPolicy", "subentry"
],
  "properties": {
    "pwdAttribute": {
      "type": "simple",
      "isRequired": true
    },
    "pwdAllowUserChange": {
      "type": "simple"
    },
    "pwdCheckQuality": {
      "type": "simple"
    },
    "pwdExpireWarning": {
      "type": "simple"
    },
    "pwdFailureCountInterval": {
      "type": "simple"
    },
    "pwdGraceAuthNLimit": {
      "type": "simple"
    },
    "pwdInHistory": {
      "type": "simple"
    },
    "pwdLockout": {
      "type": "simple"
    },
    "pwdLockoutDuration": {
```

```
      "type": "simple"
    },
    "pwdMaxAge": {
      "type": "simple"
    },
    "pwdMaxFailure": {
      "type": "simple"
    },
    "pwdMinAge": {
      "type": "simple"
    },
    "pwdMinLength": {
      "type": "simple"
    },
    "pwdMustChange": {
      "type": "simple"
    },
    "pwdSafeModify": {
      "type": "simple"
    },
    "passwordValidator" : {
      "type": "reference",
      "baseDn": "cn=Password Validators,cn=config",
      "ldapAttribute": "ds-cfg-password-validator",
      "primaryKey": "cn",
      "isMultiValued": true,
      "mapper": {
        "type": "object",
        "properties": {
          "_id": {
            "type": "simple",
            "ldapAttribute": "cn",
            "isRequired": true
          }
        }
      }
    }
  }
```

In LDAP, you can edit user entries to assign password policies. For REST to LDAP, add corresponding password policy properties to the `frapi:opendj:rest2ldap:user:1.0` resource type:

```
// Administrators can read the current password policy.
"pwdPolicy": {
```

```
  "type": "reference",
  "baseDn": "..,..",
  "ldapAttribute": "pwdPolicySubentry",
  "primaryKey": "cn",
  "searchFilter": "(&(objectclass=subentry)
(objectclass=pwdPolicy))",
  "mapper": {
    "type": "object",
    "properties": {
      "_id": {
        "type": "simple",
        "ldapAttribute": "cn",
        "writability": "readOnlyDiscardWrites"
      }
    }
  }
},
// Administrators can set a new password policy.
"newPwdPolicy": {
  "type": "reference",
  "baseDn": "..,..",
  "ldapAttribute": "ds-pwp-password-policy-dn",
  "primaryKey": "cn",
  "searchFilter": "(&(objectclass=subentry)
(objectclass=pwdPolicy))",
  "mapper": {
    "type": "object",
    "properties": {
      "_id": {
        "type": "simple",
        "ldapAttribute": "cn",
        "isRequired": true
      }
    }
  }
},
```

Alternatively, download subentries.json to replace the existing `example-v1.json` file.

Replace the directory configuration file, `config/rest2ldap/endpoints/api/example-v1.json`, with the updated mapping configuration file. If you have not already done so, enable HTTP access. For details, see Configure HTTP User APIs. The following example forces the Rest2ldap endpoint to reread its configuration:

```
$ dsconfig \
 set-http-endpoint-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:false \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
 set-http-endpoint-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

With the augmented mapping in place, grant access for password administrators to edit and assign password policies:

- Grant the necessary privileges to edit subentry password policies:
  - The `subentry-write` privilege lets password administrators edit subentries.
  - The `config-read` privilege lets password administrators reference password validator entries stored in the server configuration.
- Grant access to assign password policies, letting password administrators read users' `pwdPolicySubentry` attributes and write users' `ds-pwp-password-policy-dn` attributes.
- Grant access to assign password policies, letting password administrators write the subentry attributes, `ds-pwp-password-validator` and `subtreeSpecification`.

For this demonstration, grant access to the administrator user Kirsten Vaughan:

```
# Assign privileges:
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
```

```
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password  << EOF
dn: uid=kvaughan,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: config-read
EOF

$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: uid=kvaughan,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: subentry-write
EOF

# Grant access to subentry policies
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "pwdPolicySubentry||ds-pwp-password-policy-
dn||ds-pwp-password-validator||subtreeSpecification")
 (version 3.0;acl "Allow Administrators to manage users' password
policies";
 allow (all) (groupdn = "ldap:///cn=Directory
Administrators,ou=Groups,dc=example,dc=com");)
EOF
```

```
# Grant access to configuration-based policy elements:
$ dsconfig \
 set-access-control-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --add "global-aci:(target=\"ldap:///cn=config\")
(targetattr=\"*||+\")\
(version 3.0; acl \"Config read for Kirsten Vaughan\"; allow
(read,search,compare)\
userdn=\"ldap:///uid=kvaughan,ou=People,dc=example,dc=com\";)" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Create and assign a subentry password policy:

```
$ curl \
 --request PUT \
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --header "If-None-Match: *" \
 --data '{
  "_id": "Subentry Password Policy with Validators",
  "_schema": "examples:passwordPolicy:1.0",
  "pwdAttribute": "userPassword",
  "pwdAllowUserChange": true,
  "pwdFailureCountInterval": 300,
  "pwdLockout": true,
  "pwdLockoutDuration": 300,
  "pwdMaxFailure": 3,
  "pwdSafeModify": true,
  "passwordValidator": [{"_id": "Character Set"}, {"_id": "Length-
Based Password Validator"}],
  "subtreeSpecification": "{base \"ou=people\",
specificationFilter \"(isMemberOf=cn=Directory
Administrators,ou=Groups,dc=example,dc=com)\" }"
 }' \
 --silent \

https://localhost:8443/api/subentries/Subentry%20Password%20Policy
%20with%20Validators?_prettyPrint=true
```

Observe that the password administrator can view which password policy applies:

```
$ curl \
 --cacert ca-cert.pem \
 --user kvaughan:bribery \
 --silent \
 "https://localhost:8443/api/users/kvaughan?
_fields=pwdPolicy&_prettyPrint=true"

{
  "_id" : "kvaughan",
  "_rev" : "<revision>",
  "pwdPolicy" : {
    "_id" : "Subentry Password Policy with Validators"
  }
}
```

Observe that the field is not visible to regular users:

```
$ curl \
 --cacert ca-cert.pem \
 --user bjensen:hifalutin \
 --silent \
 "https://localhost:8443/api/users/bjensen?
_fields=pwdPolicy&_prettyPrint=true"

{
  "_id" : "bjensen",
  "_rev" : "<revision>"
}
```

## Map LDAP Entries

This example demonstrates how to map an additional existing resource at the root of the REST API.

For example, the `ds-evaluation` setup profile includes localities:

```
dn: l=Bristol,ou=Locations,dc=example,dc=com
objectClass: top
objectClass: locality
l: Bristol
street: Broad Quay House, Prince Street
```

```
dn: l=Montbonnot,ou=Locations,dc=example,dc=com
objectClass: top
l: Montbonnot
street: 55 Rue Blaise Pascal

dn: l=Lysaker,ou=Locations,dc=example,dc=com
objectClass: top
objectClass: locality
l: Lysaker
street: Lysaker Torg 2

dn: l=San Francisco,ou=Locations,dc=example,dc=com
objectClass: top
objectClass: locality
l: San Francisco
street: 201 Mission Street Suite 2900

dn: l=Vancouver,ou=Locations,dc=example,dc=com
objectClass: top
objectClass: locality
l: Vancouver
street: 201 Northeast Park Plaza Drive
```

This demonstration adds an `/api/locations` next to `/api/users` and `/api/groups`.

Edit your copy of the default REST API mapping configuration. Add a location subresource at the top level next to users and groups subresource definitions:

```
"locations": {
    "type": "collection",
    "dnTemplate": "ou=locations,dc=example,dc=com",
    "resource": "frapi:opendj:rest2ldap:location:1.0",
    "namingStrategy": {
        "type": "clientDnNaming",
        "dnAttribute": "l"
    }
}
```

Add a location schema after the other schema definitions:

```
"frapi:opendj:rest2ldap:location:1.0": {
    "superType": "frapi:opendj:rest2ldap:object:1.0",
    "objectClasses": ["locality"],
    "properties": {
        "_id": {
```

```
            "type": "simple",
            "ldapAttribute": "l",
            "isRequired": true,
            "writability": "createOnly"
        },
        "displayName": {
            "type": "simple",
            "ldapAttribute": "l",
            "isRequired": true,
            "writability": "readOnly"
        },
        "description": {
            "type": "simple"
        },
        "state": {
            "type": "simple",
            "ldapAttribute": "st"
        },
        "street": {
            "type": "simple"
        }
    }
}
```

Alternatively, download example-locations.json to replace the existing `example-v1.json` file.

Replace the directory configuration file, `config/rest2ldap/endpoints/api/example-v1.json` , with the updated mapping configuration file. If you have not already done so, enable HTTP access. For details, see Configure HTTP User APIs. The following example forces the Rest2ldap endpoint to reread its configuration:

```
$ dsconfig \
 set-http-endpoint-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:false \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
```

```
 set-http-endpoint-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

With the updated configuration loaded, view the results:

```
# List locations:
$ curl \
 --cacert ca-cert.pem \
 --user bjensen:hifalutin \
 --silent \
 "https://localhost:8443/api/locations/?
_queryFilter=true&_prettyPrint=true"

{
  "result" : [ {
    "_id" : "Bristol",
    "_rev" : "<revision>",
    "_schema" : "frapi:opendj:rest2ldap:location:1.0",
    "displayName" : "Bristol",
    "street" : "Broad Quay House, Prince Street"
  }, {
    "_id" : "Montbonnot",
    "_rev" : "<revision>",
    "_schema" : "frapi:opendj:rest2ldap:location:1.0",
    "displayName" : "Montbonnot",
    "street" : "55 Rue Blaise Pascal"
  }, {
    "_id" : "Lysaker",
    "_rev" : "<revision>",
    "_schema" : "frapi:opendj:rest2ldap:location:1.0",
    "displayName" : "Lysaker",
    "street" : "Lysaker Torg 2"
  }, {
    "_id" : "San Francisco",
    "_rev" : "<revision>",
    "_schema" : "frapi:opendj:rest2ldap:location:1.0",
    "displayName" : "San Francisco",
```

```
    "street" : "201 Mission Street Suite 2900"
  }, {
    "_id" : "Vancouver",
    "_rev" : "<revision>",
    "_schema" : "frapi:opendj:rest2ldap:location:1.0",
    "displayName" : "Vancouver",
    "street" : "201 Northeast Park Plaza Drive"
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}

# Read a single location:
$ curl \
 --cacert ca-cert.pem \
 --user bjensen:hifalutin \
 --silent \
 "https://localhost:8443/api/locations/montbonnot?
_prettyPrint=true"

{
  "_id" : "Montbonnot",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:location:1.0",
  "displayName" : "Montbonnot",
  "street" : "55 Rue Blaise Pascal"
}
```

## Nested Resources

This example demonstrates how to add device resources under user resources. Use this pattern when you have LDAP child entries under parent entries.

For example, the `ds-evaluation` setup profile includes a user with two subordinate devices:

```
dn: uid=nbohr,ou=People,dc=example,dc=com
objectClass: person
objectClass: cos
objectClass: inetOrgPerson
objectClass: organizationalPerson
```

```
objectClass: posixAccount
objectClass: top
uid: nbohr
classOfService: gold
userpassword: password
facsimileTelephoneNumber: +1 408 555 1213
givenName: Niels
cn: Niels Bohr
telephoneNumber: +1 408 555 1212
sn: Bohr
roomNumber: 0007
homeDirectory: /home/nbohr
mail: nbohr@example.com
l: Vancouver
ou: People
uidNumber: 1111
gidNumber: 1000
description: Quantum device example

dn: cn=quantum dot,uid=nbohr,ou=People,dc=example,dc=com
objectClass: device
objectClass: top
cn: quantum dot
serialNumber: WI-3005
owner: uid=nbohr,ou=People,dc=example,dc=com

dn: cn=qubit generator,uid=nbohr,ou=People,dc=example,dc=com
objectClass: device
objectClass: top
cn: qubit generator
serialNumber: XF551426
owner: uid=nbohr,ou=People,dc=example,dc=com
```

In your REST API, the users collection contains zero or more user resources. Each user resource can have a devices collection that contains zero or more device resources. The path to a device is `/users/`*`uid`*`/devices/`*`cn`*, where *uid* is the user UID, and *cn* is the device CN. For example, `/users/nbohr/devices/quantum dot`.

Edit your copy of the default REST API mapping configuration. Add the subresources definition to the `frapi:opendj:rest2ldap:user:1.0` schema:

```
"subResources": {
    "devices": {
        "type": "collection",
        "resource": "frapi:opendj:rest2ldap:device:1.0",
```

```
        "namingStrategy": {
            "type": "clientDnNaming",
            "dnAttribute": "cn"
        }
    }
}
```

Add a device schema after the other schema definitions:

```
"frapi:opendj:rest2ldap:device:1.0": {
    "superType": "frapi:opendj:rest2ldap:object:1.0",
    "objectClasses": ["device"],
    "properties": {
        "_id": {
            "type": "simple",
            "ldapAttribute": "cn",
            "isRequired": true,
            "writability": "createOnly"
        },
        "displayName": {
            "type": "simple",
            "ldapAttribute": "cn",
            "isRequired": true,
            "writability": "readOnly"
        },
        "description": {
            "type": "simple"
        },
        "owner": {
            "type": "reference",
            "resourcePath": "../..",
            "ldapAttribute": "owner"
        },
        "serialNumber": {
            "type": "simple"
        }
    }
}
```

Alternatively, download example-subresources.json to replace the default `example-v1.json` file.

Replace the directory configuration file, `config/rest2ldap/endpoints/api/example-v1.json`, with the updated mapping configuration file. If you have not already done so,

enable HTTP access. For details, see Configure HTTP User APIs. The following example forces the Rest2ldap endpoint to reread its configuration:

```
$ dsconfig \
 set-http-endpoint-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:false \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
 set-http-endpoint-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

With the updated configuration loaded, view the results:

```
# Read the user resource:
$ curl \
 --cacert ca-cert.pem \
 --user nbohr:password \
 --silent \
 "https://localhost:8443/api/users/nbohr?_prettyPrint=true"

{
  "_id" : "nbohr",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "userName" : "nbohr@example.com",
  "displayName" : [ "Niels Bohr" ],
  "name" : {
    "givenName" : "Niels",
```

```
    "familyName" : "Bohr"
  },
  "description" : "Quantum device example",
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "nbohr@example.com"
  },
  "uidNumber" : 1111,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/nbohr"
}

# List devices:
$ curl \
 --cacert ca-cert.pem \
 --user nbohr:password \
 --silent \
 "https://localhost:8443/api/users/nbohr/devices/?
_queryFilter=true&_prettyPrint=true"

{
  "result" : [ {
    "_id" : "quantum dot",
    "_rev" : "<revision>",
    "_schema" : "frapi:opendj:rest2ldap:device:1.0",
    "displayName" : "quantum dot",
    "owner" : {
      "_id" : "nbohr",
      "_rev" : "<revision>"
    },
    "serialNumber" : "WI-3005"
  }, {
    "_id" : "qubit generator",
    "_rev" : "<revision>",
    "_schema" : "frapi:opendj:rest2ldap:device:1.0",
    "displayName" : "qubit generator",
    "owner" : {
      "_id" : "nbohr",
      "_rev" : "<revision>"
    },
    "serialNumber" : "XF551426"
  } ],
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
```

```
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}

# Read a device resource:
$ curl \
 --cacert ca-cert.pem \
 --user nbohr:password \
 --silent \
 "https://localhost:8443/api/users/nbohr/devices/quantum%20dot?
_prettyPrint=true"

{
  "_id" : "quantum dot",
  "_rev" : "000000007ba330d8",
  "_schema" : "frapi:opendj:rest2ldap:device:1.0",
  "displayName" : "quantum dot",
  "owner" : {
    "_id" : "nbohr",
    "_rev" : "<revision>"
  },
  "serialNumber" : "WI-3005"
}m
```

## JSON to LDAP

This example demonstrates simple user profiles, starting from JSON and working back to LDAP. The following JSON is a example profile:

```
{
    "externalId": "newuser",
    "userName": "newuser",
    "name": {
        "formatted": "New User",
        "givenName": "User",
        "familyName": "New"
    },
    "phoneNumbers": [{
        "value": "+1 408 555 1212",
        "type": "work"
    }],
    "emails": [
        {
            "value": "newuser@example.com",
```

```
            "type": "work",
            "primary": true
        },
        {
            "value": "newuser@example.org",
            "type": "personal",
            "primary": false
        }
    ]
}
```

Notice these key features of the example profile:

- Unlike Common REST JSON objects, this profile has no `"_id"` field.
- The `"phoneNumbers"` and `"emails"` fields are multi-valued, similar to the LDAP `telephoneNumber` and `mail` attributes.

  However, both JSON fields hold arrays of nested objects. The similar LDAP attributes hold simple values. The other fields have a straightforward mapping to standard LDAP attributes, but these fields do not.

  The `"type"` field in a phone number or email is an arbitrary label assigned by the user.

▼ Define the LDAP schema

This example assumes that the user profile does not exist in LDAP yet. The aim is therefore to translate JSON objects into LDAP entries.

LDAP attributes generally hold values with tightly defined syntaxes, not arbitrary objects. JSON syntax attributes are an exception. The example profile holds a mix of fields that map directly to LDAP attributes, and fields that do not.

When determining them LDAP attributes to use, the first step is to look for natural matches with attributes defined in the default schema. Based on the *About This Reference*, you can derive the following table of correspondences:

| Profile JSON Field | LDAP Attribute |
|---|---|
| `"externalId"` | `uid` |
| `"userName"` | `uid` |
| `"name/formatted"` | `cn` |
| `"name/givenName"` | `givenName` |
| `"name/familyName"` | `sn` |

| Profile JSON Field | LDAP Attribute |
| --- | --- |
| `"emails"` | No direct match. |
| `"phoneNumbers"` | No direct match. |

For the `"emails"` and `"phoneNumbers"` fields, there is no natural match. The `uddiEMail` and `uddiPhone` attributes come the closest because they optionally include a user-defined type. However, avoid those attributes for the following reasons:

- These user profiles are unrelated to Universal Description, Discovery, and Integration (UDDI), which is focused instead on storing data for a SOAP-based web services discovery.
- All client applications would have to know how to consume the UDDI-specific format. REST to LDAP has no means to transform *`type#value`* into `{"type": "`*`type`*`", "value": "`*`value`*`"}`.
- The `uddiEmail` has no provision for the `"primary"` boolean value.

It would also be possible, but not advisable, to store `"emails"` and `"phoneNumbers"` in separate LDAP entries. The LDAP user entry could include attributes that reference email address and phone number entries by their DNs. The email and phone number entries could in turn reference users with the standard `owner` attribute. Unfortunately, there is no real value in storing email addresses and phone numbers separately from a user's entry. The email and phone settings will not be shared with other entries, but instead belong only to the profile. Client applications will expect to update them atomically with the user profile. A profile change should not require updating an arbitrary number of LDAP entries. Even if they could be conveniently configured as single updates in the REST API, it would mean that updates to a JSON resource could partially fail in LDAP, a source of potential confusion.

An apt choice for the `"emails"` and `"phoneNumbers"` fields is therefore JSON syntax attributes. This example defines the following LDAP schema definitions for appropriate JSON attributes:

```
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( example-email-oid
  NAME 'email'
  DESC 'An email address with an optional user-defined type and
primary boolean'
  EQUALITY caseIgnoreJsonQueryMatch
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1
  USAGE userApplications
  X-SCHEMA-FILE '99-example.ldif'
```

```
  X-ORIGIN 'DS Documentation Examples' )
-
add: attributeTypes
attributeTypes: ( example-phone-number-oid
  NAME 'phoneNumber'
  DESC 'A phone number with an optional user-defined type'
  EQUALITY caseIgnoreJsonQueryMatch
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1
  USAGE userApplications
  X-SCHEMA-FILE '99-example.ldif'
  X-ORIGIN 'DS Documentation Examples' )
-
add: objectClasses
objectClasses: ( example-profile-user-oid
  NAME 'profileUser'
  DESC 'A user with optional profile components having user-
defined types'
  SUP top
  AUXILIARY
  MAY ( email $ phoneNumber )
  X-SCHEMA-FILE '99-example.ldif'
  X-ORIGIN 'DS Documentation Examples' )
```

An auxiliary object class lets an entry that already has a structural object class hold additional attributes. For background information about LDAP schema, read LDAP Schema.

The following corresponding LDIF uses these definitions to define Example.com data with one user profile. Babs Jensen is the administrator and sole initial user. She can create other profiles and reset passwords:

```
dn: dc=example,dc=com
objectClass: domain
objectClass: top
dc: example
aci: (target ="ldap:///dc=example,dc=com")
 (targetattr != "userPassword")
 (version 3.0;acl "Authenticated users have read-search access";
 allow (read, search, compare)(userdn = "ldap:///all");)
aci: (target ="ldap:///dc=example,dc=com")
 (targetattr = "*")
 (version 3.0;acl "Allow Babs to administer other entries";
 allow (all)(userdn =
"ldap:///uid=bjensen,ou=People,dc=example,dc=com");)
```

```
dn: ou=People,dc=example,dc=com
objectClass: organizationalunit
objectClass: top
ou: People
aci: (target ="ldap:///ou=People,dc=example,dc=com")
 (targetattr = "*")
 (version 3.0; acl "Allow self management of profiles";
 allow (delete, write)(userdn = "ldap:///self");)

dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: profileUser
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: top
uid: bjensen
ou: People
cn: Barbara Jensen
cn: Babs Jensen
givenname: Barbara
sn: Jensen
userpassword: hifalutin
email: {"value": "bjensen@example.com", "type": "work",
"primary": true}
phoneNumber: {"value": "+1 408 555 1862", "type": "work"}
ds-privilege-name: password-reset
```

▼ Index the JSON attributes

Client applications might look up user profiles based on email addresses or phone numbers. They are not likely to look up user profiles based on user-defined labels for emails and phone numbers. Therefore, instead of indexing all fields of each JSON attribute value, index only the values:

1. Install a directory server with the `ds-evaluation` setup profile.
2. Configure a custom schema provider to allow the server to index only the JSON `"values"`:

```
$ dsconfig \
 create-schema-provider \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
```

```
 --provider-name "Value JSON Query Matching Rule" \
 --type json-query-equality-matching-rule \
 --set enabled:true \
 --set case-sensitive-strings:false \
 --set ignore-white-space:true \
 --set matching-rule-name:caseIgnoreJsonQueryMatch \
 --set matching-rule-oid:1.3.6.1.4.1.36733.2.1.4.1 \
 --set indexed-field:value \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

3. Update the LDAP schema:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( example-email-oid
  NAME 'email'
  DESC 'An email address with an optional user-defined
type and primary boolean'
  EQUALITY caseIgnoreJsonQueryMatch
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1
  USAGE userApplications
  X-SCHEMA-FILE '99-example.ldif'
  X-ORIGIN 'DS Documentation Examples' )
-
add: attributeTypes
attributeTypes: ( example-phone-number-oid
  NAME 'phoneNumber'
  DESC 'A phone number with an optional user-defined
type'
  EQUALITY caseIgnoreJsonQueryMatch
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1
  USAGE userApplications
```

```
  X-SCHEMA-FILE '99-example.ldif'
  X-ORIGIN 'DS Documentation Examples' )
-
add: objectClasses
objectClasses: ( example-profile-user-oid
  NAME 'profileUser'
  DESC 'A user with optional profile components having
user-defined types'
  SUP top
  AUXILIARY
  MAY ( email $ phoneNumber )
  X-SCHEMA-FILE '99-example.ldif'
  X-ORIGIN 'DS Documentation Examples' )
EOF
```

4. Add indexes for the `email` and `phoneNumber` JSON attributes:

```
$ dsconfig \
 create-backend-index \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name email \
 --set index-type:equality \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
 create-backend-index \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name phoneNumber \
 --set index-type:equality \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

5. Import the LDAP data with Babs's profile:

```
$ import-ldif \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backendID dsEvaluation \
 --ldifFile /path/to/opendjprofiles.ldif \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin
```

6. To check your work, read Babs's profile over LDAP:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --baseDn dc=example,dc=com \
 "(uid=bjensen)"

dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: profileUser
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: top
cn: Barbara Jensen
cn: Babs Jensen
email: {"value": "bjensen@example.com", "type": "work",
"primary": true}
givenName: Barbara
ou: People
phoneNumber: {"value": "+1 408 555 1862", "type": "work"}
sn: Jensen
uid: bjensen
```

Now that you have sample data in LDAP, create the REST to LDAP mapping.

▼ Configure the REST to LDAP mapping

The REST to LDAP mapping defines the HTTP API and JSON resources that are backed by the LDAP service and entries.

Rather than patch the default example mapping, this example adds a separate API with a new mapping for the profile view.

The new mapping uses the following features:

- A `"serverNaming"` naming strategy ensures proper handing of Common REST `"_id"` values. LDAP entries use `uid` as the RDN. JSON profiles use server-generated `"_id"` values.

  The JSON profiles have `"externalId"` values corresponding to the LDAP `uid`. However, the JSON profiles do not define `"_id"` values. Client applications will use the `"_id"` values when uniquely identifying a profile, but not when looking up a profile.

  The `"_id"` maps to the `entryUUID` LDAP operational attribute. The `entryUUID` attribute is generated on creation and managed by the LDAP server.

- As in the default example, passwords are not visible in profiles.

  Client applications use dedicated REST actions to manage password reset by administrators and to manage password modifications by users. The REST actions for password management require HTTPS.

- The `"externalId"` and `"userName"` fields both map to LDAP `uid` attributes.

  You could make it possible to update these fields after creation. This example assumes that these fields might be expected not to change, and so restricts client applications from changing them.

- The `"formatted"` and `"familyName"` fields of the name map to attributes that are required by the LDAP object class, `inetOrgPerson`. They are therefore required in the JSON profile as well.

The full mapping file is shown in the following listing:

```
{
    "version": "1.0",
    "resourceTypes": {
        "example-mapping": {
            "subResources": {
                "users": {
                    "type": "collection",
                    "dnTemplate": "ou=people,dc=example,dc=com",
                    "resource": "examples:user:1.0",
                    "namingStrategy": {
                        "type": "serverNaming",
```

```
                "dnAttribute": "uid",
                "idAttribute": "entryUUID"
            }
        }
    },
    "frapi:opendj:rest2ldap:object:1.0": {
        "isAbstract": true,
        "objectClasses": [
            "top"
        ],
        "properties": {
            "_schema": {
                "type": "resourceType"
            },
            "_rev": {
                "type": "simple",
                "ldapAttribute": "etag",
                "writability": "readOnly"
            },
            "_meta": {
                "type": "object",
                "properties": {
                    "created": {
                        "type": "simple",
                        "ldapAttribute": "createTimestamp",
                        "writability": "readOnly"
                    },
                    "lastModified": {
                        "type": "simple",
                        "ldapAttribute": "modifyTimestamp",
                        "writability": "readOnly"
                    }
                }
            }
        }
    },
    "examples:user:1.0": {
        "superType": "frapi:opendj:rest2ldap:object:1.0",
        "objectClasses": [
            "profileUser",
            "person",
            "organizationalPerson",
            "inetOrgPerson"
        ],
```

```
    "supportedActions": [
        "modifyPassword",
        "resetPassword"
    ],
    "properties": {
        "externalId": {
            "type": "simple",
            "ldapAttribute": "uid",
            "isRequired": true,
            "writability": "createOnlyDiscardWrites"
        },
        "userName": {
            "type": "simple",
            "ldapAttribute": "uid",
            "isRequired": true,
            "writability": "createOnlyDiscardWrites"
        },
        "name": {
            "type": "object",
            "properties": {
                "formatted": {
                    "type": "simple",
                    "ldapAttribute": "cn",
                    "isRequired": true
                },
                "givenName": {
                    "type": "simple"
                },
                "familyName": {
                    "type": "simple",
                    "ldapAttribute": "sn",
                    "isRequired": true
                }
            }
        },
        "emails": {
            "type": "json",
            "ldapAttribute": "email",
            "isMultiValued": true
        },
        "phoneNumbers": {
            "type": "json",
            "ldapAttribute": "phoneNumber",
            "isMultiValued": true
        }
```

```
            }
        }
    }
}
```

1. Download the mapping file, example-mapping.json.
2. Copy the mapping file to the appropriate server REST to LDAP configuration directory:

```
$ mkdir /path/to/opendj/config/rest2ldap/endpoints/rest

$ cp example-mapping.json
/path/to/opendj/config/rest2ldap/endpoints/rest/
```

   The default example defines a REST API under `/api`. This example uses `/rest` instead.

3. Enable a Rest2ldap endpoint for the API:

```
$ dsconfig \
 create-http-endpoint \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --set authorization-mechanism:HTTP\ Basic \
 --set config-directory:config/rest2ldap/endpoints/rest \
 --set enabled:true \
 --type rest2ldap-endpoint \
 --endpoint-name /rest \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

▼ Try the new API

***Look Up a Profile by Email Address***

The following query looks for profiles with email addresses containing `bjensen`:

```
$ curl \
 --request GET \
 --cacert ca-cert.pem \
```

```
 --user bjensen:hifalutin \
 --silent \
 "https://localhost:8443/rest/users/?
_queryFilter=emails/value+co+'bjensen'"

{
  "result" : [ {
    "_id" : "<generated-id>",
    "_rev" : "<revision>",
    "_schema" : "examples:user:1.0",
    "externalId" : "bjensen",
    "userName" : "bjensen",
    "name" : {
      "formatted" : [ "Barbara Jensen", "Babs Jensen" ],
      "givenName" : "Barbara",
      "familyName" : "Jensen"
    },
    "emails" : [ {
      "value" : "bjensen@example.com",
      "type" : "work",
      "primary" : true
    } ],
    "phoneNumbers" : [ {
      "value" : "+1 408 555 1862",
      "type" : "work"
    } ]
  } ],
  "resultCount" : 1,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

*Look Up a Profile by Phone Number*

The following query looks for profiles with the phone number `+1 408 555 1862`:

```
$ curl \
 --request GET \
 --cacert ca-cert.pem \
 --user bjensen:hifalutin \
 --silent \
 "https://localhost:8443/rest/users/?
_queryFilter=phoneNumbers/value+eq+'%2B1%20408%20555%201862'"
```

Notice that the telephone number is URL encoded as `%2B1%20408%20555%201862`.

*Create a User Profile*

This example creates a profile using the JSON in newuser.json:

```
$ curl \
 --request POST \
 --cacert ca-cert.pem \
 --user bjensen:hifalutin \
 --header "Content-Type: application/json" \
 --data @newuser.json \
 --silent \
 https://localhost:8443/rest/users/
```

Upon initial creation, the user has no password, and so cannot authenticate, yet.

*Set a Password*

Set a password to allow the user to authenticate.

This example uses Bash shell and jq to manipulate JSON on the command line.

The first operation gets the user profile `"_id"` and holds it in an `ID` environment variable.

The second operation performs an administrative password reset as Babs and holds the resulting generated password in a `PASSWORD` environment variable.

The third and final operation uses the generated password to authenticate as the user and modify the password:

```
$ export ID=$(jq --raw-output '.result[0] ._id' \
 <(curl --silent --request GET --cacert ca-cert.pem --user
bjensen:hifalutin 2>/dev/null \
 "https://localhost:8443/rest/users/?
_queryFilter=externalId+eq+'newuser'"))

$ export PASSWORD=$(jq --raw-output '.generatedPassword' \
 <(curl --silent --request POST --cacert ca-cert.pem --user
bjensen:hifalutin \
 --header "Content-Type: application/json" --data '{}' \
 "https://localhost:8443/rest/users/${ID}?
_action=resetPassword"))

$ curl \
 --request POST \
```

```
 --cacert ca-cert.pem \
 --user "newuser:${PASSWORD}" \
 --header "Content-Type: application/json" \
 --data "{\"oldPassword\": \"${PASSWORD}\", \"newPassword\":
\"password\"}" \
 --silent \
 "https://localhost:8443/rest/users/${ID}?
_action=modifyPassword"
```

*Read Your Profile*

This example employs the user profile `"_id"`:

```
$ curl \
 --request GET \
 --cacert ca-cert.pem \
 --user newuser:password \
 --silent \
 "https://localhost:8443/rest/users/${ID}"
```

*Changing a User's Own User-Defined Email Type*

This example employs the user profile `"_id"`.

Download the JSON patch payload, changeEmailType.json:

```
$ curl \
 --request PATCH \
 --cacert ca-cert.pem \
 --user newuser:password \
 --header "Content-Type: application/json" \
 --data @changeEmailType.json \
 --silent \
 "https://localhost:8443/rest/users/${ID}"

{
  "_id" : "<generated-user-id>",
  "_rev" : "<revision>",
  "_schema" : "examples:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "externalId" : "newuser",
  "userName" : "newuser",
  "name" : {
    "formatted" : "New User",
    "givenName" : "User",
```

```
    "familyName" : "New"
  },
  "emails" : [ {
    "value" : "newuser@example.com",
    "type" : "work",
    "primary" : true
  }, {
    "value" : "newuser@example.org",
    "type" : "home",
    "primary" : false
  } ],
  "phoneNumbers" : [ {
    "value" : "+1 408 555 1212",
    "type" : "work"
  } ]
}
```

*Remove Your Email Address*

This example employs the user profile `"_id"`.

Download the JSON patch payload, removeWorkEmail.json:

```
$ curl \
 --request PATCH \
 --cacert ca-cert.pem \
 --user newuser:password \
 --header "Content-Type: application/json" \
 --data @removeWorkEmail.json \
 --silent \
 "https://localhost:8443/rest/users/${ID}"

{
  "_id" : "<generated-user-id>",
  "_rev" : "<revision>",
  "_schema" : "examples:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "externalId" : "newuser",
  "userName" : "newuser",
  "name" : {
    "formatted" : "New User",
    "givenName" : "User",
    "familyName" : "New"
  },
```

```
  "emails" : [ {
    "value" : "newuser@example.org",
    "type" : "home",
    "primary" : true
  } ],
  "phoneNumbers" : [ {
    "value" : "+1 408 555 1212",
    "type" : "work"
  } ]
}
```

*Add Your Cell Phone Number*

This example employs the user profile `"_id"`.

Download the JSON patch payload, addPersonalCell.json:

```
$ curl \
 --request PATCH \
 --cacert ca-cert.pem \
 --user newuser:password \
 --header "Content-Type: application/json" \
 --data @addPersonalCell.json \
 --silent \
 "https://localhost:8443/rest/users/${ID}"

{
  "_id" : "<generated-user-id>",
  "_rev" : "<revision>",
  "_schema" : "examples:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "externalId" : "newuser",
  "userName" : "newuser",
  "name" : {
    "formatted" : "New User",
    "givenName" : "User",
    "familyName" : "New"
  },
  "emails" : [ {
    "value" : "newuser@example.org",
    "type" : "home",
    "primary" : true
  } ],
  "phoneNumbers" : [ {
```

```
    "value" : "+1 408 555 1212",
    "type" : "work"
  }, {
    "value": "+1 408 555 1234",
    "type": "personal cell"
  } ]
}
```

***Delete Your Profile***

This example employs the user profile `"_id"`:

```
$ curl \
 --request DELETE \
 --cacert ca-cert.pem \
 --user newuser:password \
 --header "Content-Type: application/json" \
 --silent \
 "https://localhost:8443/rest/users/${ID}"
```

## REST API Documentation

API descriptors provide runtime documentation for REST APIs. Requests for API descriptors use the reserved query string parameters, `_api` and `_crestapi`. By default, DS servers do not return descriptors, but respond instead with HTTP status code `501 Not Implemented`.

NOTE

Although it is possible to serve the descriptors at runtime, do not use production servers for this purpose.

Instead, prepare the documentation by reading API descriptors from a server with the same API as production servers. Publish the documentation separately.

Preparing documentation for a Rest2ldap endpoint is an iterative process:

1. Enable API descriptors for the connection handler you use:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
```

```
 --handler-name HTTPS \
 --set api-descriptor-enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

2. Restart the connection handler to take the configuration change into account:

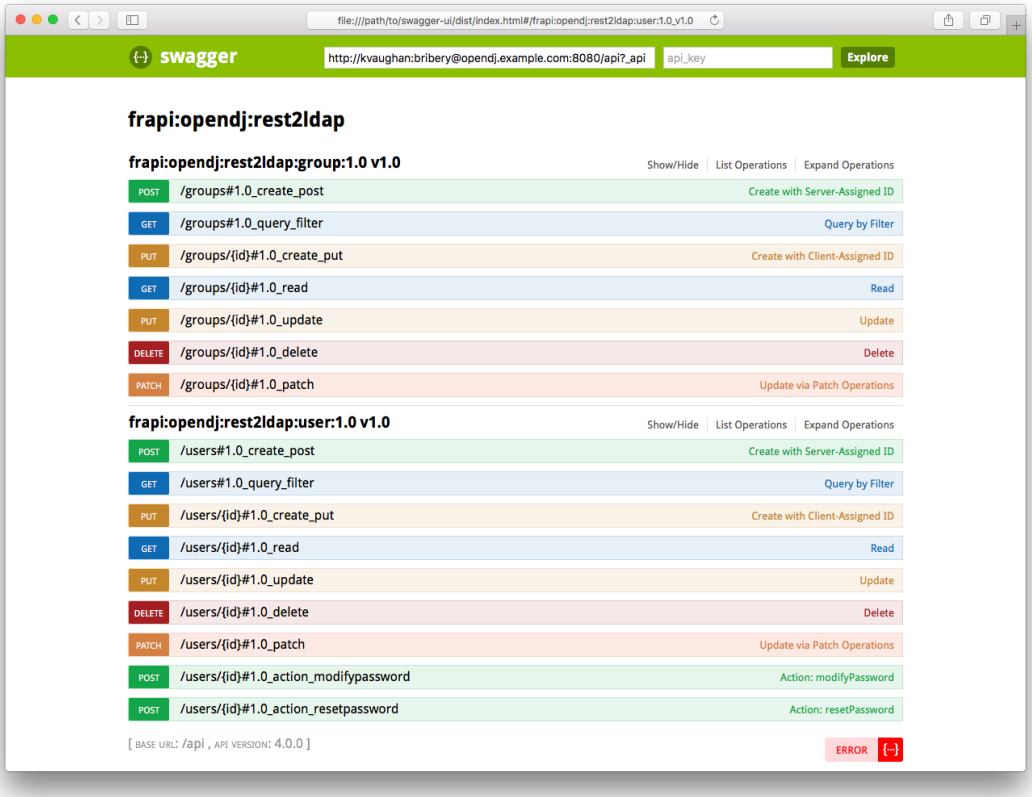
```
$ dsconfig \
 set-connection-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name HTTPS \
 --set enabled:false \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
 set-connection-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name HTTPS \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

3. Configure the API.
4. Run a local copy of a tool for viewing OpenAPI documentation, such as Swagger UI.
5. View the generated documentation through the tool by reading the OpenAPI format descriptor.

   For example, read the descriptor for the `/api` endpoint with a URL such as `https://kvaughan:bribery@localhost:8443/api?_api` for directory data, or `https://admin:password@localhost:8443/admin?_api` for the server configuration.

The following screenshot shows example documentation:




If your browser does not display the generated documentation, disable CORS settings. See your browser's documentation or search the web for details.

6. Update the API configuration.

7. Force the Rest2ldap endpoint to reread the updated configuration file:

```
$ dsconfig \
 set-http-endpoint-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:false \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt

$ dsconfig \
 set-http-endpoint-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
```

```
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin \
 --no-prompt
```

8. Edit the descriptor.
9. Publish the final descriptor alongside your production service.

## REST to LDAP Reference

DS software offers these alternatives for HTTP access to directory data:

| | REST to LDAP Gateway | DS Server |
|---|---|---|
| Implementation | The gateway is a servlet that connects to remote LDAP server(s). | The DS server exposes RESTful HTTP APIs to its directory data. |
| Base Configuration Directory | `WEB-INF/classes` | `opendj/config` |
| Connection Configuration | A single configuration file defines how the gateway connects and authenticates to LDAP servers.<br><br>For details, see Gateway LDAP Connections. | The server uses an internal connection.<br><br>Identity mappers define how HTTP user identities map to LDAP user identities. For details, see Identity Mappers. |
| Gateway Configuration | A single configuration file defines which LDAP features the gateway uses.<br><br>For details, see Gateway LDAP Features. | The server configuration defines which LDAP features are enabled. |
| API Configuration | In both cases, one or more configuration files define the HTTP APIs that map JSON resources to LDAP entries.<br><br>For details, see API Configuration. | |

Unlike standard JSON, REST to LDAP JSON configuration files permit `//` -style comments.

## Gateway LDAP Connections

A single `config.json` file defines how the gateway connects and authenticates to LDAP servers. The top-level fields are:

```
{
  // Secure connections to remote LDAP servers.
  "security": {},
  // Connect and authenticate to remote LDAP servers.
  "ldapConnectionFactories": {},
  // Set authorization policies for access to directory data.
  "authorization": {}
}
```

### *Security*

The `"security"` field covers the parameters for secure connections to remote servers:

▼ More information

```
{
  "security": {
    // Specifies the policy for trusting server certificates
exchanged
    // during SSL/StartTLS negotiation. This setting and the
following
    // trust policy settings will be ignored if there is no
connection
    // security. Acceptable values are:
    //
    // "trustAll" - blindly trust all server certificates (not
secure)
    // "jvm"      - only certificates signed by the authorities
    //              associated with the host JVM will be
accepted (default)
    // "file"     - use a file-based trust store for validating
    //              certificates. This option requires the
following
    //              "fileBasedTrustManager*" settings to be
configured.
    //
    "trustManager": "jvm",
```

```
    // File based trust manager configuration (see above).
    "fileBasedTrustManagerType": "JKS",
    "fileBasedTrustManagerFile": "/path/to/truststore",
    "fileBasedTrustManagerPasswordFile": "/path/to/pinfile",

    // Specifies the key-manager for TLS client authentication
    // (mutual TLS, mTLS) against the remote server.
    // Acceptable values are:
    //
    // "none"     - disables TLS client authentication,
    //              no client certificates will be used.
(default)
    // "jvm"      - use the JVM's default keystore
    //              for retrieving client certificates.
    // "file"     - use a file-based key store
    //              for retrieving client certificates.
    // "pkcs11"   - use a PKCS#11 token
    //              for retrieving client certificates.
    "keyManager": "none",

    // Keystore based key manager configuration (see above).
    "fileBasedKeyManagerType": "JKS",
    "fileBasedKeyManagerFile": "/path/to/keystore",
    "fileBasedKeyManagerPasswordFile": "/path/to/pinfile",

    // PKCS11 based key manager configuration
    "pkcs11KeyManagerPasswordFile": "/path/to/pinfile"
  }
}
```

### *LDAP Connection Factories*

The `"ldapConnectionFactories"` field configures connection and authentication to remote servers:

▼ More information

```
{
  "ldapConnectionFactories": {
    // Unauthenticated connections used for performing bind
requests.
    "bind": {
      // Indicates whether LDAP connections should be secured
using
```

```
      // SSL or StartTLS. Acceptable values are:
      //
      // "none"     - use plain LDAP connections (default)
      // "ssl"      - secure connection using LDAPS
      // "startTLS" - secure connection using LDAP+StartTLS
      //
      "connectionSecurity": "none",

      // This alias references a client SSL key-pair which will
be used
      // for performing client authentication (mutual TLS or
mTLS) between
      // between this gateway and the remote LDAP server.
      "sslCertAlias": "client-cert",

      // Re-usable pool of 24 connections per server.
      "connectionPoolSize": 24,

      // Check pooled connections are alive every 30 seconds
with a 500ms
      // heart beat timeout.
      "heartBeatIntervalSeconds": 30,
      "heartBeatTimeoutMilliSeconds": 500,

      // The preferred load-balancing pool.
      "primaryLdapServers": [{
          "hostname": "localhost",
          "port": 1389
      }],
      // The fail-over load-balancing pool (optional).
      "secondaryLdapServers": [
          // Empty.
      ]
    },

    // Authenticated connections which will be used for searches
during
    // authentication and proxied operations (if enabled). This
factory
    // will re-use the server "bind" configuration.
    "root": {
      "inheritFrom": "bind",

      // Defines how authentication should be performed. Only
simple
```

```
      // and SASL/External authentication are supported at the
moment.
      // If the OAuth 2.0 authorization policy is configured
below,
      // then the directory service must be configured
      // to allow the user configured here to perform proxied
authorization.
      "authentication": {
        // The type of LDAP bind request to use, which must be
"simple"
        // (the default), "sasl-scram" or "sasl-external".
        "policy": "simple",

        "simple": {
            "bindDn": "uid=admin",
            "bindPassword": "password"
        },

        // SASL SCRAM mechanisms require the user's password
        // to be stored using a compatible storage scheme.
        "sasl-scram": {
            // The SCRAM mechanism name, which defaults to
SCRAM-SHA-256.
            "scramMechanism": "SCRAM-SHA-256",
            "bindDn": "uid=admin",
            "bindPassword": "password"
            // Optionally: "scramMinIterations": 2048
            // to reduce the accepted minimum number of
iterations.
        }

        // SASL/External does not have any configurable
parameters.
      }
    }
  }
}
```

### *Authorization*

The `"authorization"` field sets authorization policies for access to directory data. It has the following top-level fields:

```
{
  "authorization": {
    // Authorization policies to use. One of "anonymous", "basic",
or "oauth2".
    "policies": [],
    // Perform all operations using a pre-authorized connection.
    "anonymous": {},
    // Use HTTP Basic authentication's information to bind to the
LDAP server.
    "basic": {},
    // Use an OAuth2 authorization method.
    "oauth2": {}
  }
}
```

The `"anonymous"` object has the following settings:

▼ More information

```
{
  "anonymous": {
    // Specify the connection factory to use to perform LDAP
operations.
    // If missing, the "root" factory will be used.
    "ldapConnectionFactory": "root"
  }
}
```

The `"basic"` object has the following settings:

▼ More information

```
{
  "basic": {
    // Indicates whether the filter should allow alternative
authentication
    // and, if so, which HTTP headers it should obtain the
username and
    // password from.
    "supportAltAuthentication": true,
    "altAuthenticationUsernameHeader": "X-OpenIDM-Username",
    "altAuthenticationPasswordHeader": "X-OpenIDM-Password",

    // Define which LDAP bind mechanism to use
    // Supported mechanisms are "simple", "sasl-plain", "sasl-
```

```
scram", or "search"
    "bind": "search",

    // Bind to the LDAP server using the DN built from the HTTP
Basic's username
    "simple": {
      // Connection factory used to perform the bind operation.
      // If missing, "bind" factory will be used.
      "ldapConnectionFactory": "bind",

      // The Bind DN Template containing a single {username},
      // which will be replaced by the authenticating user's
name.
      // (For example: uid=
{username},ou=People,dc=example,dc=com)
      // If missing, "{username}" is used.
      "bindDnTemplate": "uid=
{username},ou=People,dc=example,dc=com"
    },

    // Bind to the LDAP server using a SASL Plain bind request
    "sasl-plain": {
      // Connection factory used to perform the bind operation.
      // If missing, "bind" factory will be used.
      "ldapConnectionFactory": "bind",

      // Authorization identity template containing a single
{username},
      // which will be replaced by the authenticating user's
name.
      // (For example: u:{username})
      "authzIdTemplate": "u:{username}"
    },

    // Bind to the LDAP server using a SASL SCRAM bind request.
    // SASL SCRAM mechanisms require the user's password to be
stored
    // using a compatible storage scheme.
    "sasl-scram": {
      // Connection factory used to perform the bind operation.
      // If missing, "bind" factory will be used.
      "ldapConnectionFactory": "bind",

      // The SCRAM mechanism name, which defaults to SCRAM-SHA-
256.
```

```
      "scramMechanism": "SCRAM-SHA-256",

      // Authorization identity template containing a single
{username}
      // which will be replaced by the authenticating user's
name.
      // (For example: u:{username})
      "authzIdTemplate": "u:{username}"
    },

    // Bind to the LDAP server using the resulting DN of a
search request.
    "search": {
      // Connection factory used to perform the search
operation.
      // If missing, "root" factory will be used.
      "searchLdapConnectionFactory": "root",

      // Connection factory used to perform the bind operation.
      // If missing, "bind" factory will be used.
      "bindLdapConnectionFactory": "bind",

      // The {username} filter format parameters will be
substituted
      // with the client-provided username,
      // using LDAP filter string character escaping.
      "baseDn": "ou=people,dc=example,dc=com",
      "scope": "sub", // Or "one".
      "filterTemplate": "(&(uid={username})
(objectClass=inetOrgPerson))"
    }
  }
}
```

The `"oauth2"` object has the following settings:

▼ More information

```
{
  "oauth2": {
    // Access tokens associated realm.
    // This attribute is optional and has a string syntax.
    "realm": "myrealm",

    // Defines the list of required scopes required to access
```

```
the service.
    // This field is required and cannot be empty.
    "requiredScopes": [ "read", "write", "uid" ],

    // Specify the resolver to use to resolve OAuth2 access
token.
    // This attribute is required and its value must be one of
"openam", "rfc7662", "cts".
    // Note that the JSON object corresponding to this attribute
value must be present
    // and well formed in the "oauth2" JSON attribute.
    "resolver": "openam",

    // Configures caching of access token introspection results.
    // This attribute is optional, if it is not present, no
token caching
    // will be performed.
    "accessTokenCache": {
      // Indicates whether the access token caching should be
used.
      // This attribute is optional (default value is false)
      // and must have a boolean syntax.
      "enabled": false,

      // Specifies the maximal caching duration for an access
token.
      // Once this delay is over, token will be refreshed from
an access token resolver
      // (see "oauth2/resolver").
      // This attribute is optional, its default value is "5
minutes".
      // The duration syntax supports all human readable
notations from day
      // ("days", "day", "d") to nanosecond ("nanoseconds",
"nanosecond", "nanosec",
      // "nanos", "nano", "ns")
      // Any negative or zero values are incorrect.
      "cacheExpiration": "5 minutes"
    },

    // The OpenAM access token resolver configuration.
    // This attribute must be present if the "oauth2/resolver"
is equal to "openam".
    // If "oauth2/resolver" is set to another resolver, this
attribute will be ignored.
```

```
    "openam": {
      // Defines the OpenAM endpoint URL where the request
should be sent.
      // This attribute is required and must have a string
syntax.
      "endpointUrl":
"http://openam.example.com:8080/openam/oauth2/tokeninfo",

      // This alias points at an existing certificate that is
used for TLS authentication
      // for secure communication between this gateway
      // and the OpenAM access-token resolver.
      "sslCertAlias": "client-cert",

      // The default authzIdTemplate demonstrates how an
authorization DN
      // may be constructed from the "uid" field in the
following example
      // OpenAM tokeninfo response:
      // {
      //     "scope":["uid"],
      //     "realm":"/",
      //     "expires_in":45,
      //     "uid" : "bjensen",
      // }
      // This attribute is required and has a string syntax.
      // It must start with either 'dn:' or 'u:'.
      "authzIdTemplate": "dn:uid=
{uid},ou=People,dc=example,dc=com"
    },

    // The RFC-7662 (see https://tools.ietf.org/html/rfc7662)
    // access token resolver configuration.
    // This attribute must be present if the "oauth2/resolver"
is equal to "rfc7662".
    // If "oauth2/resolver" is set to another resolver, this
attribute will be ignored.
    "rfc7662": {
      // Defines the token introspection endpoint URL where the
request should be sent.
      // This attribute is required and must have a string
syntax.
      "endpointUrl":
"http://openam.example.com:8080/openam/oauth2/myrealm/introspect
",
```

```
      // This alias points at an existing certificate that is
used for TLS authentication
      // for secure communication between this gateway and the
introspection
      // access-token resolver.
      "sslCertAlias": "client-cert",

      // Token introspect endpoint requires authentication.
      // It should support HTTP basic authorization
      // (a base64-encoded string of clientId:clientSecret)
      // These attributes are mandatory.
      "clientId": "client_id",
      "clientSecret": "client_secret",

      // The default authzIdTemplate demonstrates how an
authorization DN
      // may be constructed from the "username" field in the
following example
      // introspect response:
      // {
      //     "active": true,
      //     "token_type": "access_token",
      //     "exp": 3524,
      //     "username" : "bjensen",
      // }
      // This attribute is required and has a string syntax.
      // It must start with either 'dn:' or 'u:'.
      "authzIdTemplate": "dn:uid=
{username},ou=People,dc=example,dc=com"
    },

    // The CTS access token resolver.
    // This attribute must be present if the "oauth2/resolver"
is equal to "cts".
    // If "oauth2/resolver" is set to another resolver, this
attribute will be ignored.
    // Note: You can use {userName/0} in authzIdTemplate
configuration to access
    // user id from the default CTS access token content config.
    "cts": {
      // The connection factory to use to access CTS.
      // This attribute must reference a connection factory
      // defined in the "ldapConnectionFactories" section.
      // Default value: "root"
```

```
      // (That is, the "root" connection factory will be used to
access the CTS).
      "ldapConnectionFactory": "root",

      // The access token base DN.
      // This attribute is required and must have a string
syntax.
      "baseDn": "ou=famrecords,ou=openam-
session,ou=tokens,dc=example,dc=com",

      // The default authzIdTemplate demonstrates how an
authorization DN
      // may be constructed from the "userName" field in the
following example
      // CTS access token entry:
      // {
      //     "active": true,
      //     "tokenName": ["access_token"],
      //     "exp": [3524],
      //     "userName" : ["bjensen"],
      // }
      // This attribute is required and has a string syntax.
      // It must start with either 'dn:' or 'u:'.
      "authzIdTemplate": "dn:uid=
{userName/0},ou=People,dc=example,dc=com"
    },

    // ONLY FOR TESTING: A file-based access token resolver.
    // This attribute must be present if the "oauth2/resolver"
is equal to "file".
    // If "oauth2/resolver" is set to another resolver, this
attribute will be ignored.
    "file": {
      // Directory containing token files.
      // You can test the rest2ldap OAuth2 authorization support
      // by providing JSON token files under
      // the directory set in the configuration below.
      // File names must be equal to the token strings.
      // The file content must a JSON object with the following
attributes:
      // 'scope', 'expireTime' and all the field(s) needed to
resolve the authzIdTemplate.
      "folderPath": "/path/to/test/folder",

      // The default authzIdTemplate demonstrates an
```

```
authorization DN constructed
      // from the "uid" field in a fake token file:
      // {
      //     "scope": ["read", "uid", "write"],
      //     "expireTime": 1961336698000,
      //     "uid": "bjensen"
      // }
      // This attribute is required and has string syntax.
      // It must start with either 'dn:' or 'u:'.
      "authzIdTemplate": "dn:uid=
{uid},ou=People,dc=example,dc=com"
    }
  }
}
```

## Gateway LDAP Features

A single `rest2ldap/rest2ldap.json` file defines the LDAP features that the gateway uses. The settings have the following defaults:

```
{
  "useMvcc": true,
  "mvccAttribute": "etag",
  "readOnUpdatePolicy": "controls",
  "useSubtreeDelete": true,
  "usePermissiveModify": true,
  "useServerSideSortForJson": true,
  "returnNullForMissingProperties": false,
  "localSortMaxEntries" : 1000
}
```

▼ More information

| Field | Description |
|---|---|
| `"useMvcc"` | Whether the gateway supports multi-version concurrency control (MVCC).<br><br>If `true`, specify the `"mvccAttribute"`. |

| Field | Description |
| --- | --- |
| `"mvccAttribute"` | The LDAP attribute to use for MVCC.<br><br>This lets a client application check whether this is the correct version of the resource with the header:<br><br>`If-Match: mvcc-value` |
| `"readOnUpdatePolicy"` | Specifies the policy for reading an entry after addition, after modification, and before deletion.<br><br>One of the following:<br><br>• `"controls"` : Use RFC 4527 read-entry controls to reflect the state of the resource on update.<br><br>The remote LDAP servers must support RFC 4527.<br>• `"disabled"` : Do not read the entry or return the resource on update.<br>• `"search"` : Perform an LDAP search to retrieve the entry after addition, after modification, and before deletion.<br><br>The JSON resource returned might differ from the updated LDAP entry. |

| Field | Description |
| --- | --- |
| `"useSubtreeDelete"` | Whether to use the LDAP Subtree Delete request control (OID: `1.2.840.113556.1.4.805`) for LDAP delete operations resulting from delete operations on resources.<br><br>Client applications deleting resources with children must have access to use the control.<br><br>When `true`, the gateway attempts to use the control. It falls back to searching for and deleting children if the server rejects the request. |
| `"usePermissiveModify"` | Whether to use the LDAP Permissive Modify request control (OID: `1.2.840.113556.1.4.1413`) for the LDAP modifications corresponding to PATCH and PUT requests.<br><br>Set this to `false` when remote LDAP servers do not support the control. |
| `"useServerSideSortForJson"` | Whether to use the LDAP Server-Side Sort request control (OID: `1.2.840.113556.1.4.473`) to request that the server sort the results before returning them.<br><br>When `false`, the gateway sorts search results locally. In this case, set `localSortMaxEntries`, which effectively limits the maximum `_pageSize` that the gateway accepts. |
| `"returnNullForMissingProperties"` | Whether missing (unmapped) JSON properties should be included in JSON resources. By default, a REST to LDAP mapping omits JSON fields for LDAP attributes that have no values. |

| Field | Description |
| --- | --- |
| `"localSortMaxEntries"` | The maximum number of entries supported by the local sort mechanism.<br><br>When a request includes a `_sortKey` parameter, the gateway does the following. If the `_sortKey` parameter targets:<br><br>• A normal LDAP attribute, the gateway includes a server-side sort request to the LDAP server.<br>• A JSON syntax LDAP attribute, the action depends on the `useServerSideSortForJson` setting.<br>• An attribute that is in a referenced entry, the gateway sorts the results locally. |

## API Configuration

Mapping files define APIs by defining how JSON resources map to LDAP entries. REST to LDAP lets you define multiple APIs, and multiple versions for each API.

A mapping file name has the form `rest2ldap/endpoints/base-path/root-resource.json`:

- The base-path must match the Rest2ldap endpoint `base-path` setting in the DS server configuration.
- The root-resource matches the root resource name in the API's `"resourceTypes"`.
- Each file defines a single version of the API.

The sample API file `rest2ldap/endpoints/api/example-v1.json` has the following structure:

```
{
  "version": "1.0",          // (Optional) version for this API.
  "resourceTypes": {         // (Required) resources for this API.
    "example-v1": {          // (Required) root resource type. Name
must match file basename.
      "subResources": {      // (Required) The base path, /api, is
implicit.
```

```
        "users": {},        // All resource collections, such as
/api/users/ and
        "groups": {}        // /api/groups/ are explicitly defined
here.
      }
    },

    // The sample also defines the resource types used by the
root's "subResources".
    // This is optional, but critical to readability of the
"subResources" definition.
    // Keep resource type names unique to avoid clashes with
definitions elsewhere.
    // See the full text of the sample to view how inheritance
works.
    "frapi:opendj:rest2ldap:object:1.0": {},    // Parent type of
all objects.
    "frapi:opendj:rest2ldap:user:1.0": {},      // Basic user
type, parent of
    "frapi:opendj:rest2ldap:posixUser:1.0": {}, // user with uid,
gid, home dir.
    "frapi:opendj:rest2ldap:group:1.0": {}      // Basic group
type.
  }
}
```

The example API defines the following resource collections:

- `"users"`
- `"groups"`

For reference information, see Subresources.

The example API defines the following resource types:

- `"frapi:opendj:rest2ldap:object:1.0"`
- `"frapi:opendj:rest2ldap:user:1.0"`
- `"frapi:opendj:rest2ldap:posixUser:1.0"`
- `"frapi:opendj:rest2ldap:group:1.0"`

For reference information, see Resource Types.

### *Version*

The optional `"version"` field specifies the version of the root resource of the API. For multiple versions of the same API, use multiple mapping files.

▼ More information

Valid version strings include:

- `"*"` (Default, no version specified.)
- `"integer"`
- `"integer.integer"`

Each integer is a positive decimal integer.

A client application requests a specific version by setting the request header:

```
Accept-API-Version: resource=version
```

If more than one version of the API is available, but the client does not specify a version header, REST to LDAP uses the latest version of the API.

### *Subresources*

The `"subResources"` object specifies the API under the current path.

A `"subResources"` object has the following fields, shown with their default values:

```
{
  "name": {
    "type": "collection || singleton",  // Required
    "dnTemplate": "",
    "glueObjectClasses": [],
    "isReadOnly": false,
    "namingStrategy": {},
    "resource": ""
  }
}
```

▼ More information

| Field | Description |
| --- | --- |
| `"type"` | The type, either `"collection"` or `"singleton"`.<br><br>A collection is a container for other resources that clients can create, read, update, delete, patch, and query. When `"type": "collection"`, the definition has these settings:<br><br>`{`<br>`  "type": "collection",`<br>`  "namingStrategy": {},  // Required`<br>`  "resource": "",  // Required`<br>`  "dnTemplate": "",  // Optional`<br>`  "glueObjectClasses": [],  // Optional`<br>`  "isReadOnly": false  // Optional`<br>`}`<br><br>A singleton is a resource with no children. When `"type": "singleton"`, the definition has these settings:<br><br>`{`<br>`  "type": "singleton",`<br>`  "resource": "",  // Required`<br>`  "dnTemplate": "",  // Optional`<br>`  "isReadOnly": false  // Optional`<br>`}` |

| Field | Description |
| --- | --- |
| `"dnTemplate"` | The relative DN template where the LDAP entries are located.<br><br>If this is an empty string, the LDAP entries are located directly beneath the parent LDAP entry.<br><br>DN templates can use variables in braces `{}`. REST to LDAP substitutes DN template variables with values extracted from the URL template. |
| `"glueObjectClasses"` | One or more LDAP object classes associated with intermediate "glue" entries forming the DN template.<br><br>Required if the DN template contains one or more RDNs. |
| `"isReadOnly"` | Whether this resource is read-only. |

| Field | Description |
| --- | --- |
| `"namingStrategy"` | How to map LDAP entry names to JSON resources.<br><br>LDAP entries mapped to JSON resources must be immediate subordinates of the `"baseDn"`.<br><br>REST to LDAP supports the following naming strategies:<br><br>• `"type": "clientDnNaming"`: The RDN and resource ID are both derived from a single user attribute.<br><br>In the following example, the `uid` attribute is the RDN. Its value is the JSON resource ID:<br><br>`{ "namingStrategy": { "type": "clientDnNaming", "dnAttribute": "uid" } }`<br><br>• `"type": "clientNaming"`: The RDN and resource ID are derived from separate user attributes.<br><br>In the following example, the RDN attribute is `uid`. The JSON resource ID is the value of the `mail` attribute:<br><br>`{ "namingStrategy": { "type": "clientNaming", "dnAttribute": "uid", "idAttribute": "mail" } }` |

| Field | Description |
| --- | --- |
| | • `"type": "serverNaming"` : The RDN is derived from a user attribute and the resource ID from an operational attribute.<br><br>In the following example, the RDN attribute is `uid` . The JSON resource ID is the value of the `entryUUID` operational attribute:<br><br>`{ "namingStrategy": { "type": "serverNaming", "dnAttribute": "uid", "idAttribute": "entryUUID" } }` |
| `"resource"` | The resource type name of the subresource.<br><br>A collection can contain objects of different types as long as all types inherit from the same super type. In that case, set `resource` to the super type name. |

### *Resource Types*

The required `"resourceTypes"` object maps resource type names to resource type definitions.

One of the resource type names must match the basename of the mapping file. This resource is referred to as the *root resource* of the API.

▼ More information

You can reuse a resource type name when specifying the same definition in different mapping files. For example, all mapping files might define the abstract base type `frapi:opendj:rest2ldap:object:1.0` , and use this everywhere as the `superType` for other resource types.

If the definitions differ, however, you must use a different resource type name for the changed resource type. For example, if you add a new user type in another API based on `frapi:opendj:rest2ldap:user:1.0`, but with a different definition. If you still use the original resource type in your APIs, you must also use a different name for your new user type.

Examples in the sample mapping file use the namespace prefix `frapi:opendj:rest2ldap:`. You may use any string allowed in JSON.

REST to LDAP does not provide a mechanism for inheriting resource types between mapping files. To reuse a resource type from another mapping file, manually copy its name and definition, taking care to change the name if you change the definition.

A resource type definition object has the following fields. All fields are optional, and have the default values shown here:

```
{
  "unique-name": {
    "properties": {},
    "subResources": {},
    "isAbstract": false,
    "superType": "",
    "objectClasses": [],
    "supportedActions": [],
    "includeAllUserAttributesByDefault": false,
    "excludedDefaultUserAttributes": []
  }
}
```

▼ More information

| Field | Description |
|---|---|
| `"properties"` | Map of property names to property definitions.<br><br>For details, see Properties. |

| Field | Description |
| --- | --- |
| `"subResources"` | Map of subresource names to subresource definitions.<br><br>The names are URL path templates, setting the relative path where the subresources are located. URL path templates can set variables in braces `{}`. For example, suppose LDAP entries for devices are under the following base DNs:<br><br>• `ou=others,ou=devices,dc=example,dc=com`<br>• `ou=pcs,ou=devices,dc=example,dc=com`<br>• `ou=phones,ou=devices,dc=example,dc=com`<br>• `ou=tablets,ou=devices,dc=example,dc=com`<br><br>You can define a DN template, `ou={type},ou=devices,dc=example,dc=com`.<br><br>Given the paths relative paths:<br><br>• `devices/others`<br>• `devices/pcs`<br>• `devices/phones`<br>• `devices/tablets`<br><br>REST to LDAP substitutes `{type}` with the last path element to resolve the DN template and locate the entries in the correct LDAP organizational unit.<br><br>For details, see Subresources. |
| `"isAbstract"` | Whether this is an abstract resource type used only for inheritance. |

| Field | Description |
| --- | --- |
| `"superType"` | The resource type that this resource type extends.<br><br>Resource types that extend another type inherit properties and subresource definitions. |
| `"objectClasses"` | LDAP object classes names for the LDAP entries underlying the JSON resource.<br><br>On creation, REST to LDAP adds these to the object classes on the LDAP entry.<br><br>The LDAP object classes are never visible in the JSON resource. |

| Field | Description |
| --- | --- |
| `"supportedActions"` | Names of the DS-specific ForgeRock® Common REST actions that this resource type supports.<br><br>Action names must match the actions allowed on the resource in the underlying implementation:<br><br>• `"accountUsability"` : Return account usability information.<br><br>For an example, see Account Usability Action.<br><br>• `"create"` : Create a resource by HTTP POST.<br><br>This action is implicitly supported. Do not include it in the list.<br><br>For an example, see Create (HTTP POST).<br><br>• `"modifyPassword"` : Change one's own password given the old and new password.<br><br>For an example, see Change Your Password.<br><br>• `"resetPassword"` : Reset a user's password to a generated value.<br><br>For an example, see Reset a Password. |
| `"includeAllUserAttributesByDefault"` | Whether to include all LDAP user attributes as properties of the JSON resource.<br><br>When `true` , the JSON field names match the LDAP attribute names. |
| `"excludedDefaultUserAttributes"` | LDAP user attributes to exclude when `"includeAllUserAttributesByDefault": true` . |

### *Properties*

The `"properties"` object specifies how the JSON resource maps to the underlying LDAP entries.

A `"properties"` object has the following fields, shown with their default values:

```
{
  "name": {
    "type": "See the list below.", // Required
    "baseDn": "",
    "defaultJsonValue": "",
    "extensibleJsonOrderingMatchingRule": "",
    "isBinary": false,
    "isMultiValued": false,
    "isRequired": false,
    "jsonQueryEqualityMatchingRule": "caseIgnoreJsonQueryMatch",
    "ldapAttribute": "name",
    "mapper": {},
    "optionalJsonPropertyName": "",
    "primaryKey": "",
    "propertyName": "",
    "resourcePath": "",
    "schema": "",
    "searchFilter": "(objectClass=*)",
    "value": "",
    "writability": "readWrite"
  }
}
```

The `"type"` must be one of the following:

- `"constant"`
- `"json"`
- `"object"`
- `"reference"`
- `"reverseReference"`
- `"resourceType"`
- `"simple"`

▼ More information

| Field | Description |
| --- | --- |
| `"type"` | The type, which determines the fields the object has:<br><br>*`"constant"`*<br>A fixed value:<br><br>`{ "name": { "type": "constant", "value": "A valid JSON value." } }`<br><br>*`"json"`*<br>A `Json` syntax LDAP attribute:<br><br>`{ "type": "json", "ldapAttribute": "name", // Required "baseDn": "", "defaultJsonValue": "", "extensibleJsonOrderingMatchingRule": "", "isBinary": false, "isMultiValued": false, "isRequired": false, "jsonQueryEqualityMatchingRule": "caseIgnoreJsonQueryMatch", "schema": "", "writability": "readWrite" }`<br><br>The `"ldapAttribute"` must be a `Json` syntax LDAP attribute.<br><br>*`"object"`*<br>An object with its own mapping:<br><br>`{ "type": "object", "properties": {} }`<br><br>*`"reference"`* |

| Field | Description |
| --- | --- |
| | An LDAP entry found by reference.<br><br>This is useful with LDAP attributes that take another entry's DN, such as `manager`, and (group) `member`.<br><br>The mapping can define the location of the reference by:<br><br>• Resource path, the path to another JSON resource:<br><br>`{ "type": "reference", "resourcePath": "", // Required "isMultiValued": false, "isRequired": false, "ldapAttribute": "name", "optionalJsonPropertyName": "", "searchFilter": "(objectClass=*)", "writability": "readWrite" }`<br><br>When REST to LDAP reads the reference, it returns the `_id` and `_rev` fields by default.<br><br>• Base DN, the LDAP base DN under which to find entries referenced by the JSON resource:<br><br>`{ "type": "reference", "baseDn": "", // Required "mapper": {}, // Required "primaryKey": "", // Required "isMultiValued": false, "isRequired": false, "ldapAttribute": "name", "optionalJsonPropertyName": "", "searchFilter": "(objectClass=*)", "writability": "readWrite" }` |

| Field | Description |
| --- | --- |
| | When REST to LDAP reads the reference, it returns all fields by default.<br><br>REST to LDAP follows these rules to determine field types for references:<br><br>• If the LDAP attribute is defined in the LDAP schema, then the REST to LDAP mapping uses the most appropriate type in JSON.<br><br>For example, numbers appear as JSON numbers, and booleans as booleans.<br><br>• If the LDAP attribute only has one value, it maps to a scalar.<br><br>• If the LDAP attribute has multiple values, it maps to an array.<br><br>*"reverseReference"*<br>An LDAP entry that references this one:<br><br>`{ "type": "reverseReference", // Required: "resourcePath": "", // Required: "propertyName": "", // Optional: "searchFilter": "(objectClass=*)" }`<br><br>The `"propertyName"` is the name of the property that references this resource.<br><br>When REST to LDAP reads the resource, it does not return reverse references by default. Use the `_fields` parameter to explicitly request reverse reference fields.<br><br>*"resourceType"*<br>The name of a resource type defined elsewhere in this mapping file.<br><br>*"simple"*<br>An LDAP attribute: |

| Field | Description |
| --- | --- |
| | <br>```<br>{<br>  "type": "simple",<br>  "ldapAttribute": "name", // Required<br>  "defaultJsonValue": "",<br>  "isBinary": false,<br>  "isMultiValued": false,<br>  "isRequired": false,<br>  "writability": "readWrite"<br>}<br>```<br><br>Use simple mappings where the correspondence between JSON properties and LDAP attributes is one-to-one. |
| `"baseDn"` | The base LDAP DN where REST to LDAP finds entries referenced by the JSON resource.<br><br>Base DN values can be literal values, such as `dc=example,dc=com`. Alternatively, they can use the following notation:<br><br>*`{url-template}`*<br>REST to LDAP replaces this with the literal value used in the request.<br><br>For example, if the path resource path is `/{tenant}/users`, and the base DN is `ou=people,dc={tenant},dc=com`, a request for `/example/users`, references `ou=people,dc=example,dc=com`.<br><br>*`".."`*<br>This is like `..` in a path, meaning "the parent directory."<br><br>Paths are big-endian, whereas DNs are little-endian. To reference a `"resourcePath"`, you write `../../groups`. To reference a `"baseDn"`, you write `ou=groups,..,...`<br><br>Notice a limitation in this reference to group member entries: all group members must be people. The configuration does not support nested groups and members in other locations. |

| Field | Description |
| --- | --- |
| `"defaultJsonValue"` | A JSON value to return if no corresponding LDAP attribute is present. |
| `"extensibleJsonOrderingMatchingRule"` | The JSON ordering matching rule to use when requesting an extensible server-side sort.<br><br>The default rule will ignore case and whitespace when sorting values of JSON fields.<br><br>For a description of the extended server-side sort syntax, see Server-Side Sort. |
| `"isBinary"` | Whether the underlying LDAP attribute holds a binary value, such as a JPEG photo or a digital certificate.<br><br>When `"isBinary": true`, the JSON resource holds the base64-encoded value. For details, see Binary Resources. |
| `"isMultiValued"` | Whether the field can take an array value.<br><br>Most LDAP attributes are multi-valued. A literal-minded mapping from LDAP to JSON would be full of array properties, many with only one value.<br><br>To minimize inconvenience, REST to LDAP returns single value scalars by default, even when the underlying LDAP attribute is multi-valued. By default, the JSON resource gets the first value returned for a multi-valued LDAP attribute.<br><br>When `"isMultiValued": true`, a single value is still returned as a scalar. If the LDAP attribute has multiple values, REST to LDAP returns an array. |
| `"isRequired"` | Whether the LDAP attribute is mandatory and must be provided to create the resource. |

| Field | Description |
| --- | --- |
| `"jsonQueryEqualityMatchingRule"` | When a query filter in the HTTP request uses a JSON path that points to a field in a JSON attribute value, it uses the matching rule specified by this property to compare the query filter with attribute values:<br>• `"caseIgnoreJsonQueryMatch"` : Ignore case when finding matches (default).<br>• `"caseExactJsonQueryMatch"` : Respect case when finding matches. |
| `"ldapAttribute"` | The attribute in the LDAP entry underlying the JSON resource.<br>By default, REST to LDAP assumes the JSON field name matches the LDAP attribute name. |
| `"mapper"` | How the referenced entry content maps to the content of this JSON field.<br>A mapper object can have all the fields described in this table. |

| Field | Description |
| --- | --- |
| `"optionalJsonPropertyName"` | Use this when creating a reference for an attribute that has `NameAndOptionalJSON` syntax.<br><br>REST to LDAP returns optional JSON from the attribute as the value of a property having the name you specify. For example, if you configure `"optionalJsonPropertyName": "_refProperties"`, then the `"_refProperties"` field of the reference contains the optional JSON.<br><br>Suppose the LDAP entry contains an attribute with this syntax, `relationship: {"optional": "JSON"}cn=Some relationship,dc=example,dc=com`. The configuration specifies `"optionalJsonPropertyName": "_refProperties"` in the `relationship` reference. The JSON resource has a relationship field like the following:<br><br>`{ "relationship": { "_id": "Some relationship", "_refProperties": {"optional": "JSON"} } }` |
| `"primaryKey"` | Indicates which LDAP attribute in the mapper holds the primary key to the referenced entry. |
| `"propertyName"` | Name of another field of the JSON resource. |

| Field | Description |
| --- | --- |
| `"resourcePath"` | A path to another JSON resource.<br><br>Resource path values use the following notation:<br><br>*{url-template}*<br>REST to LDAP replaces this with the literal value used in the request.<br><br>*/path*<br>The absolute path to the resource.<br><br>`..`<br>The `..` refers to the relative parent path. Use `..` to reference an arbitrary resource in the same collection.<br><br>For example, according to the default sample API, `/api/users/bjensen` refers to a user, and `/api/groups/Directory Administrators` refers to a group. To refer to Babs Jensen's manager from her account, the resource path is to some other user, `...` To refer to groups Babs belongs to, the resource path is to some group, `../../groups`.<br><br>The following example shows how `..` references a manager entry next to the current entry:<br><br>`{ "manager": { "type": "reference", "resourcePath": ".." } }`<br><br>The following example shows a reference to group member entries:<br><br>`{ "members": { "type": "reference", "resourcePath": "../../users", "ldapAttribute": "uniqueMember", "isMultiValued": true` |

| Field | Description |
| --- | --- |
| | <pre>  }<br>}</pre><br>Notice a limitation in this reference to group member entries: all group members must be people. The configuration does not handle nested groups and members in other locations.<br><br>The following example uses a resource path to define a manager's reports. All users, managers and their reports, are under `users/`:<br><br><pre>{<br>  "reports": {<br>    "type": "reverseReference",<br>    "resourcePath": "..",<br>    "propertyName": "manager"<br>  }<br>}</pre> |
| `"schema"` | Specifies a JSON Schema that applies values of type `"json"`.<br><br>When no schema is specified, REST to LDAP accepts arbitrary JSON values. |
| `"searchFilter"` | The LDAP filter to use when searching for a referenced entry. |
| `"value"` | Use with `"type": "constant"` to specify the constant value. |

| Field | Description |
| --- | --- |
| `"writability"` | Whether the mapping supports updates:<br>• `"createOnly"` : This attribute can be set only when the entry is created. Attempts to update this attribute thereafter result in errors.<br>• `"createOnlyDiscardWrites"` : This attribute can be set only when the entry is created. Attempts to update this attribute thereafter do not result in errors. Instead, the update value is discarded.<br>• `"readOnly"` : This attribute cannot be written. Attempts to write this attribute result in errors.<br>• `"readOnlyDiscardWrites"` : This attribute cannot be written. Attempts to write this attribute do not result in errors. Instead, the value to write is discarded.<br>• `"readWrite"` : This attribute can be set at creation and updated thereafter. |

Was this helpful?